CLEO to pyCLEO: Modernizing the Control Library for Use with the Green Bank Telescope

Document Management Information

- Revision Number: 0.3
- Revision Release Date: 2020-08-31
- Purpose of Revision: Final Draft
- Scope of Revision: All sections of document.

Purpose and Scope

This document describes the current state of development for the pyCLEO application library and provides documentation for its construction and use. Integral to use with the Green Bank Telescope (GBT), pyCLEO aims to satisfy the needs of telescope operators, engineers, programmers, NRAO staff, and observers.

How This Document Is Organized

This document is organized into the following sections:

The Architecture Background provides a brief outline of the CLEO application. Additionally, it will look at the various CLEO features that will be implemented in the new design, pyCLEO, as well as any issues that can easily be rectified as a new program is created from scratch.

The Views section provides detailed descriptions of various aspects of pyCLEO. pyCLEO runs on a Model View Controller (MVC) system. Based on this architecture the Views will be split into; Model View which is a look at the data and where we get it, GUI view which describes the graphical aspects of Qt, the App View which covers the coding aspects of Qt, (both of which fall under the View in MVC), and the Controller View which describes the MgrsApp masterclass.

The Reference section includes an Acronym List and Glossary, as well as Links mentioned throughout the paper to additional background and resources on both CLEO and pyCLEO.

The Appendix section provides notes on smaller developmental endeavors within the larger pyCLEO project that we explored during the course of this experimental phase. Additionally, this section houses future plans, and documentation of our workflow and user feedback. Finally, this section includes a thank you to the many contributors who have just begun the large undertaking of pyCLEO.

Architecture Background

Problem Background

Control Library for Engineers and Operators, or CLEO, is the current application meant for use with the GBT. CLEO was originally developed by Ron Maddalena, Kevin Crump, and

Christine Rebinski and has been used for the past 20 years. The responsibility of maintenance is currently in the hands of the Software Development Division (SDDev) at the Green Bank Observatory (GBO). Written in Tcl/Tk, CLEO has become outdated as the number of people who know and use Tcl drop off. However, the need for a reliable and efficient GBT monitoring application has not waned.

While CLEO is presented as one program it is essentially a package of over 40 applications which can be run from the CLEO Launcher. Each individual application will show the status and values of all the instruments necessary to run the GBT and are essential to observations and instrument management.

Goals and Design Requirements

The goal of pyCLEO is to create a reliable, fast, and sleek application to ease the workload of those working with the GBT. Additionally, it should be written in a more modern and popular language so it can continue to be used for many years to come.

While the creation of a brand-new application is no small undertaking, in the case of pyCLEO we know what the end product will look like. Because of this, design requirements are relatively straight forward and leave us more time to add new, useful, features.

It is important to note that this project is still very much in its developmental phase. See the Appendix for notes on the software development process, future plans, and the current state of development.

Solution Background

Approaches

CLEO works as an MVC system in which Segeste acts as the RPC layer, or Remote Procedure Call layer. This in-between layer connects the software, in this case CLEO, with a manager to acquire the necessary values that are being called upon by the application. Like CLEO, Segeste is written in Tcl/Tk. As Green Bank continues to update their operating systems, Segeste becomes increasingly at risk to break. To surmount this problem the SDDev presents the following solutions:

- Patch/rewrite Segeste so CLEO can run as is
- Rewrite CLEO in an entirely new language using a different RPC layer

Out of the proposed solutions to the problems that are arising with CLEO, the SDDev decided to rewrite CLEO in a different language. This approach makes the most sense considering that until computing deems it necessary to update the OS, there is no impending need for a new program, meaning the team has ample time to fully develop and roll out an entirely new set of applications. Segeste breaking is an impending eventuality and no one currently on the SDDev is confident in Tcl/Tk, so patching would be a difficult and time-consuming task. Essentially, rewriting CLEO in a new programming language is an act to prevent the inevitability of future problems which will stem from continued use of an outdated language.

Upon further discussion, the best language and GUI library combination to use for this task was deemed to be PyQt, a combination of Python and Qt.

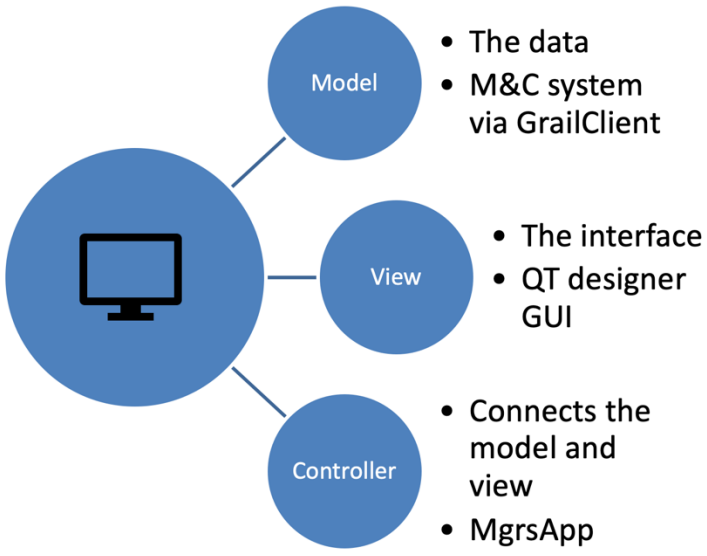
Requirements Coverage

Stakeholder	Use Case	Related View
Operators	Usability in the app	GUI View
Software Developers	Modifiability, patching ease	Related Views, Controller View
Scientists	Scheduling, monitoring the GBT status during observations	GUI View

Views

A Related Views and Context

As mentioned in the previous section, pyCLEO will be a PyQt based application.



The following views will be used to describe how pyCLEO works: Model View, GUI View, and Controller View. The context between each view can be seen in Figure 2.

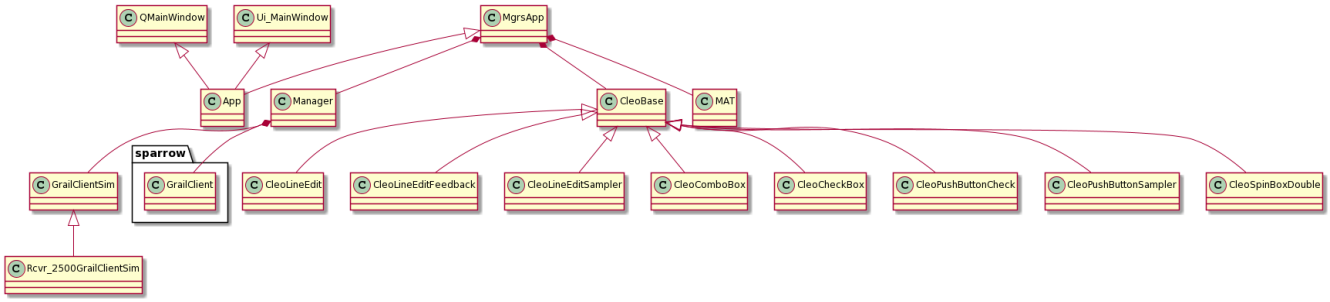
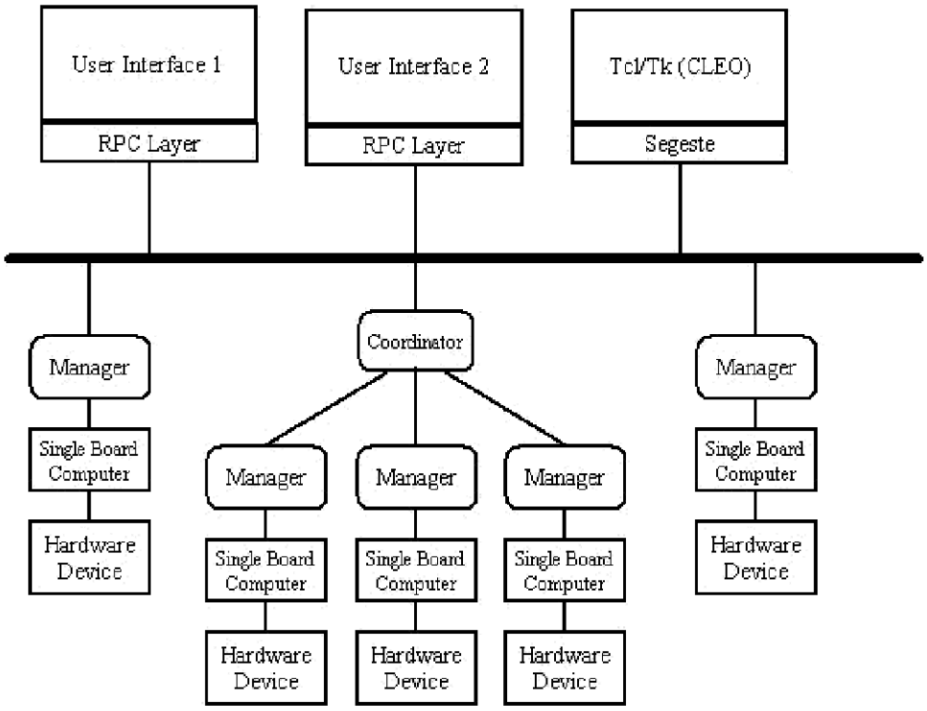


Figure 3 shows all of the related views. While these relationships are discussed in depth in the following three sections, this flow chart illustrates these relationships. Please note that for the rest of the paper our test application used in all examples will be RCVR4_8.

Model View

Primary Presentation



Element Catalog

- Model – The model holds all of the data that is coming in from the hardware device, in pyCLEO the model is called the Grail
- User Interfaces – Digital way to interact with a hardware device, an application
- Tcl/Tk (CLEO) – The current user interface for the GBT
- RPC Layer - Remote Procedure Call Layer, a program can use this to request a service from another system without needing to know how that other system works

- Segeste – The RPC layer for CLEO
- Coordinator – Parent of multiple Managers
- Manager – A piece of software that provides an interface which you can use to monitor and control a piece hardware
- Single Board Computer – The computer which controls a hardware device
- Hardware Device – An instrument used for observations

Architecture Background

Figure 4 displays the path between a piece of hardware and the CLEO application. The software that connects CLEO to the manager (see Glossary) of a hardware device is called Segeste. The program CLEO requests information from the GBT Model, Segeste and then displays that information on a GUI which the user can see and interact with. As discussed in the Approaches section, the Segeste Model is written in Tcl/Tk and the odds that it will break after a software update are ever increasing. Therefore, we have chosen to replace Segeste with Grail.

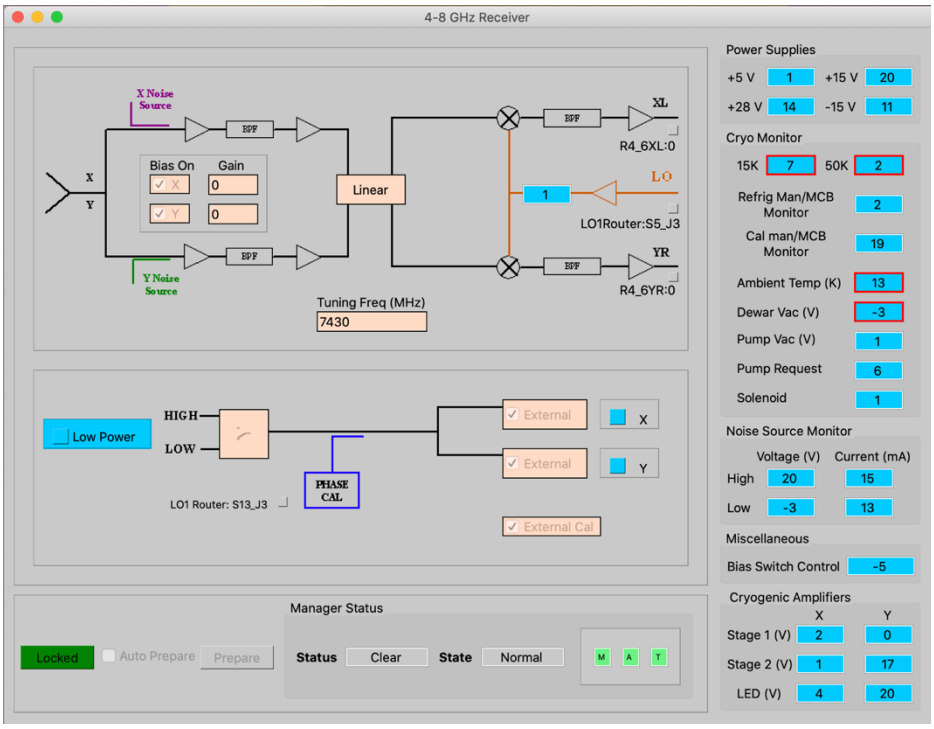
Grail is a M&C system which was already built into the GBO infrastructure. To view and edit the model, pyCLEO can request information from the GrailClient, and, in turn, submit information to the GrailClient to change the model when necessary.

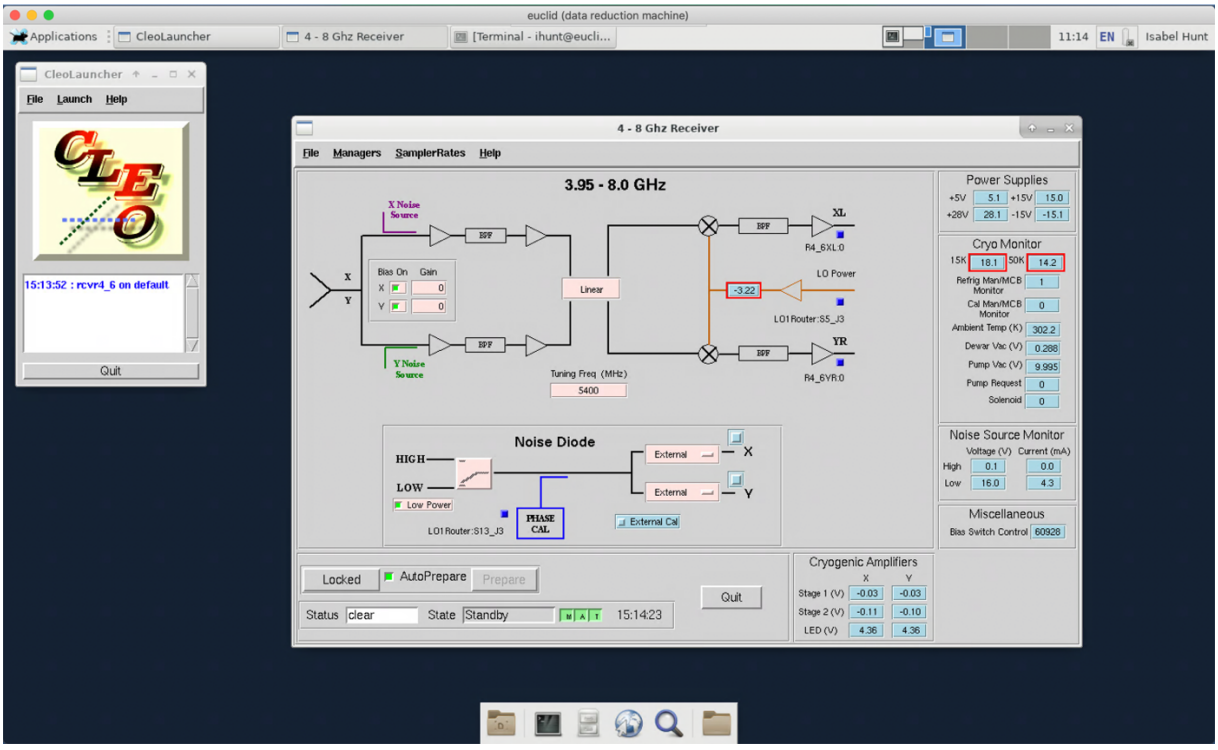
See the following for more details about sending commands with Grail see:

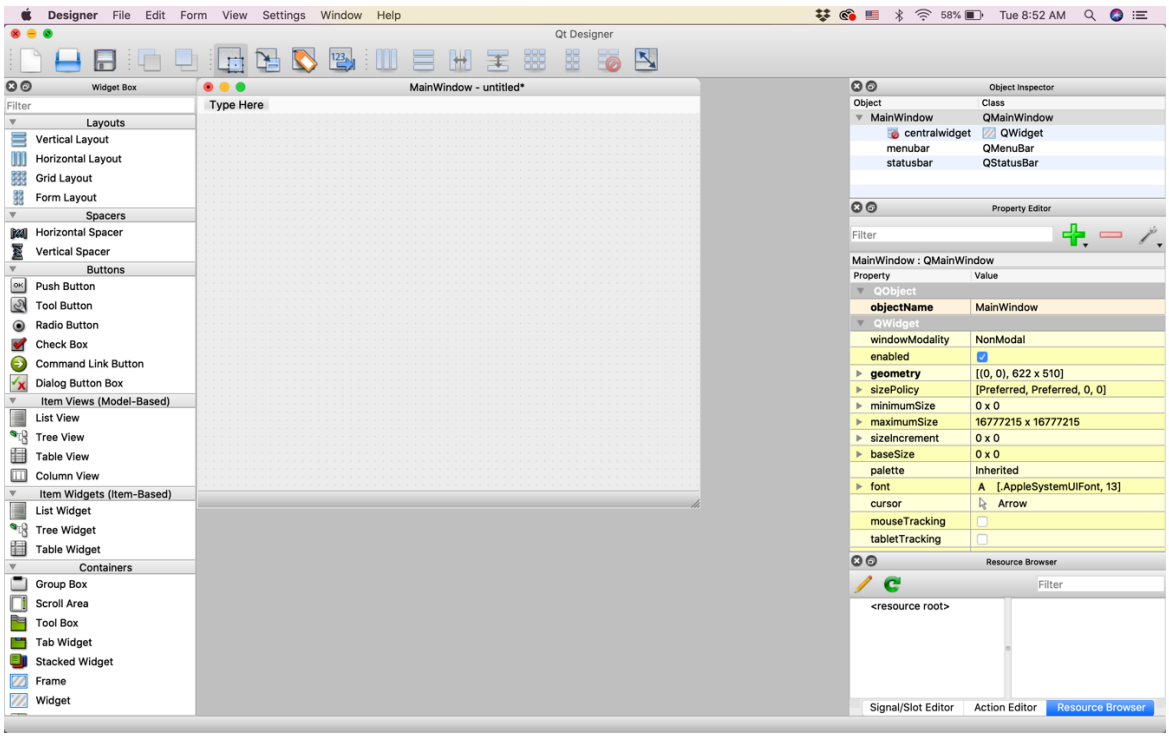
http://docs.greenbankobservatory.org/pyCLEO/howto.html#sending-commands-with-grail

GUI View

Primary Presentation







Element Catalog

- View – The view is the interface which the user sees and interacts with, in pyCLEO this is the Qt window, or GUI which has widgets to present the model
- GUI .ui file – The file type which works with the Qt system
- Qt Designer – The application used to build complicated GUIs
- 4-8 GHz Receiver – One of many instruments on the GBT, for this paper we are looking solely at the rcvr4_8 application.
- CLEO Launcher – Program used to launch any CLEO application

Architecture Background

The view in the case of pyCLEO is the actual application window users will be interacting with. The first explicit design requirements that operators gave to the SDDev was that pyCLEO should look very similar to the original CLEO because it would waste time and effort to learn a new program which looks different but does the same thing as the old program. With that in mind, we recreated the 4-8 GHz Receiver Application; pyCLEO is seen in Figure 5 and the CLEO window is seen in Figure 6 for comparison. We started with the receiver windows because they have a lot of complicated features and we wanted to make sure that PyQt could handle our more complicated design requirements. While the following design techniques work for a majority of applications, every application will have their own individual design requirements and will need to be created on a case by case basis.

Qt Designer is a downloadable application which can be used to design complicated application windows without hard coding every widget (see Glossary). We start with a blank

QMainWindow, change the StyleSheet (see Glossary) background to rgb(201, 201, 201), and add in the necessary widgets. Each of these widgets has their own StyleSheet in Qt Designer which has to be changed to match design choices that were made in the original CLEO, however, because these customizations are general to lots of widgets, the style sheet is changed in the CleoBase Class (see App View). Images can be added to the different widgets as needed. The switches are also considered images; these need to be hard coded in each Managers' associated .py file.

Qt uses a signals and slots construct to connect the model to the widgets on each GUI, for more information on how this system works see the Qt documentation:

https://doc.qt.io/qt-5/signalsandslots.html

To work with this system each widget gets a specific name which will correspond with the Controller. The naming convention is as follows:

For a pyCLEO window with a single Manager: <widget_type>_<parameter/sampler_name>

For a pyCLEO window with multiple Managers:
<widget_type>_<manager>_<parameter/sampler_name>

App View

Element Catalog

- App class – This class holds all of the information for an individual application and connects all aspects of the application to their respective masterclasses.
- CleoBase – A wrapper class which defines how each widget should act in the a given application
- QtMainWindow and Ui_MainWindow - Wrapper classes which are necessary to run a Qt based application and open a window.

Architecture Background

The QtMainWindow and Ui_MainWindow classes are from the PyQt library which is a masterclass to the App class and defines how the main application window will act. The .ui file is instantiated within the App class. The App class holds all of the widget names. This defines them as variables and connects the widgets' slots in Qt to the signals from MgrsApp.

The CleoBase class encompasses all of the widget design and style sheets. Since all of the widgets, in general, act the same the code can be very generalized for every application to use. Cleo Base is a chile of the MgrsApp class.

Controller View

Element Catalog

- Controller – The controller is what connects the model and the view together. In pyCLEO this is called MgrsApp

Architecture Background

MgrsApp is an integral piece of code in the pyCLEO system which uses callbacks to talk to Grail and store/send information to the Manager. MgrsApp is constrained in the sense that it only works best when you are dealing with a single Manager. For example, the 4-8 GHz receiver has a singular Manager associated with it and the rcvr4_8 App class is a direct child of the MgrsApp. However, for a well-used CLEO application, like Device Explorer, in which a user deals with multiple Managers, this system doesn't work as well, it's too low down the hierarchy. Since we are still in the development stage, let us ignore this for now and focus on the other aspects of the Controller View.

As discussed under the App View, the App class defines all of the widgets as variables. MgrsApp uses these names to call the desired values from the Model and present them in the View. To reiterate what is illustrated in Figure 2, MgrsApp acts as the link between the data provided to us by Grail, and the presentation of that same data on an interactive GUI. MgrsApp can also send commands to Grail in the case of a Parameter. Since some instruments take commands, they need to be able to be controlled by pyCLEO, MgrsApp allows us to do this.

Reference Materials

Acronym List

GBO: Green Bank Observatory

GBT: Green Bank Telescope

SDDev: Software Development Devision

M&C: Monitor and Control

CLEO: Control Library for Engineers and Operators

pyCLEO: Python Control Library for Engineers and Operators

MVC: Model View Controller

GUI: Graphical User Interface

Glossary

Manager: A piece of software that provides an interface which you can use to monitor and control a piece hardware. For example, every receiver on the GBT has a manager associated with it. Every manager has its own set of individual parameters and samplers.

Parameters: Interfaces the user can interact with and tell the instrument to do something. This of parameters as physical switches.

Samplers: Interfaces that simply provide information or data about the instrument. Think of samplers as dials or gauges.

Widgets: Tools from Qt which we use to display information on a GUI. pyCLEO uses: CleoLineEditFeedBack, CleoLineEditSampler, CleoComboBox, CleoCheckBox, CleoPushButtonCheck, CleoPushButtonSampler, CleoSpinBoxDouble. We are gradually introducing other widgets and figuring out the best ways to use each widget depending on the kind of information we are displaying.

StyleSheet: The way Qt allows users to customize widget displays. We can change color, of both widgets and text, font, font size, outline color, and much more.

Qt Designer: Qt tool used to design Qt widgets

Qt Creator: Qt IDE used to design Qt widgets with more customizability

MatPlotLib: A plotting library for the Python coding Language

Links

*Referenced in paper

*Sending commands with Grail:

http://docs.greenbankobservatory.org/pyCLEO/howto.html#sending-commands-with-grail

*Qt documentation for Signals and Slots:

https://doc.qt.io/qt-5/signalsandslots.html

*pyCLEO documentation:

http://docs.greenbankobservatory.org/pyCLEO/index.html

CLEO user's manual:

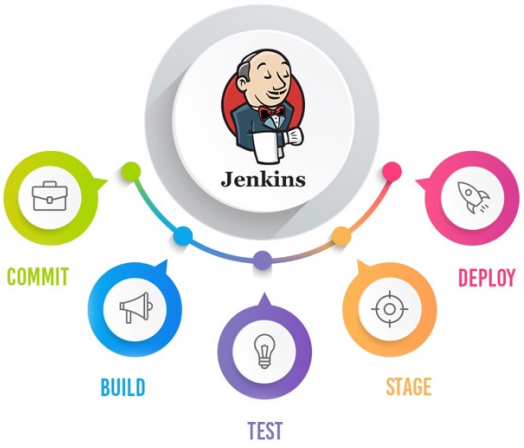
https://www.gb.nrao.edu/~rmaddale/GBT/CLEOManual/index.html

The user interfaces for the NRAO-Green Bank Telescope by Ron Maddalena:

http://greenbankobservatory.org/~rmaddale/GBT/Spie2002.pdf

Appendix – Design History

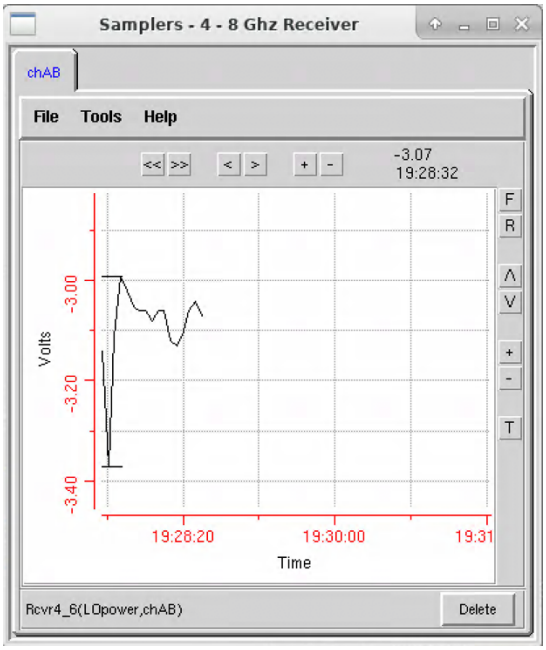
General Application Development



Despite the idea of pyCLEO having been first conceived a few years ago, it has only just gained traction at the beginning of this summer. Without pressure to complete the project and no official deadline in place, pyCLEO production is not a priority, but has begun steady progress and gained a loose workflow.

The SDDev is using a variety of tools and techniques to complete pyCLEO. Jira is used for project management and tracking progress. We are using Git for code creation and version management. The specific members who are mainly responsible for code production meet weekly to discuss progress and any arising issues. Those same members also meet bi-weekly with product owner, Pete Chestnut, with other regular key users to get vital feedback and demo in-progress working applications. Jenkins is used to stage, test, and deploy code. While we have not done this yet for pyCLEO, we intend to start rolling out code in the near future to cast a wider net in terms of feedback. The final step that is not completed as regularly as the previously mentioned steps are is to write documentation. The link to current pyCLEO documentation can be found in the Reference Material Section but can only be accessed on the GBO network.

Use of the PyQtGraph feature



Double clicking on a parameter or sampler on any receiver window will toggle a simple graph feature. The graph will automatically update and show live GBT values, the popup also offers many options like Save, Print, Max/Min, and exact values when you hover over the line.

To implement this feature in pyCLEO we started by investigating the PyQtGraph library. PyQtGraph is a scientific graphics and GUI library.

The first step in this process was to implement the pop-up window. Double clicking a CleoLineEdit widget was not built into the CleoWidgests wrapper class, additionally, based on what we have read on PyQt documentation, there is no straightforward double right click function for line edit widgets. As a work-around we placed a transparent tool button over the sampler and ensured it was on top of the line edit widget in layers, so when a user presses the sampler, they are actually pressing a tool button and then toggling the pop-up window. This could be easily implemented as a wrapper class in the CleoWidgets but currently, it is not implemented in any higher level than in the receiver window in which we were experimenting.

Upon reading more documentation on the PyQtGraph library we found it to be a fairly limited collection. PyQtGraph does offer the ability to download all data as a .csv file which is convenient. Essentially, creating a very simple graph and being able to download that file would be possible and relatively easy to implement. However, any new features that we might want to add would be difficult considering the limitations of PyQtGraph. We could implement this feature using PyQtGraph, or we could switch to something like MatPlotLib and do the same thing, and possibly get more features. Additionally, more people know how to use MatPlotLib, and therefore it would be more time efficient to use a library that developers are familiar with.

The Current State of the pyCLEO Project

The current state of the pyCLEO project is still very much in the early stages. With the current MVC architecture we have, collectively, created approximately 7 different ‘working’ pyCLEO applications. Working, meaning that the application is getting live values from the telescope and also automatically updating those values. These test applications have been primarily receiver applications since that is what MgrsApp wraps. Upon branching out into other applications, like a rudimentary Device Explorer we have discovered that MgrsApp provides very limited functionality; it’s only set up to provide optimal use for receiver applications which make up only ~1/3 of applications. Additionally, MgrsApp is constrained in of itself because it only accepts scalar inputs. We plan to pause production to take a step back and come up with a more inclusive architecture before continuing to roll out applications.

Future Ideas

One of the best things about pyCLEO is that we know what it’s going to look like when it’s done. There won’t be lots of time allocated toward designing layouts or colors because these choices were already made when CLEO was created. This extra time provides the SDDev a unique opportunity to fix old problems, add new features, and make this the best possible programs for its users. The following are a list of ideas we have come up with; this list is in progress and we expect to come up with even more ideas as we continue the alpha testing process and get user feedback and suggestions.

The logging and message systems are essential parts of working with the GBT. However, this vital information is in a separate application, and when working with many CLEO windows, this can become cumbersome. We would like to integrate both logging and messages into each application window, possibly in a separate tab, so you can always see updates for the manager you’re working on.

A major priority is to implement metrics as soon as possible. One major problem we have encountered during the pyCLEO creation process is that there is no metric on CLEO. Meaning, we have no quantifiable data on what applications and features are most clicked on, making it difficult to know what our priorities in pyCLEO should be. While implementing these metrics might not be super helpful immediately, in the long run, as more applications roll out, or new features are added, this information will be invaluable.

Managing CLEO windows is a problem that was solved by the CLEO container application, which allows the user to open multiple applications in one window, and each application shows up on its own individual tab. As we create pyCLEO we are experimenting with different ways to deal with this issue. We are looking into different docking techniques which can be achieved thorough Qt Creator. In effect, the user could pick which Samplers and Parameters they want shown, organize them how they want in a window, and then save that layout for later use. This has also led us to inspect resizing windows and fonts to satisfy personal preferences.

Our final major idea is to create entirely new applications. Since we are still in the early stages of creating original CLEO windows, there have been no moves to create new applications. However, we know there have been requests for specific applications, in addition there are new instruments being added to the telescope that need their own applications. So, while we have no set list of new programs, we do intend to create multiple new applications, further expanding the tools pyCLEO provides.

Thank You and Credits

Thank you to Jim Braatz, the NRAO Summer Student Program Coordinator, and to Will Armentrout and Ryan Lynch, the Green Bank site coordinators for making this REU program possible.

And a huge thank you to the Software Development Division at Green Bank: Nathaniel D. Sizemore, Paul Marganian, Thomas Chamberlin, Matt Harrison, Kasey Purcell, Laura Jensen, Ramon Creager, and Joe Brandt.