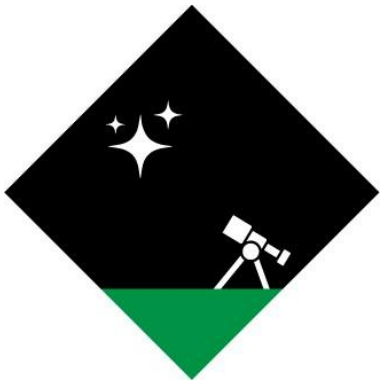


Operation Dark Sky

Technological Feasibility Study

March 4th, 2022

Team: Luke Thompson, Jordan Tatum, and Justin Ceccarelli

Sponsored by: Jim Clark, Peter Kurtz, and Henrique Schmitt

Mentored by: Anirban Chetia and Tomos Prys-Jones

1 Introduction 2
2 Technological Challenges 3
2.1 - Programming Language 3
2.2 - Graphical User Interface 3
2.3 - Data processing and Graphing 4
3 Technology Analysis 4
3.1 - Programming Language 4
3.1.1 - Desired Characteristics: 4
3.1.2 - Java 5
3.1.3 - C 5
3.1.4 - Python 6
3.1.5 - Final Decision: 7
3.1.6 - Proving Feasibility: 7
3.2 - Graphical User Interface 8
3.2.1 - Desired Characteristics: 8
3.2.2 - PyQT 9
3.2.3 - Tkinter 10
3.2.4 - Kivy 12
3.2.5 - wxPython 13
3.2.6 - Final Decision 15
3.2.7 - Proving feasibility 15
3.3 - Graphing Tools 16
3.3.1 - Desired Characteristics: 16
3.3.2 - Matplotlib 17
3.3.3 - Plotly 17
3.3.4 - Seaborn 18
3.3.5 - Final Decision: 19
3.3.6 - Proving Feasibility 19
3.4 - Communicating with the Backend 19
3.4.1 - Desired Characteristics: 19
3.4.2 - Analysis: 20
3.4.3 - Final Decision: 20
3.4.4 - Proving Feasibility: 20
4 Technology Integration: 20
4.1 - GUI code Architecture 20
5 Conclusion 22

1 Introduction

The Navy Precision Optical Interferometer (NPOI) is an astronomical observatory operated by the U.S. Naval Research Laboratory (NRL). While the NPOI does conduct research on unique astronomical phenomena like binary stars, it also carries out a range of practical functions that support the U.S. Navy's navigational and communication capabilities. Even in the modern world of satellites and GPS, the Navy still relies on the stars as a form of navigation that is entirely independent of technology or political influence. The NPOI is the largest baseline interferometer in the world, and can conduct research that would be impossible at any other observatory. The following paragraphs will cover the general functioning of the NPOI, the operational challenge that it is currently facing, and Team Dark Sky's proposed solution to the problem.

The NPOI is primarily dedicated to astronomical research with the aim of improving our understanding of the stars and planets in our galaxy. Its status as the world's largest baseline interferometer makes it especially well suited to studying binary stars, or pairs of stars that orbit each other. Like most observatories the NPOI can only operate at night, during fair weather, and beneath a cloudless sky. Since these circumstances are out of the observatory's control, they have to make the most of every opportunity to collect data. This means planning each night of observation well in advance to make sure that the equipment is positioned and then repeatedly re-positioned to capture the appropriate stars at precise times over the course of an evening. Properly planning an event of this scale poses a significant challenge, especially considering the constant movement of the stars in the sky relative to the earth. The astronomy team at the NPOI would need to perform an exhausting amount of technical work to determine the optimal timing and ordering of their observations, and then would need to carefully coordinate the observatory's equipment during the night to match those plans. This takes an experienced professional an exceptional amount of time to do just once, but must be carried out every night the NPOI observes the stars.

In order to handle this organizational challenge, the NPOI created a software application they called "obsprep", which is short for observation preparation. This software allows an astronomer to create a list of the stars they wish to observe before calculating the optimal time and order to observe those stars to maximize the observatory's uptime. In addition, obsprep can communicate with the interferometer's hardware, allowing it to quickly transition from observing one star to the next by shifting its mirrors.

When obsprep was originally written in the late 90s, it met the NPOI's needs for planning and organizing the observatory's nightly work. Unfortunately, in the decades since then the software's front-end has become outdated to the point where it requires frequent maintenance simply to avoid crashes and unplanned downtime. The GUI is also an outmoded relic that requires a great deal of experience and training both to operate and to even install the software on a new machine. The NPOI has decided to scrap the original implementation, and has hired team Dark Sky to create a new front-end application for the obsprep software. The NPOI envisions a solution that can be easily installed on a user's work machine regardless of operating system, that allows the user to select multiple desired stars for a specific night, and

allows the user to generate and save graphs of the stars' progress through the sky. There are also several stretch goals that the design team will consider, including a formatted text output and an installer script that can perform the installation automatically. The software's backend will remain intact, which includes the functionality that calculates star positions and communicates with the hardware that aims the telescope. The user-interface will be completely rebuilt with a modern design to be fast, functional, and intuitive.

Now that we have established a broad outline of what we hope to accomplish, we are ready to consider the specific features and technical requirements that this project will need to include. In the remainder of this document we will first outline the Technological Challenges that must be overcome before assessing possible technology-based solutions in the Technology Analysis. Finally we will outline our proposed architecture for implementing these technologies into our product.

2 Technological Challenges

This section will investigate the technological challenges facing this project, and the design decisions that need to be made in order to create a functional end product. These decisions are listed below, and the different options will be described and compared in the Technology Analysis sections.

2.1 - Programming Language

The project will be written in the programming language that offers the most appropriate tools and libraries for our particular needs. We will not be choosing a specific operating system to run the product, as the client has specified that our solution should be cross-platform. Given the requirements of our client and the experience of our team, we are only considering three languages: Python, Java, and C. We will consider these options based on their cross-platform support, ease of maintenance, and available libraries and frameworks.

2.2 - Graphical User Interface

The product will need a graphical user interface that can collect user input and display appropriate outputs. This interface should be capable of handling a wide variety of complex user inputs, yet still be simple and intuitive enough to be used without extensive experience or training. We will investigate publicly available software packages, and choose the most appropriate one based on functionality, documentation, and sustainability.

2.3 - Data processing and Graphing

The product will need to generate and display data in several different styles. We will need to implement software into the product that can transform tabular data into charts and graphs with specific formats. We will choose and implement a graphing tool based on standardization, longevity, and variety of features.

Now that we know what specific features the project must include, we can consider what options are available. The following section will address the challenges mentioned above, and evaluate a variety of technologies that can potentially be implemented to overcome these challenges.

3 Technology Analysis

In this section we will go over the details of the technological challenges listed above, and consider what specific technologies we can use to solve the problem. We will address each challenge one at a time, starting with the programming language, followed by the graphical user interface and then the graphing utility. After introducing the decision that needs to be made, we will state our desired characteristics for the technology before describing the different options. Once all options have been covered, we will summarize our findings in a table and choose the technology that best meets our needs.

3.1 - Programming Language

The programming language plays a major role in determining the scope of our project and what resources are available. Each language offers a unique range of built-in tools that are key to their operation and define what is simple or difficult to implement. We also have to consider the existing infrastructure at the NPOI, as our application will need to communicate with their systems and will be maintained by NPOI personnel. The personal preferences of our team is also a factor, although all of us are willing to work outside of our comfort zones to ensure an optimal product.

3.1.1 - Desired Characteristics:

For this project, the most crucial elements we need from the programming language are cross platform support, access to frameworks and libraries, and ease of maintenance. Cross-platform functionality is a core requirement for our product, and as such any language that is limited to just one operating system is automatically disqualified. A language that is simple to implement across multiple platforms will make our work much easier. Another essential characteristic of any language is what prebuilt tools and libraries are available. We are looking for a language with reliable and easy-to-use graphical interface and 2D graphing tools. Finally, ease of maintenance will be paramount to make sure that our solution will continue to benefit our client for years to come. There are several popular general purpose programming languages that perform well in these metrics.

Alternatives:

With these factors in mind we have chosen to consider the Java, Python and C programming languages. In the following subsections we will break down the advantages and disadvantages of each language in terms of the characteristics described above

.

3.1.2 - Java

Cross platform support:

Java is an innately cross platform language. Java can be seen both as a compiled language and an interpreted language, meaning that Java programs are compiled and then run on the Java virtual machine(JVM), not directly on the host machine. This means that a machine of any platform can run a java program provided it has a JVM. All major platforms can create a java virtual machine, but not every personal computer will have a JVM installed on it. Requiring access to the JVM would make installation slightly more complicated for users that do not already have a JVM on their computer.

Frameworks and libraries:

The most prominent GUI creation tool in Java is Swing. Swing is a longstanding tool for GUI development in Java that may be approaching the end of its lifespan. Oracle intends to phase out Swing in favor of the more modern JavaFX. This decision has not been well received, as JavaFX is widely regarded as buggy and less reliable than Swing. Oracle began dropping support for Swing in favor of JavaFX, but ultimately reversed the decision in 2018 after widespread backlash. While Java does have the capacity for GUI creation, and even has access to a number of IDEs that rapidly accelerate GUI design, Java does not have a prominent GUI creation library that has been consistently supported for more than a few years.

Ease of maintenance:

Java is an object oriented language, and liberally applies the principles of object oriented programming across all of its functions. Peter Kurtz, our technical point of contact with our client, specifically mentioned that he dislikes object oriented programming and requested we avoid it within this project.

3.1.3 - C

Cross Platform Support:

C has the least cross platform support of our three candidates. C itself is a cross platform language in that C code can be converted into a usable program for any platform. Unlike Java and Python, which are at least on some level interpreted languages, C is almost exclusively a compiled language. The C code can be distributed to any platform, but to be executed it must be compiled into bytecode and then assembly. As such, cross platform development in C would require either the end user to compile the code for our GUI, or an additional script capable of performing this function on any platform.

Framework/Library Access:

C is a relatively low level language compared to Java and Python. This requires programmers to assert more direct control over their designs. C is a powerful language that has been used to create a remarkable variety of applications, including many with graphical user

interfaces. Popular graphical libraries include SDL and GTK. C is usually chosen for its precise resource control and efficiency, with the caveat that C provides very little convenience or support to the programmer compared to an interpreted language. An interpreted language and its associated development environments can alert the programmer to problems before they occur, dynamically adjust structures like arrays, and automatically free unused memory to prevent memory leak errors, and more. Higher level languages offer a fantastic amount of streamlining compared to a language like C, which while efficient is much more demanding to the programmer on how its tools can be used. For example, simply importing libraries can be a time-consuming process in C. If we design multiple C files that make use of the same graphical library, which we are likely to do, we must also now determine the correct order of declaration not only for our imported libraries but also our own files. This action is much simpler in Python, which only requires an import command in the file that needs the library, no header file required.

Ease of Maintenance:

Maintaining a C program would require greater documentation from our team and greater effort from our clients compared to Python. C uses simplistic syntax and functions that typically require more experience and understanding to implement. The code is less self-explanatory than other languages and as such requires more consideration and more effort to produce a program that can be easily understood when read. This will require a greater time investment from our team while writing the code, as well as more effort spent documenting and commenting within the software to ensure proper maintenance.

3.1.4 - Python

Cross Platform support:

Python is an innately cross platform language. Python is compiled into bytecode that once distributed can be run on any computer that has an up-to-date python interpreter. This is superior to C which would require compilation on the end user's computer and roughly on par with Java's cross platform support, with only a slight edge going to Java as the JRE is estimated to be more widely distributed to end users than the Python run time environment.

Framework/Library Access:

Python has a broad assortment of libraries and frameworks available. PyQt and TKinter are prominent examples that are adapted and modernized from the C tools QT and GTK. These libraries have a long standing history of development on all three major PC platforms and are relatively easy to pick up and learn. The options covered here will be further explored in section 3.2, Graphical User Interface.

Ease of Maintenance:

Python will be inherently easier for our client to maintain going forward since their current GUI already operates in Python. Additionally Python's syntax is very modernized, and is closer to human language than most programming languages. This makes analyzing the code faster and easier for programmers of all skill levels. The convenience of features like garbage

collection will reduce the number of functions that need to be implemented, cutting down on the overall amount of code that there is to read.

Metrics:

Each language has been ranked in each desired characteristic. The average of these rankings is then taken and the language with the lowest average is the preferred language for our program.

Table 3.1 - Rating for Programming Language Options

	Cross platform support	Frameworks / libraries	Ease of maintenance	Average
Java	1	2	3	2
C	3	3	2	2.6
Python	2	1	1	1.3

3.1.5 - Final Decision:

Python performs well in all of our desired characteristics. It has strong cross platform support and a variety of robust GUI creation tools. Python is easier to read and understand than the alternatives, and will pose less of a challenge to maintain. It also does not force object-oriented programming the way Java does. The GUI must be able to send and receive information from the Obsprep backend written in C, a task which Python, as a glue language, was specifically designed for. Moving forward we will be limiting our analysis to tools available in python.

3.1.6 - Proving Feasibility:

Moving forward we will begin development of our prototype in Python. We are confident this is feasible as the current edition of the Obsprep GUI is in Python. We plan to develop a demo of our product this semester that will allow us and our client to begin refining our expectations and preferred design for the GUI. The prototype will allow a user to click through the GUI and explore what it would be like to work in our GUI with a premade data set.

3.2 - Graphical User Interface

The quality of a Graphical User Interface (GUI) is absolutely critical to the satisfaction of the user, as it serves as the face of the program. For our project it is important that our GUI be user friendly, intuitive, efficient and professional. The user will rely on our program to share and record their scientific findings. Our GUI will be expected to allow users to input data and select options in a simple, intuitive manner, and then display output data accurately and legibly.

3.2.1 - Desired Characteristics:

We have decided on four characteristics that we will use to evaluate the different GUI library options. These characteristics are functionality, documentation, design quality and sustainability. Functionality is an important characteristic because we will need the GUI to perform very specific functions. If those functions are not built into the library, then we will need to spend time developing them ourselves. Proper documentation is essential to make full use of all the functions of a library, and will make it easier to implement specific functions into our GUI. Design quality refers to the degree of polish within the library, such as the lack of bugs, consistent language, and reliable outputs. Finally, sustainability reflects the degree of support the library can expect to receive in the future. If a library does not receive support, it will gradually deteriorate, affecting any product built using that library. Each option will receive a score out of ten in each category, where a one represents a complete lack of the characteristic, a five represents a reasonable degree of competence, and a ten represents an optimal implementation of the characteristic. The total scores for each framework will be out of forty, and the highest scored framework will be used for the final product.

Alternatives:

With Python there are a great deal of possible frameworks to choose from. Our team used programming websites like stack overflow, geeks for geeks, and w3 schools to identify the most popular and well-regarded GUI frameworks. These were PyQT, Tkinter, Kivy and wxPython. We will be doing a brief overview of each framework and why it is being considered.

PyQt is a GUI framework for Python that provides a library of tools to help develop and design a user interface. Some of these tools include location and positioning services, multimedia, NFC and Bluetooth connectivity, a Chromium based web browser, and traditional UI development. Many modern interfaces are built with PyQt, and it is a strong contender for our framework choice. PyQt supports cross platform functionality and has an easy to develop installer script. The framework is very simple to install, requiring only a one line terminal command.

Tkinter is the next most popular option. Tkinter is an open-source framework that is well known for its simplicity. This makes it easy for new developers to learn and use. This particular framework is a strong candidate because it comes built into all versions of Python. Being built in will decrease installation time and make it so new users for the program no longer have to install additional packages. Tkinter allows for cross platform functionality, and makes creating an installer script a simple affair. Tkinter also offers flexibility and stability which will help us when implementing specific tasks into our project.

Kivy is an open source multi-platform GUI that can run on all operating systems. It allows developers to create innovative interfaces with a multi touch user interface. The main purpose with Kivy is for developers to create an application once and use it on all devices. This framework allows developers to quickly create interactive designs and rapid prototypes. Kivy is mostly aimed towards web application interfaces but can be used for building any interface.

wxPython is the smallest and least popular framework we are considering. Despite its smaller community wxPython comes packed with quality tools for development. One benefit of wxPython is that it features a design environment that lets you easily preview an interface while you design it. It also has a large library of widgets that add a lot of design options for an interface. wxPython offers a lot of flexibility with objects and connections within the interface. The framework can deploy its applications for all operating systems. It unfortunately does not have an easy way of creating an installer script for the application.

In the next section we will be going into more detail about the frameworks discussed above. We will be considering the pros and cons of each framework in terms of the four desired characteristics: functionality, documentation, design quality, and sustainability.

3.2.2 - PyQT

Functionality:

One of the benefits PyQt offers is its coding versatility. PyQt's structure is designed around signals and connections that create contact between objects. It also offers more than just a framework. Qt uses a broad variety of platform API's for Networking, database development and more. It offers primary access to those tools through a special API. PyQt has a very steep learning curve which means more time learning and less time developing. This framework is a commercial product, and as such we will need to acquire a license to use it.

Rating:

Given the advantages and disadvantages of PyQt we have decided to give it a rating of 8. This is because it offers very strong Functionality through its coding versatility and local API access to useful tools. It does not get rated higher though because it's a challenging framework to learn and we would have to pay for a commercial license to use it in our application.

Documentation:

PyQt's documentation offers a lot of information about how to set up and utilize their framework. Their documentation is broken up into easy to comprehend sections based on each part of the framework. Although their documentation lacks Python specific documentation. The documents leave a lot of the functionality of the framework in question. There is also a lack of examples of the functions being used, it just describes the function itself.

Rating:

The final rating we have given the documentation on PyQt is a 6. The documentation describes a lot of the tools and functions available in the framework, but it also leaves a lot unanswered.

Design Quality:

PyQt offers a variety of user interface components. Some of these components include buttons or menus, all designed with a basic interface for all. It also includes advanced widgets

that reactively scale. Qt offers a design interface to visually design your interface before coding. The only downside is the framework lacks a widget variety.

Rating:

The design quality of PyQt receives a 9. This is because it offers a variety of user interface components, has advanced widgets and a design interface that can help us visually structure our GUI. PyQt offers a nearly perfect design quality but lacks a variety of widgets.

Sustainability:

PyQt is one of the most used frameworks for developing graphical user interfaces. It has a large and active community that helps maintain and upkeep the framework. Currently it is unavailable on Python 3.5 and newer. Its last update to PyQt was in 2016

Rating:

The current sustainability is currently looking bright but is not guaranteed. As of right now it is working and functional. Qt's community is big and help keeps to maintain the framework and add to it but the developing team has not updated it since 2016. The sustainability is not guaranteed leaving the future of it in question. For these reasons we are giving it a 6.

3.2.3 - Tkinter

Functionality:

Tkinter offers more flexibility and stability compared to other frameworks. Meaning that it will respond better to different user and system requirements. It is easier and faster to implement than the other frameworks because it is built into Python. Being built into Python means it works very well with other external libraries. The learning curve is low and the framework is considered easy to master. Tkinter also offers simple syntax for developers to read and understand the code. On the other hand, debugging can be difficult with Tkinter. It is also not purely Pythonic, which can make it difficult to grasp.

Rating:

Looking through all of the advantages and disadvantages of the functionality of Tkinter we give it a rating of 8. This is because iit offers great flexibility, an easy learning curve, compatibility to external libraries and its fast implementation. We are still missing out on some key points though. The challenge of debugging may prove to be a problem during development and it not being purely Pythonic makes it more difficult to integrate.

Documentation:

Tkinter has deep documentation about every function included in the framework. The documentation is hosted on Python's own website making it very reliable and frequently updated. They also have their own documents on their website with a bit more information available. Tkinter is one of the most widely used GUI frameworks with a massive community

helping and contributing to problem solving. It also offers video tutorials on how to implement and use the framework. There are some older, outdated items in the documentation that refer to previous versions, but they are usually easy to identify.

Rating:

With all the research looking into Tkinter's documentation we have decided to give it a perfect 10. There is so much documentation between Python's website and Tkinter's own website. As well as hundreds of tutorials online on how to implement and build a GUI with it. There is very little to complain about when it comes to the framework.

Design Quality:

Tkinter offers three geometry managers to help design the layout of the application: place, pack and grid. It offers a layered approach which makes it easy to take advantage of the many features offered by the framework. Unfortunately Tkinter does not offer access to advanced widgets. It also lacks a design interface on par with the other options. It doesn't offer a native look or feel to the application

Rating:

With its lack of advanced widgets and non-native design interface will make it difficult to develop a complex layout. Our design will be simpler and more straightforward than it would be with the alternatives. For this reason we are giving Tkinter a rating of 5 for design quality. Its geometry manager design and layered approach does help balance out the negatives. Overall the design with Tkinter will not be as sharp or advanced as the other frameworks.

Sustainability:

With Tkinter being built into Python it will be around as long as Python is around. The last update to Tkinter was in November 2021, so it is still being actively updated and maintained. There are examples of many applications that were built with Tkinter and still use their interface. Some examples are restaurant management systems and traffic signal violation detection systems. With newer frameworks being developed and offering better design Tkinter could be left in the past and not be worked on anymore

Rating:

We can expect Tkinter to be carefully maintained for the foreseeable future. With it being built into Python it receives frequent updates, so we have decided to give it a 10 for sustainability.

3.2.4 - Kivy

Functionality:

Kivy offers smooth compatibility with various platforms including Windows, Android, Linux, iOS, MacOS, and Raspberry Pi. Its purpose is to write code once that can be used on all

platforms.Kivy performs better than HTML5 cross platform alternatives. This is an open source code framework that is free to use for development**.** Kivy is typically for web and mobile applications. It lacks flexibility when it comes to code structure and use. It also uses a kV language which is not suitable for it to compile the code alone. The package size for Kivy is unusually large. Kivy is known for having a steep learning curve and long development cycles.

Rating:

Looking at all the factors we have discussed for Kivy's functionality we have decided to give it a 6. Although it offers great cross-platform functionality it falls off in other areas. This is mostly because the tool is designed for web and mobile applications, and the steep learning curve. Realistically its functionality is not practical for our project.

Documentation:

Kivy has deep documentation of all its functions. The documentation breaks down into different parts, basic functions, programming guide and widgets. The documentation also shows a gallery of examples of how to implement different interface designs There is also a tutorial page on how to set up an environment and get started. Kivy lacks a dedicated community so there is not a lot of support for questions. It also only briefly touches on Kivy's specific language.

Rating:

Through all of the research done we have come to a conclusion that we give Kivy's documentation a 7. This is because it has thorough documentation on the functions and how to implement specific widgets and designs. We cannot give it a higher rating because there is a lack of community that has answered specific questions and there is not enough about Kivy's specific coding language.

Design Quality:

It has access to a plethora of advanced widgets and offers multi touch support for mobile applications. With Kivy you can include media such as videos, audio and gifs. It also creates a native interface for users on all systems. The framework does have some impractical use widgets such as sounds buttons.

Rating:

Kivy is a well designed framework that offers a broad range of useful interface tools. It is very robust and we are giving it a 9. Many of the functions are mobile-specific and not relevant to our project, which prevents this framework from receiving a 10.

Sustainability:

Kivy is a newer framework that is offered in newer versions of Python, unlike some of its competitors. It was last updated in 2021; we are unsure exactly which month. There are not a lot of major applications built with this framework, which is largely due to its youth. Since Kivy lacks a sizable community there is a real risk of it losing support.

Rating:

As it stands it is unclear whether or not it will be around in the future. There are not a lot of applications built using it, it does not have a large community, and it is designed for web and mobile applications. For these reasons we are giving it a 3 for sustainability. Overall the longevity of Kivy is unknown and it is not a safe choice when developing a long lasting application.

3.2.5 - wxPython

Functionality:

wxPython is very flexible when it comes to user interaction and system requirements. It offers a lot of functions within the framework making performing tasks easier. It does not handle large amounts of data easily and starts to lose performance when doing so. Due to wxPython's external library and rich widget selection it sacrifices runtime and is a slower framework compared to the ones previously discussed. It is currently under development, and contains more bugs than many of its competitors. It needs to be installed making it harder for the clients to use. Finally it has a steep learning curve which can slow down development time.

Rating:

wxPython is clearly still in development. It struggles to handle large amounts of data, runs slowly, and contains bugs due. On the other it has a rich selection of widgets and quality features. With this in mind, we give wxPython a 5 for functionality.

Documentation:

The documentation has different sections for different functions in the framework. It also offers an overview document to help walk you through documentation. The documentation is a single page and only tells you about the function and not how to implement it. There are no examples of setting up an environment with the framework. The community is small and there is not a lot of support online. There is also a lack of other websites such as, geeksforgeek and stackoverflow describing the framework or offering tutorials.

Rating:

The documentation for wxPython is very limited. For this reason we are giving it a 4. If there was not a documentation page describing functions it would be very hard to figure out how to use and implement it.

Design Quality:

wxPython offers a large variety of widgets with a lot of features built into them. Its applications also don't lose any quality across platforms. User interfaces are native looking and easy to use. It comes built-in with support for menu icons and key-board functions. Finally it offers a design

interface to help design and plan the interface before development. The only disadvantage is that it is tough to learn how to implement a widget.

Rating:

With wxPython their main focus is being able to create a native looking and user friendly interface. Due to it being a very design heavy framework we are going to give it a 10. With its large variety of widgets, no quality loss between platforms, built-in support for menu icons and design interface this framework offers the best design quality possible.

Sustainability:

wxPython was most recently updated in November 2021, demonstrating active support. It lacks a particularly large or vocal community. It is currently under active development.

Rating:

wxPython's short lifespan and absence of community calls the sustainability of the library into question. With no real security behind its longevity we have decided to give it a 4. It is currently under development which makes it a risky investment.

Table 3.2 - Rating for GUI Framework Options

	Functionality	Documentation	Design Quality	Sustainability	Total
PyQt	**8**	**6**	**9**	**6**	**29**
Tkinter	**8**	**10**	**5**	**10**	**33**
Kivy	**6**	**7**	**9**	**3**	**25**
wxPython	**5**	**4**	**10**	**4**	**23**

3.2.6 - Final Decision

Through our research we have come to the decision to use Tkinter as our GUI framework. With Tkinter deep documentation we will be able to develop a more in depth GUI for our sponsors. Tkinter offers cross platform functionality which is a necessity for our clients. It will allow us to have design freedom when developing our GUI. Our application will be able to be user friendly while keeping high functionality. With all these points Tkinter is the clear answer when choosing a framework for our GUI. The other options are strong as well but for our specific case and needs there's a lot of unnecessary functionality involved with them. We will be able to achieve cross platform functionality, an intuitive interface, an organized file directory, and even create an installer script for our application.

3.2.7 - Proving feasibility

Through these next few weeks we will be starting to develop with Tkinter and learn how to successfully develop an interface. Our plan is simple: we will start with a very basic interface. This will include adding some widget structure as well as accepting input from the user. The next step will be to develop a more in depth application that will take in input and give out some kind of output. After that we will create a basic back end where we have our basic interface communicate with our basic backend. Then we will include a more advanced interface within our project that can accept multiple inputs and more difficult tasks. Once we have the more detailed front end working with our basic backend we will then connect it to the actual backend. Finally for our prototype we will have a user interface that will retrieve data from the backend and display it to the user. Below we have a few pictures displaying a more in depth interface (Image 3.2.2) with Tkinter and our basic one we have developed (Image 3.2.1)

Image 3.2.1 - Basic User Interface

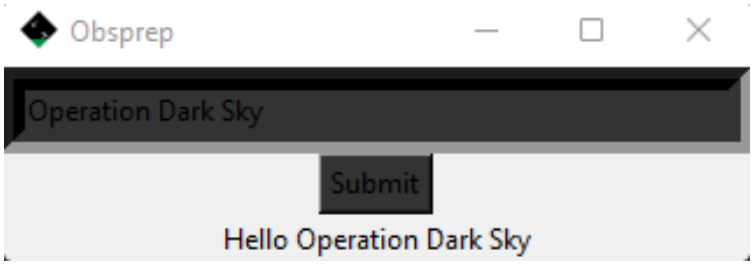
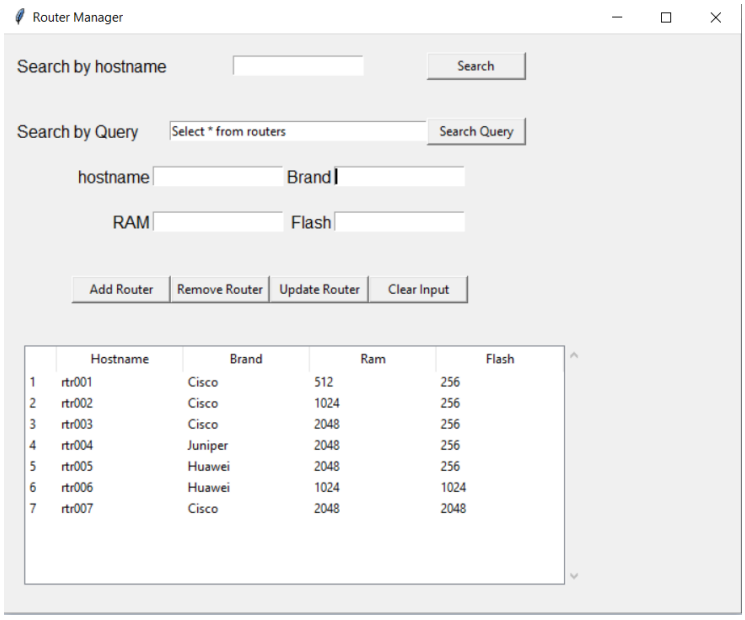


Image 3.2.2 - Advanced User Interface

I



In this next section we will be discussing potential libraries for displaying graphs for the stars data. This is an important task for the observers to be able to track when the stars are observable. Operational downtime will be reduced and their research will be able to continue with ease. There are a handful of graphing libraries that Python offers. We will be going over all potential options.

3.3 - Graphing Tools

OBSPREP currently provides graphing functionality to the team at the NPOI. Our clients require this functionality to be maintained in our new frontend for them, and also expect us to make the creation, recording, and distribution of these graphs more intuitive than it currently is. The client expects simple 2D graphs with an X and Y axis that charts when stars are potential targets for observation. The Y axis is the desired stars, and the X axis is the time of night. The graph appears as a horizontal bar at the height of a particular star's ID when that star is visible. So if a star is a valid target from 2am to 6am, at that star's height on the graph there will be a line drawn from 2am to 6am. There are a number of alternative tools to use for this task. Some of the options we examined are matplotlib, seaborn, and plotly. These alternatives will be compared below.

3.3.1 - Desired Characteristics:

We require a graph creation tool that can generate 2D graphs based on user input, and that can have their information easily converted to text. We are looking for a data visualization tool that is simple to learn to use (not complex), easy to transcribe to text, and will likely be given continued support in the long term.

Alternatives

3.3.2 - Matplotlib

Complexity:

Matplotlib is the easiest to pick up and use of our alternatives. It has the fewest features, but also the least complexity. We do not have high demands for the kinds of graphs we need to create so the tool that can make simple graphs with the least expectation on behalf or the part of the programmer responsible for maintaining our program later the better. Matplotlib does not make complex graphical representations easy, but it does make simple graphical representations trivial to implement.

Transcription:

Matplotlib is also the easiest to transcribe to text. The simplicity of it’s implementation carries over to this criteria. The simpler it is to construct the graph the easier it will be to write a function that reads in a text input and constructs the commands to draw the graph.

Longevity:

Matplotlib has been the preeminent data visualization tool in python for almost 20 years. Matplotlib's longevity is reinforced by the fact that many of its competitors, such as seaborn, are based on matplotlib. Meaning that support for matplotlib cannot be dropped without also compromising many of the alternatives to matplotlib.

3.3.3 - Plotly

Complexity:

Plotly is the most complex of the examined alternatives. Plotly provides highly interactive graphs through the use of javascript and HTML. This allows for feature rich graphs to be made, but the ability to accurately understand what is happening when the code is executed requires the programmer to have an understanding of javascript and HTML as well as python. This also means that the graphs are regularly opened by the user's web browser, and while plotly does not need to access the internet, our client has specifically asked us to avoid entangling the internet with this program and the use of a web browser may be considered a compromise of that wish. Plotly is a product that aims to provide an incredibly expansive features list, as such it is also highly complex.

Transcription:

Plotly would be difficult to transcribe to and from a text format. We would likely make a large number of assumptions about the graphs structure in the method we use to reconstruct the graph from text which while remaining technically accurate may result in a graph that does not visually match what the original creator of the graph designed. It is possible to create a text file and interpret a text file completely according to the original graphs creator's design, but this would involve the creation of a much larger package to perform this function than with our other two candidates.

Longevity:

Plotly likely has the least longevity of our three options as well. Plotly is a highly modern and customer focussed product. Plotly aims to provide the most visually stunning and interactive graphs that they can, this is valuable for presentations and artistic purposes but less valuable for efficient data collection and storage. This also means that as trends in various industries outside of computer science change, plotly is subject to being replaced by a similar tool with something even as simple as a different aesthetic or brand.

3.3.4 - Seaborn

Complexity:

Seaborn is based on matplotlib and aims to improve on the formula. Seaborn was designed to make the inclusion of multiple variables on a graph easier. This does mean the complexity is higher but not much higher, and it is possible that the ability to streamline the addition of more variables may be of use to us. We know we will be tracking multiple variables, but it is unclear if the added complexity is worth the added features.

Transcription:

Transcription into seaborn will only be slightly more difficult than into matplotlib. The concern is that to resolve errors that occur in seaborn we may need to fall back onto matplotlib anyways. If we simply use matplotlib the entire time this situation is avoided.

Longevity:

Seaborn ranks highly on longevity, it has a successful history as a data visualization tool. However it is based on matplotlib which means that any support for seaborn inherits support for matplotlib.

Metrics:

The alternatives are ranked relative to each other. For example Matplotlib is the simplest alternative and therefore ranks best on complexity so it receives a 1. The average of these scores is then taken and the alternative with the highest ranking average (lowest numerical average) is the preferred choice.

Figure 3.3 - Rating the Graphing Tools

Valuable Trait:	Complexity	Transcription	Longevity	Average
Matplotlib	1	1	1	1
Seaborn	2	2	2	2
Plotly	3	3	3	3

3.3.5 - Final Decision:

Matplotlib is the utility we should use to create the graphs for our product. The unique challenges associated with our graphical implementations are mostly a matter of collecting and disseminating information in multiple formats that are not actual graphs. Turning them into text, into graphs, and back again. Matplotlib is the easiest to translate from a visualization, to text, and back again. Matplotlib is also the easiest to learn to use. Our client has not expressed interest in many of the more highlevel graphing utilities such as interactive graphs, although this is something we will be listening for in their coming feature requests. Sometimes, the simplest and easiest to use option will serve better than a feature heavy service.

3.3.6 - Proving Feasibility

The prototype of our GUI will include a selection of the kinds of graphs it can create. These graphs will be pre-generated from data of our choice and will allow our client to review how well these graphs suit their needs. Matplotlib is the current tool used by the Obsprep GUI and it seems to be sufficient for their needs.

3.4 - Communicating with the Backend

With the current Obsprep software's backend being developed in C we face the challenge of how to communicate with it. We need to be able to convert our Python requests to C types as well we need to be able to convert the C data types to Python types. This proves to be a challenge because Python uses type hinting syntax meaning you do not need to declare a data type before variables. Through this section we go over the details and planning of how we are choosing how to communicate with the backend.

3.4.1 - Desired Characteristics:

While doing our research of connecting the frontend of an application with the backend we have come up with characteristics we will need. Some of the characteristics we will need are efficient communication. This is important because we do not want to lose any runtime speed when retrieving data. Another important trait we need is it to work both ways going from Python to C and from C to Python. Our final trait we are looking for is to be able to utilize our programming languages memory management.

Alternatives:

Despite our research we have found very few options for communicating with a backend in C. This includes looking at websites such as stackoverflow, geeksforgeeks and w3 schools. The only available option for us is a glue language called Cython. Cython allows us to easily change our Python frontend code to Cython and then from Cython to C. This also works in the reverse as well. It is the best solution as well as the only solution we have.

3.4.2 - Analysis:

Cython is a glue language that will make it so we can work with external C libraries. We will also be able to utilize the memory management of Python and C. With Cython we will be able to use Python's type-hinting syntax. Through the use of Cython we will be able to have the option for speed when needed. We will also have the option for safety when necessary. Unfortunately this will be a learning process for the whole team because none of us are familiar with Cython. It will add extra work because on top of creating Python files we will also need to create a Cython file to be able to communicate with the backend. We will also have to create extra code in our Python files to be able to convert it to Cython. Finally the future developers that will maintain the code will have to learn the syntax as well.

3.4.3 - Final Decision:

With the problem at hand of having to communicate with the current backend we have decided to use Cython. It will make communicating our Python frontend with the C backend efficient. We will be able to keep Python's type-hinting syntax keeping our code simple and readable. As well with our choice of Cython we have the memory management from Python and C. The choice to go with a glue language is the best choice when considering all of the benefits

and negatives. This will make our code efficient, secure and able to easily communicate with the current obsprep backend.

3.4.4 - Proving Feasibility:

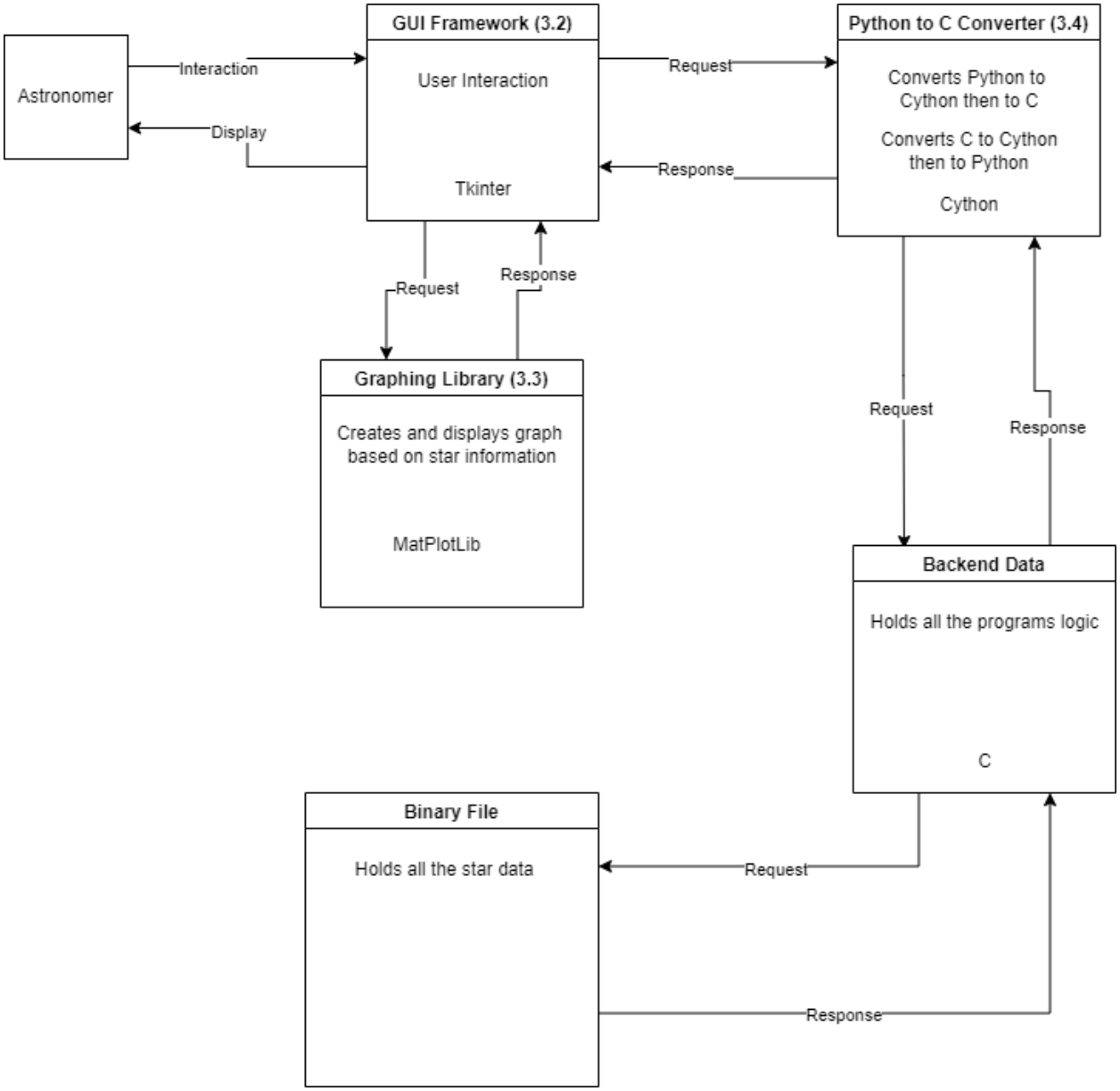
Our plan to implement Cython and prove its functionality will be simply hooking it up with our demo frontend. The first step we will take is to create a simple program that uses it to convert Python to C. Then we will make it a more complex program changin Python inputs to C data. Finally with the demo frontend it will be simple but we will have it utilize Cython to communicate with the backend. Our prototype will be able to grab some data from the backend with our Cython glue language. With these steps in place we will be able to prove how useful Cython is to our project.

4 Technology Integration:

4.1 - GUI code Architecture

In this section we will describe the structure of the code. Figure 4.1 below provides a visual map of the different elements of the software, and how they interact. The process starts with the Astronomer deploying and interacting with the user interface. The GUI then sends a request to our glue language Cython where it converts it to C. Then Cython communicates with our backend requesting data from it. The backend accesses the binary file where the data is stored. After the binary file gets the request it returns the information back to the backend code. Then the backend code returns a response to Cython where it is converted back to Python. Cython sends the Python response to the frontend code. Then if a graph is needed to display our frontend will call the MatPlotLib library with the retrieved data to create a graph. Finally the GUI will display the response to the Astronomer.

Figure 4.1 - Code Architecture



5 Conclusion

The goal of this project is to create a modern version of the NPOI's existing obsprep software with an intuitive user interface and well-written, easily maintainable code. The current form of obsprep has an unnecessarily complex interface, numerous bugs, and outdated references to features that no longer exist. Managing it costs our client valuable time that could be spent on more important tasks, but is currently unavoidable due to the essential functions the software performs.

In this Feasibility document team Dark Sky has attempted to break down the problem into specific challenges that can be overcome through the implementation of features and technology. Based on the analysis performed here, the team plans to:

- Write the software in in the Python programming language
- Create a Graphical user interface using the tkinter package
- Process and display data using the Matplotlib library
- Interact with the backend using the Cython library