|  |
| --- |
| CSM 500 - Project |
| Software Suite |
| The Creation of an Internal SaaS Project |

|  |
| --- |
| Student 220491901  October 1, 2024 |

Table of Contents

[Table of Figures 2](#_Toc178693099)

[Disclaimer 3](#_Toc178693100)

[Introduction 3](#_Toc178693101)

[Report Roadmap 3](#_Toc178693102)

[Presentation of the problem 4](#_Toc178693103)

[Calculating the Uncertainties 4](#_Toc178693104)

[Other Software Options 5](#_Toc178693105)

[Presentation of solutions to the problems 6](#_Toc178693106)

[Review of older programs 6](#_Toc178693107)

[Language choices 6](#_Toc178693108)

[The new JPL Software Suite 8](#_Toc178693109)

[Introduction to the Suite 8](#_Toc178693110)

[Improvements over the past software options 9](#_Toc178693111)

[Challenges during development 12](#_Toc178693112)

[Part-time development 12](#_Toc178693113)

[One man team 13](#_Toc178693114)

[Not having access to instruments 13](#_Toc178693115)

[Missed goals 14](#_Toc178693116)

[Looking to the future 16](#_Toc178693117)

[In Conclusion 16](#_Toc178693118)

[References 17](#_Toc178693119)

# Table of Figures

[Figure 1: MetCal FSC example 5](#_Toc178596118)

[Figure 2: UI Example 1 6](#_Toc178596119)

[Figure 3: UI Example 2 6](#_Toc178596120)

[Figure 4: Jinja Templating Example 7](#_Toc178596121)

[Figure 5: Loading errors 8](#_Toc178596122)

[Figure 6: Software Suite Splash Screen 9](#_Toc178596123)

[Figure 7: Instrument Search Overlay 9](#_Toc178596124)

[Figure 8: Flexibility of test parameters 9](#_Toc178596125)

[Figure 9: List of Recommended Standards 10](#_Toc178596126)

[Figure 10: Multimeter Class example 10](#_Toc178596127)

[Figure 11: Text-based connections 11](#_Toc178596128)

[Figure 12: Custom made connection image 11](#_Toc178596129)

[Figure 13: Index page for UUT 11](#_Toc178596130)

[Figure 14: Setup page for UUT 11](#_Toc178596131)

[Figure 15: Instrument connection page 11](#_Toc178596132)

[Figure 16: Tests page for UUT 11](#_Toc178596133)

[Figure 17: Project file structure 12](#_Toc178596134)

[Figure 18: B2900 Series calibration times 13](#_Toc178596135)

[Figure 19: Bootstrap Tour 14](#_Toc178596136)

[Figure 20: Bench section 15](#_Toc178596137)

[Figure 21: MetCal failure dialogue box 16](#_Toc178596138)

# Disclaimer

This report is written for the MSc Computer Science Project Module – CSM500. It is not intended to be released publicly unless Andrew Davis, student 220491901 grants consent.

# Introduction

In modern metrology, software solutions play a critical role in the calibration of instruments, ensuring measurement accuracy and reduction of complexity. Engineers and technicians often face the challenge of managing multiple software tools, many of which have different interfaces, functionalities, and levels of user-friendliness. This fragmentation results in inefficiencies, miscommunication between systems, and a higher likelihood of errors. To address this, the Jet Propulsion Laboratory (JPL) started developing a project to deliver a comprehensive software suite designed to unify and streamline calibration tasks. The JPL Software Suite (Suite) is not only aimed at reducing the confusion stemming from multiple software solutions but also at improving the precision, efficiency, and user experience of the calibration process. The goal is to create a single, intuitive platform that engineers can use for both current and future calibration tasks. This report details the conception, development, and implementation of this Software Suite. It also addresses the challenges encountered during the development and the plans for enhancements.

## Report Roadmap

The structure of this report is as follows:

1. **Presentation of the problem**: This section discusses the key challenges associated with calibration software, focusing on basic calculation of uncertainties in electrical, or direct, measurements and the limitations of other software solutions currently available in the market.
2. **Presentation of solutions to the problems**: This section discusses how the problems listed in the previous section are going to be solved by the Software Suite.
3. **The new JPL Software Suite**: This section introduces the Software Suite and the reasons behind the development that further emphasizes the solutions to the problems from the first section.
4. **Challenges during development**: This section discusses the development process of the Software Suite, focusing on the challenges that were faces while trying to reach the end goal.
5. **Missed goals**: This section discusses which features that were planned for development were not achieved and why they may or may not be developed in the future.
6. **Looking to the future**: This final section discusses more features that would be beneficial to include in future versions of the Software Suite to improve the user experience.

# Presentation of the problem

## Calculating the Uncertainties

Metrology, the science of measurement and its application [1], has many interesting aspects to the field but it is probably best known for the calibration of instrument which involves two steps: a verification and, if necessary, an adjustment. There are many factors that determine whether the calibration result will be a pass or fail, which may include whether the item is calibrated at room temperature, time since last auto-test or auto-calibration, length of the cables, accuracy of the display, number of measurements taken at each setpoint, or even if the battery is new. Those factors go into the Measurement Decision Risk, or the process of estimating the probability of making incorrect acceptance or rejection decisions based on measurement data. [2] These factors are also used in the calculation of the Measurement Uncertainty, or the non-negative parameter characterizing the dispersion of the quantity values being attributed to a measurand, based on the information used [3].

When it comes to calculating the uncertainty of a measurement, there are two types of measurements that are considered: Direct and Indirect [4]. Electrical instruments mostly deal with direct measurements, making the calculations relatively simpler since factors like computation error or room temperature not having any effect on the warping of the instrument.

Eliminating these unnecessary factors, a simplified calculation of the combined uncertainty leaves only four factors to consider: Unit Under Test (UUT) resolution, Standard (Std) – or the more precise instrument used to calibrate the UUT – resolution, Std accuracy or bias, and repeatability of the measurements. In the following equation, 𝑢 means Uncertainty and means Combined Uncertainty.

Std\_bias, Std\_resolution, and UUT\_resolution all come from the manufacturer’s specifications, for example ±10V, 0.0001V, and 0.01V respectively, but the repeatability is taken at the time of measurement which is calculated based on the number of readings (n). The next equation is for the standard deviation of the sample:

Where is the 𝑖𝑡ℎ reading and the mean value () of the sample is computed from:

Which we then use in the following calculation for the repeatability uncertainty:

Looking at the repeatability uncertainty equation, which trickles into combined uncertainty equation, having a high standard deviation will greatly affect the uncertainty of the measurements and thus creating a need to reduce any errors to keep that number small. But what steps could be taken to achieve a small repeatability uncertainty? As discussed below, there are a variety of additional options that can contribute to minimizing this uncertainty.

## Other Software Options

Typically, the calibration process would be done manually, especially in the beginning so that a procedure can be writing or, if one already exists, can be followed by someone unfamiliar with the instrument. But in these cases, operator bias can affect the measurements. A particular operator may incorrectly set up connections, misread the display, or input the wrong information. For this reason, creating automated procedures are preferred because there is consistency in gathering the data and a chance to provide clear(er) instructions on how connections should look.

Other software solutions are available to purchase for most instruments that already have automated procedures, but most of the time there are barriers to future development with said programs if it is allowed. Fluke’s MetCal Software [5] is written in its own language, using Function Select Codes (FSC), which requires a multi-week training course to fully understand the entire program [6].

A screenshot of a computer

Description automatically generated

Figure 1: MetCal FSC example

Northrup Grumman’s SureCal Calibration Software [7], on the other hand, requires a separate software to be able to write new procedures that uses an entirely different language, Visual Basic. These variations add to the cost of owning the software, as the purchaser would need to consider additional training for its team to learn, for example, an older programming language that many people may not know any more. A more common software used amongst engineers is called LabVIEW [8], which also requires a multitude of training due to the software being graphical in nature and not tied to a specific programming language. Besides LabVIEW, if the procedure is not in an executable form, the other software solutions do not have the flexibility to adjust testing parameters or repeat a specific test without restarting the entire procedure, which can occur in cases like loss of communication to the instruments.

# Presentation of solutions to the problems

One of the issues with having the different software solutions is that each software solution requires its own resources to download and install. This also leads to having to search through or remember where these files reside on the local machine and what their capabilities are or which instruments they are able to calibrate. There is no workaround for the bought software programs, so the focus is on the in-house built software and simplifying them down into a single software package that is more manageable, cost-effective, and lighter on the computer resources.

## Review of older programs

Engineers are known to not be very User Interface (UI) focused, getting things working as quickly as possible, which can create varying looks for each software program created. Not only will the UI be different, but if there is no established handbook or procedure on how to create the software, then each one could be made in different programming languages, causing issues with future development or maintenance.

|  |  |
| --- | --- |
| A screenshot of a device  Description automatically generated  Figure 2: UI Example 1 | A screenshot of a computer  Description automatically generated  Figure 3: UI Example 2 |

## Language choices

From the beginning, the main goal for the Suite was not just about combining other software into one location, but to anticipate new technicians needing to continue development of a new instrument procedures and maintain the software due to bugs or security issues. Thus, finding the right language(s) to use that are easy to learn and have longevity is imperative for developing in-house software that is both sustainable dynamic enough to meet unique calibration requirements. This decision came down to utilizing Python, alongside a more dynamic frontend with HTML, appeared to have the strongest potential for the solution to fixing the problems listed in the previous section.

With Python becoming increasingly popular among programming languages [9], it appeared as a strong candidate for transitioning previous software programs written in other languages and to be utilized for any future work, making it the new standard for writing programs. Being one of the first languages people are learning due to it being very beginner-friendly and having a big developer community building libraries, Python is quick to get started from a user-standpoint and has plenty of support and resources for communicating with the instruments. During the testing period, there is no need to constantly build out the software, as all that is needed is to run the Terminal or Python Idle to an interactive instance of Python. But only running scripts opposed to fully developed programs can be confusing to those who are unfamiliar with programming or are not computer savvy, which brings up the need to have a UI.

Python has some frameworks for web development, more specifically Django [10] and Flask [11], but there are features those frameworks consisted of that were deemed unnecessary for the Suite project such as an integrated database. Flask is already a lightweight version of Django, but the Software Suite needed to be even lighter. Luckily, someone else had the same idea and created Eel [12] which is a lighter version of Flask. The main feature that is consistent with all three frameworks is the use of the Jinja templating language [13]. Having the use of Jinja was pivotal since JPL has a Django template for creating branded websites [14].

A screen shot of a computer program

Description automatically generated

Figure 4: Jinja Templating Example

Creating reusable templates cuts down on the development time when adding new instrument to the Suite while maintaining a consistent look across the program. The only factor that would need to change with each page is the main content within that page. The use of reusable templates minimizes how much HTML is needed to learn, and it is minimized even further with JPL’s use of Bootstrap [15] by not needing to worry about specific pixel sizes for buttons, grids, pictures, etc.

The Suite does have some interaction with a database, but since it was previously established and not used exclusively for the software, SQL will also be used through a Python package that was developed by a previous engineer. The same engineer also created an API to read temperature and humidity data from the calibration lab to be included on the final calibration report, which is the final part the Suite needs to interact with. Python handles Excel interactions easier than the previously used programming languages since it does not substantially use up the Excel resource from the rest of the computer. In other programming languages, such as C#, the program takes control of the Excel resource on the local machine, not allowing any other instances of Excel to be open at the same time. If the program does not properly close the resource, then Excel is not properly unloaded for further interaction outside of the program, causing the user to force quit the app.

# The new JPL Software Suite

## Introduction to the Suite

As with most other software, there is a lot of behind-the-scenes action that may cause the program to take a while to load or otherwise make the program appear ineffective. With the previous software written be the other engineers, the software would load right away but ultimately delay most of the work, causing the software to glitch after loading, as seen in Figure 5.

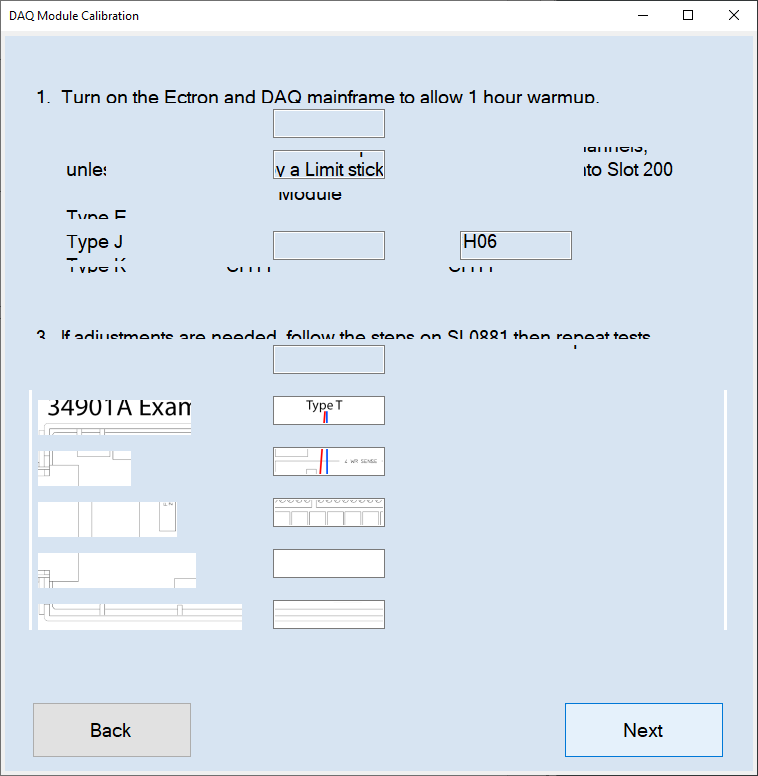


Figure 5: Loading errors

This was due to the program switching between the frontend and backend but only allowing one process to happen at a time, blocking the other process until complete. The issue could most likely be solved with the implementation of multi-threading [16], but that is a higher-level concept of computer science that beginners tend to have a hard time wrapping their head around. The way the Suite handles this issue is to move all the backend loading to the beginning of the execution of the software but masking it behind a splash screen, as one would see with other professional software. If there is a need to have a long wait for the backend again while operating the software, then a dropdown appears on the screen that tells the user there is a background activity happening. Having the frontend be based in HTML, along with the help of the Eel framework, threading is more inherent with the code and there is less of a learning curve to allow concurrency.

|  |  |
| --- | --- |
| A red and blue text on a white background  Description automatically generated  Figure 6: Software Suite Splash Screen | Figure 7: Instrument Search Overlay |

## Improvements over the past software options

The Suite has a major improvement over the previously mentioned software options in the way of being more flexible. Not just with the UI being web based, so it can be more dynamic and change depending on which instrument is connected to the computer, but also allowing the user to change test parameters if needed. Typically, a service manual or calibration procedure will dictate which parameters to test on the UUT, but if those tests fail and adjustments are not holding or are not possible, then a new limitation is placed on the instrument that could shrink the operating range or tell the end user of the instrument to not use a specific function or feature of the instrument. Without being able to change the parameters tested, the technician or engineer would not know how strict of a limitation is needed. On top of the parameters tested, the other programs may be hard coded to look for a specific channel or output if the instrument has more than one. But if that channel or output is broken or is known already to be limited, then changing to a different one is needed to test the instrument to its full capacity, meaning the software user needs to input which channel or output that would be.

A screenshot of a computer

Description automatically generated

Figure 8: Flexibility of test parameters

As mentioned previously, before the automated procedure could be written there has to be an established manual procedure along with its data report file which contains the uncertainty analysis. Part of the uncertainty analysis is calculating which standards are to be used to calibrate the UUT, although the manufacturer may have suggestions within the manual. Sometimes the manual will point to a specific instrument and other times it will list the requirements and give suggestions on which instrument to use, which may be the better approach since not all calibration labs will have the same instrumentation for their standards.

A screenshot of a test equipment

Description automatically generated

Figure 9: List of Recommended Standards

The Suite took a similar approach to the recommended standards in the sense that there are classes written within the backend for each available instrument that has the potential to be a standard but then one combined class that is based on the instrument’s type. For example from Figure 9, a digital voltmeter is required to test this UUT and the manual recommends a Keysight 34401A. The calibration lab instead may have a Keysight 3458A or a Fluke 8508A as the go to digital voltmeter for the better accuracy specifications, but the Suite has classes for all the instruments so that the software can follow both the manufacturer’s suggestion as well as the lab’s preference.

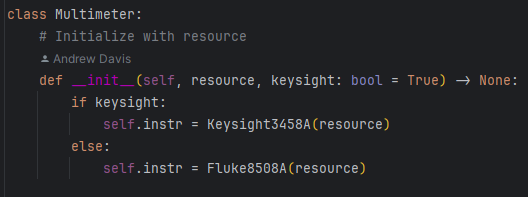


Figure 10: Multimeter Class example

One thing that manuals and other software might be unclear about is how to properly connect the standard(s) to the UUT for a given test. Some only have words describing how the connections should look, while others have pictures that are unclear due to multiple cable coloring being the same or the images being in black and white. The Suite intends to only include the clearest pictures available by the manufacturer or, if none are available, to have pictures created from scratch. Mentioning back to operator bias being an issue, having precise pictures will mitigate these issues so that there is no confusion on connections to potential cause failures during the tests.

|  |  |
| --- | --- |
| Figure 11: Text-based connections | Figure 12: Custom made connection image |

Operator bias does not come just from working with the instruments, it also comes from working with the software itself. Having integrations with an API and the Python library to communicate with the database, the Suite makes working with it very simple and cutting out the user input as much as possible. The only input needed into the Suite is the instrument identification on the setup page, but other than that the user just needs to click on a few buttons to navigate through the different pages and to run the tests. The Suite lets the backend handle more of the heavy lifting while the frontend is kept light and quick.

|  |  |
| --- | --- |
| Figure 13: Index page for UUT | Figure 14: Setup page for UUT |
| Figure 15: Instrument connection page | Figure 16: Tests page for UUT |

As previously mentioned, implementing the Jinja templating framework allows for faster reproducibility when new instruments are added to the Suite. The templates, and JavaScript files, are set up in a way that is easy to copy and paste into a new directory with minimal changes. This was done to emulate what Django does where sections are broken down into apps and can either be included or not. Breaking the files up into folders for each instrument also keeps everything organized and easy to locate, knowing what type of files and their names are needed for future instruments.

A screenshot of a computer

Description automatically generated

Figure 17: Project file structure

# Challenges during development

The Suite was an ambitious project from the get-go, especially since it has been a one-man development team, so there was bound to be issues along the way. On top of that, this project is seen as a luxury verses a need which meant development was not full time. There are still many features that have not been implemented yet or to completion.

## Part-time development

The metrology group does not have a dedicated software engineer whose sole purpose is to develop software, that is just not something that ever felt like a necessity by managers since the focus is on calibrations and clearing out the backlog. Therefor the employees that are hired are typically electrical or mechanical in nature, not software. When someone is not calibrating instruments, then the backlog climbs and money is being wasted. If someone were to develop software for the group, they would need the skills to do so but also do it when the backlog is low enough to give them time to do something else or spend their off hours working on development. It is much more economical to buy a $20,000 software license than to spend full-time employment hours on developing a software, at least in the manager’s eyes. But this goes back to the question of whether the instrument is supported by said software. Then it comes down to weighing the option of how many units are calibrated per year, how long do they take to calibrate, and will the customers be willing to pay for that calibration. For the customer, turn around time is important because they ask for calibrations to happen within a day, even if that is an unreasonable ask.

Let’s take the Keysight B2900 series for example. In the past, there was only a manual procedure for calibration, and it would take over eight hours per calibration. But after building a relationship with the manufacturer, the metrology group discovered these instruments could be done in an hour if they had the calibration software, which was exclusive to the manufacturer. With a little bit of convincing, the manufacturer allowed the metrology group to use their software and allowed them to cut their calibration and turn around time on the B2900 series ten-fold. From Figure 18, one can see from the ACTUAL\_TIME\_HRS and CAL\_PROCEDURE columns, prior to April 2020 the B2900 series calibration was done manually compared to after when the software was provided which cut down calibration hours significantly.

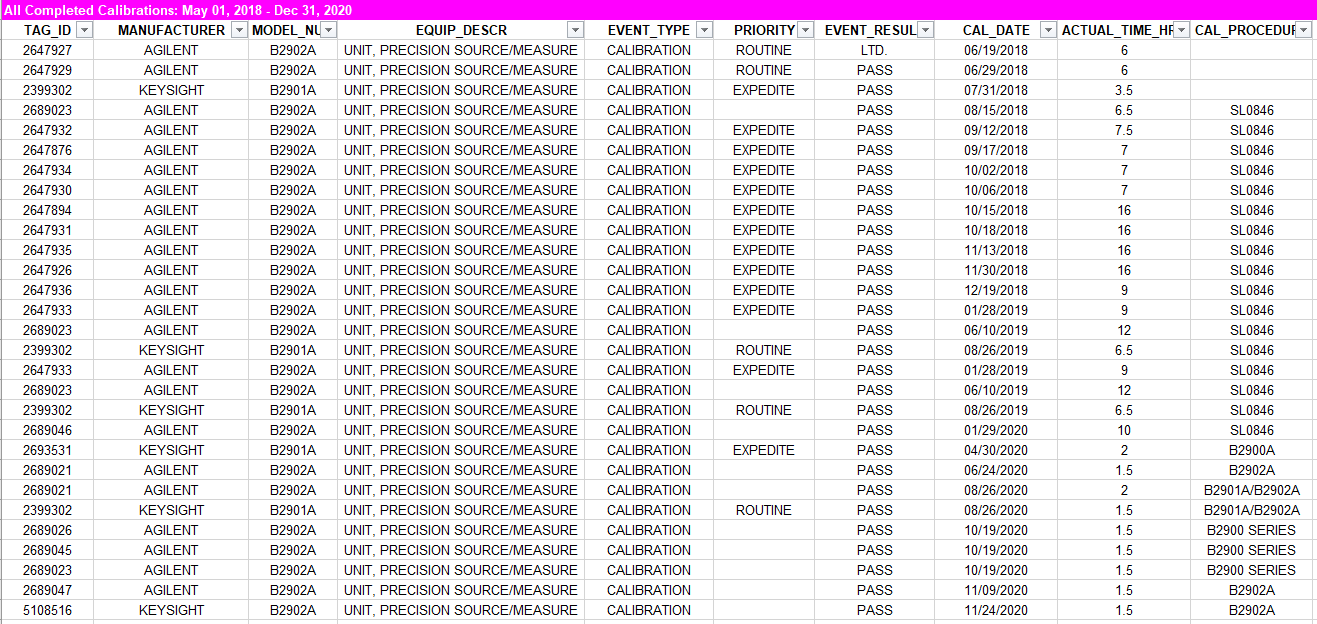


Figure 18: B2900 Series calibration times

But what if that relationship was not built and allowed access to the software? There are cost saving benefits for developing automated procedures, although it may take an initial higher upfront cost. This is the other reason why development happened part time.

Another issue that arises when working part time on a project is that there will be long stretches of time where the project does not get touched, then when it does nothing makes sense or new skills have been learned and therefor the code could use some refactoring. This only leads to extended timelines as well as never completing the project. The development phase ends up being a series of loops rather than a straight line. One of the biggest factors on this issue is not having a team of developers.

## One man team

There has been mention that previous engineers created an API or Python libraries to help in the development process, so there are others with the skills to help develop the Suite, but it goes back to the previous section on time and money or even if those other engineers want to be involved in a big project such as the Software Suite. Being a one-man team makes thinking of and creating test cases a monumental task since end users will always find new and creative ways to use the software or find bugs. Without other team members to provide their input or think of different tests, the released program will be in an alpha or beta state littered with bugs that could cause end users unwilling to use it, not providing feedback on what needs to be fixed.

## Not having access to instruments

Having a suite of instruments included in one software solution is hard to accomplish if there is only one instrument available to develop tests for. The bulk of the instruments that is owned by the company is not owned by the metrology group. Even though there is calibration interval of one year, the recall of the instrument is not mandatory and thus may only come through the metrology group once per three years, or once per ten, or even once in a lifetime of the instrument. What makes the development process of those instruments harder is if there exists only a handful versus being plentiful, and those handful could be owned by a single group opposed to multiple groups within the company. At that point, the calibration of those instruments will still be manual since they would be rare, but if the instruments come through often, even if there are very few, it is still beneficial to develop an automated procedure if it creates a faster calibration process. Not having access to the instruments means that development can only happen based on the manual or previous data reports. Without having the instrument on hand doesn’t allow the Suite to test how long pauses need to be between measurements or if the instrument is reading the command correctly.

# Missed goals

With everything just mentioned, some planned features get put onto the back burner to get the main features completed on time for release. Luckily those planned features don’t impact the final immensely but assist in making the user’s experience more pleasant. One of those features is implementing a tour of the Suite for first time users by using Bootstrap Tour [17].

A screenshot of a computer

Description automatically generated

Figure 19: Bootstrap Tour

Bootstrap Tour takes the user through a step-by-step tour of the program, stopping at specific pages and sections to explain what each one does. The main thing the Suite needs to do upon using it for the first time is to set the user preferences for the server location as well as which lab the user is in and the way they sign off the datasheets. Without the tour, and without any checks at the moment, the Suite throws an error saying it cannot find certain paths or variables within the user\_settings.ini are blank. The Suite will look like it is still working, being able to navigate through the pages, but when the user gets to the tests page and tries to run the tests nothing will happen due to the errors that are present.

Speaking of the user preferences page, there is a section added on the frontend that has not influence on the backend currently. The section is question allows the user to tell the Suite which standards are connected to the computer. The reason this is important is because the library talking to the database has the ability to check with the standards are next due for calibration. In order for a calibration to be valid on the UUT, the standards need to have a valid calibration as well, otherwise that particular standard is not able to be used. The code to check for the calibration date requires the instrument’s identification number. Since that information is not something that can be queried from the standard itself – the identification number is assigned after purchase, which is different than the instrument’s serial number – there would be no way of knowing what that number is unless explicitly listed within the Suite, hence the section in the user preferences page.

A screenshot of a computer

Description automatically generated

Figure 20: Bench section

Other than getting the calibration date, this section is not a huge importance to the Suite since the backend runs a scan on the computer to check connected instruments and see if the proper ones are connected for the calibration procedure, which is seen on Figure 15: Instrument connection page of the Suite. Changing the Python library to allow searching for serial numbers in combination with model numbers, since serial numbers are supposed to be unique but they are unique across the same model number but not across different models or manufacturers, would fully make this section unnecessary and cut out even more interaction the end user has with the Suite.

Another feature that was planned to be implemented was to have the tests pause if there is a failure. MetCal has this feature, and it is very helpful because the failure could be to having the connections incorrect or it could be an actual failure of the UUT. Stopping the tests will let the technician or engineer evaluate what the cause of the failure is at the moment it happens opposed to going through the entire process and then having to go back and repeat everything. In the event of an incorrect connection, it can be fixed at the first instance of a failure without it trickling down to the other tests. The reason this may happen is because the technician or engineer could become too comfortable with the calibration process and not look at the connection image or description and slip up by connecting a cable in the wrong location or are tired from a long day’s work and misconnect on accident.

A screenshot of a test results

Description automatically generated

Figure 21: MetCal failure dialogue box

The MetCal feature shows the setpoint, the result, the error difference, and the tolerance which would all be very useful for the Suite to have since it will be an overlay that would obscure the main window. Although the overlay could move, having all the information on it would ease the work needed from the user.

# Looking to the future

Although all the previously mentioned goals were not accomplished in this timeline, they are still going to be developed and incorporated into the Suite in the future. On top of that, there is a planned feature that needs more research but would be a major improvement over all other software solutions. This feature is the ability to reconnect to the instrument(s) if there is a loss of communication. The reason this can happen is because either the commands are being sent too quickly and the instrument gets overwhelmed and freezes up, or if the reset command is sent too many times it can also freeze up. There are instruments that are older and have internal issues that causes loss of connection and need a hard reset, or power cycling the instrument, then the instrument should be working properly again until the next internal issue happens again. The Suite would try to reconnect to the instrument two to three times before notifying the end user that they need to step in and hard reset to try the reconnection process again.

# In Conclusion

The development of the JPL Software Suite marks a significant step forward in addressing the challenges associated with existing calibration software. By offering a unified solution that incorporates the strengths of previous systems while mitigating their limitations, the Suite aims to be the most robust tool available for the metrology group. Built with Python, one of the most widely-used and accessible programming languages, the Suite provides an approachable and flexible platform for both current and future development. This decision is pivotal for the long-term viability of the project, as future engineers and technicians will be able to easily adapt and contribute to the software’s growth. The Suite’s ability to handle multiple instruments, customizable test parameters, and providing clear visual instructions enhances its usability compared to existing solutions. Furthermore, by integrating features such as custom Python libraries and APIs that communicate with the database, the Suite ensures that users can perform calibration tasks with minimal input, allowing the backend to handle the heavy lifting.

However, the development of the Suite has not been without its challenges. The project has been largely the effort of a single developer working part-time, which has led to delays in implementing some planned features. Despite these constraints, the progress made thus far is impressive, and the foundational work has been completed. Future improvements, such as error-handling mechanisms, tutorials for first time users, and automated communication recovery, are expected to make the software even more robust. The ongoing development of features and improvements will ensure that the Suite remains a leading solution in the field, offering flexibility, ease of use, and precision. By continuing to focus on user needs and incorporating automation, the Suite can significantly reduce the time and effort required to perform calibrations while maintaining high standards of accuracy. The long-term vision for the Suite is to expand its capabilities and make it the go-to software for calibration tasks within the metrology group at JPL. With enough time and development, hopefully the Suite can expand out to other groups and companies within the NASA ecosystem or beyond.

# References

[1] NIST. (2024). Metrology. [Online]. Available: <https://www.nist.gov/metrology>

[2] NASA Satern Training. (2020). Measurement Decision Risk. [Online]. Available: <https://satern.nasa.gov/>

[3] NASA Satern Training. (2020). Measurement Uncertainty Analysis. [Online]. Available: <https://satern.nasa.gov/>

[4] Keyence. (2024). Measurement Fundamentals. [Online]. Available: <https://www.keyence.com/ss/products/measure-sys/measurement-selection/basic/method.jsp>

[5] Fluke. (2024). MetCal Software. [Online]. Available: <https://us.flukecal.com/products/calibration-software/met-cal%C2%AE-software>

[6] Fluke. (2024). Fluke Training. [Online] Available: <https://us.flukecal.com/training>

[7] Northrup Grumman. (2024). SureCal Calibration Software. [Online]. Available: <https://www.northropgrumman.com/cyber/surecal-calibration-software>

[8] National Instruments. (2024). What is NI LabVIEW? [Online]. Available: <https://www.ni.com/en/shop/labview.html?cid=PSEA-7013q000001fLK5AAM-CONS-GOGSE_161597904920&utm_keyword=labview&gad_source=1&gclid=Cj0KCQjwxsm3BhDrARIsAMtVz6N-2SUj56uB-BZ87dyrSeJXVI_AxOklcnzQBTiPZiE1ISQW3XhMDYQaAq3nEALw_wcB#521715&s_kwcid=AL!6304!3!691563588225!b!!g!!labview>

[9] Flatiron School. (2023). Python Popularity: The Rise of A Global Programming Language. [Online]. Available: <https://flatironschool.com/blog/python-popularity-the-rise-of-a-global-programming-language/>

[10] Django Project. (2024). Django Project. [Online]. Available: <https://www.djangoproject.com/>

[11] Pallets Projects. (2024). Flask. [Online]. Available: <https://flask.palletsprojects.com/en/3.0.x/>

[12] GitHub. (2023). Python-eel. [Online]. Available: <https://github.com/python-eel/Eel>

[13] Pallets Projects. (2024). Jinja. [Online]. Available: <https://palletsprojects.com/projects/jinja/>

[14] Jet Propulsion Laboratory. (2023). JPL Brand Toolkit for Applications. [Online]. Available: <https://designlab.jpl.nasa.gov/toolkit/jpl-brand/applications/#JplApplications>

[15] Bootstrap. (2024). Bootstrap. [Online]. Available: <https://getbootstrap.com/>

[16] Tech Target (2023). Definition: Thread. [Online]. Available: <https://www.techtarget.com/whatis/definition/thread>

[17] Bootstrap Tour. (2024). Bootstrap Tour. [Online]. Available: <https://bootstraptour.com/>