THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF ELECTRICAL ENGINEERING

Design and Validation of a Low Cost 8-Channel qRT-LAMP Test with Real-Time Result Readout

DEAN DEROSA
SPRING 2022

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Electrical Engineering
with honors in Electrical Engineering

Reviewed and approved* by the following:

Weihua Guan
Associate Professor of Electrical Engineering
Associate Professor of Biomedical Engineering (Courtesy)
Thesis Supervisor

Julio Urbina
Associate Professor of Electrical Engineering
Honors Adviser

* Electronic approvals are on file.

ABSTRACT

Widespread, fast, and accurate viral screenings are a necessity to track the spread of disease, inform local policy decisions, and mitigate the future spread. Real time reverse-transcriptase polymerase chain reaction (qRT-PCR) is an effective method of diagnosing viral infections yet suffers from excessive complexity in its assay design and implementation, as well as a steep per-unit cost of benchtop devices. Comparable results can be realized using real-time reverse-transcriptase loop-mediated-isothermal amplification (qRT-LAMP), built from components available for a fraction of the total cost of a benchtop PCR machine. This paper outlines the design of the iNAAT, a system that performs a multiplexed eight-channel severe acute respiratory syndrome coronavirus-2 (SARS CoV-2) screening using qRT-LAMP, with the assistance of a smartphone that controls the initiation of the test and displays the measured fluorescent signal.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to extend gratitude to Dr. Weihua Guan and all of his graduate students for welcoming me into their lab, and their friendliness and helpfulness in answering all of my questions. A special thank you is in order for Aneesh Kshirsager, with whom I worked most closely, for overseeing the completion of this project and giving me guidance in how to approach technical problems. I would also like to thank my family and friends for providing me with the motivation and encouragement required to excel in my studies throughout my time at Penn State.

iNAAT Assay Design

Basics of RT-PCR and RT-LAMP

As of November 2020, the COVID-19 pandemic cost an estimated $16 trillion in reduced commerce and diminished work time productivity [1]. As of March 2022, the pandemic also resulted in an estimated 6.1 million deaths worldwide [2]. The CDC has reiterated the importance of fast and accurate tests to slow the spread of COVID-19 by identifying infected individuals and employing contact tracing to socially isolate other at-risk individuals [3]. Throughout the pandemic many Americans had difficulty finding testing sites [4].

Real time reverse-transcriptase polymerase chain reaction (qRT-PCR) is a tool used to quantify the concentration of a specific mRNA sequence in a biological sample. It is often referred to as the “gold standard” for viral screenings, because of its high selectivity, sensitivity, versatility across a diverse range of genome sequences, and efficacy when performed on different types of biological samples [5].

qRT-PCR involves three distinct processes, themselves composed of constituent steps: the production of complementary DNA strands (cDNA) from an RNA sample using reverse transcriptase, the exponential amplification of the cDNA sample using classic PCR, and the quantification of amplification using a fluorescent signal. PCR is composed of three major steps: denaturation, the splitting of the double strand DNA; annealing, the binding of the primers to the single strand DNA; and extension, when the transcriptase catalyzes the binding of the single nucleotides to the genetic sequence of interest [6].

As an alternative to qRT-PCR, real-time reverse-transcriptase loop-mediated-isothermal amplification (qRT-LAMP) is used. While PCR uses a thermo-cycler to change the temperature at which each reaction takes place, LAMP takes place at a constant temperature. This reduces the dimensionality of the optimization. While the design of both LAMP and PCR assays must select the most selective and sensitive genome sequence, PCR must optimize the three temperatures and three reaction times, while LAMP must only optimize one [7].

All nucleic acid amplification must trigger additional cycles of itself. PCR uses thermocycling to denature, anneal and extend DNA. LAMP uses six priming sites, split into a forward group and a backward group each composed of three primers. The middle primer within each group is chained to the complementary sequence of the first (innermost) primer. It anneals to the middle sequence of interest. Then, the annealing of the outermost primer causes strand displacement of the initial primer. Since this freed structure contains both the innermost primer and its complement, it forms a loop structure. A loop introduces instability to subsequent annealing and elongation sequences. From here, a chain reaction begins, where a loop structure keeps getting extended, while ‘shedding’ another loop structure that itself gets extended. A visual of this process can be found in figure 1. Initial trials of LAMP were able to detect 100 copies of an HBV target. [8]

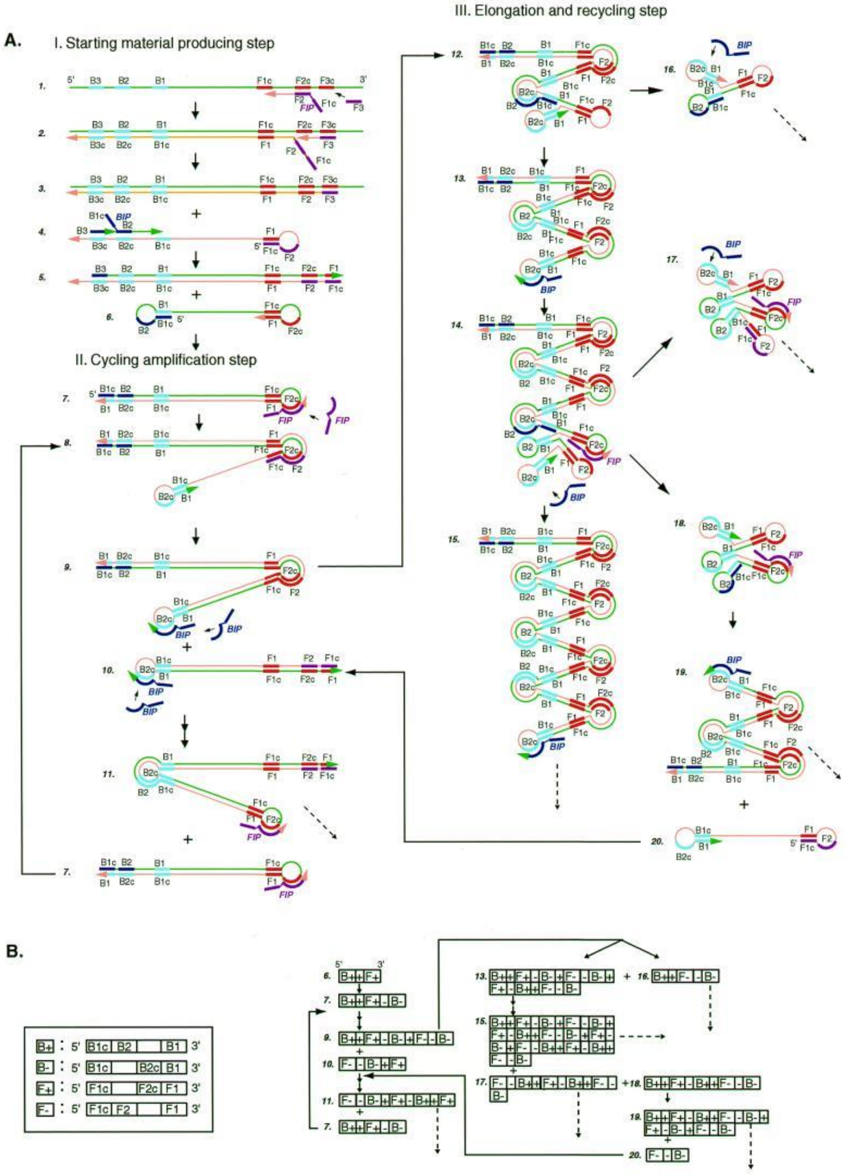
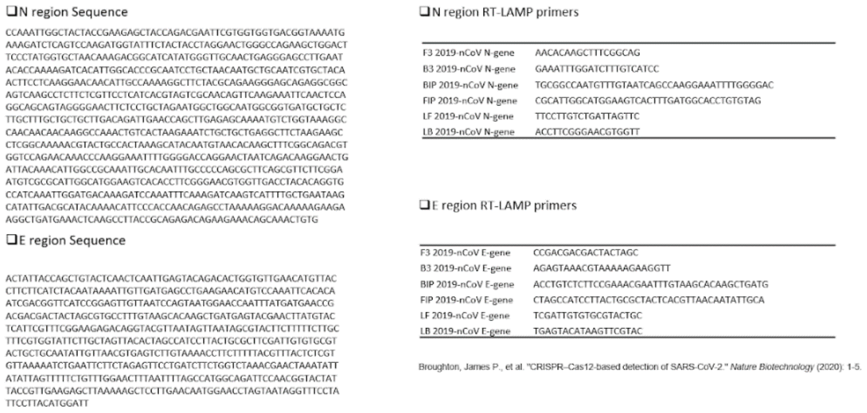

A.
I. Starting material producing step
II. Cycling amplification step
III. Elongation and recycling step
FIP
BIP
B.
B+ : 5' B1c B2 B1 3'
B- : 5' B1c B2c B1 3'
F+ : 5' F1c F2c F1 3'
F- : 5' F1c F2 F1 3'

Figure 1- Visual representation of LAMP, adapted from [8]

Selection of RT-LAMP Assay Genetic Sequence of Interest

Broughton et.al. created primers targeting the nucleocapsid (N) gene and the envelop (E) gene of SARS CoV-2 for use in RT-LAMP tests [8]. The complete genetic sequences of interest and corresponding primer sequences are shown in figure 2. Tang et. al. compared the performance of the aforementioned primer sets [9]. The tests were performed on heat-inactivated SARS-CoV-2 RNA samples at varying concentrations from 2 copies/µL to 2x10^{5} copies/µL. The tests took place at a constant temperature of 65°C. Tests using the N gene primers detected two out of three positively spiked saliva samples with 20 copies/µL, while a test using the E gene primers detected 0 out of 3 identical samples. Additionally, the time to positive, the point at which the fluorescence signal crosses the positive threshold, did not exceed 10 minutes for the N gene primer tests at all concentrations, while time to positive for the E gene primer tests did not fall below 13 minutes for all concentrations. The results are shown in figure 3. As a result, the iNAAT amplifies the N gene sequence.



❑N region Sequence

```
CCAAATTGGCTACTACCGAAGAGCTACCAGACGAATTCGTGGTGGTGACGGTAAAATG
AAAGATCTCAGTCCAAGATGGTATTTCTACTACCTAGGAACTGGGCCAGAAGCTGGACT
TCCCTATGGTGCTAACAAAGACGGCATCATATGGGTTGCAACTGAGGGAGCCTTGAAT
ACACCAAAAGATCACATTGGCACCCGCAATCCTGCTAACAATGCTGCAATCGTGCTACA
ACTTCCTCAAGGAACAACATTGCCAAAAGGCTTCTACGCAGAAGGGAGCAGAGGCGGC
AGTCAAGCCTCTTCTCGTTCCTCATCACGTAGTCGCAACAGTTCAAGAAATTCAACTCCA
GGCAGCAGTAGGGGAACTTCTCCTGCTAGAATGGCTGGCAATGGCGGTGATGCTGCTC
TTGCTTTGCTGCTGCTTGACAGATTGAACCAGCTTGAGAGCAAAATGTCTGGTAAAGGC
CAACAACAACAAGGCCAAACTGTCACTAAGAAATCTGCTGCTGAGGCTTCTAAGAAGC
CTCGGCAAAAACGTACTGCCACTAAAGCATACAATGTAACACAAGCTTTCGGCAGACGT
GGTCCAGAACAAACCCAAGGAAATTTTGGGGACCAGGAACTAATCAGACAAGGAACTG
ATTACAAACATTGGCCGCAAATTGCACAATTTGCCCCCAGCGCTTCAGCGTTCTTCGGA
ATGTCGCGCATTGGCATGGAAGTCACACCTTCGGGAACGTGGTTGACCTACACAGGTG
CCATCAAATTGGATGACAAAGATCCAAATTTCAAAGATCAAGTCATTTTGCTGAATAAG
CATATTGACGCATACAAAACATTCCCACCAACAGAGCCTAAAAAGGACAAAAAGAAGA
AGGCTGATGAAACTCAAGCCTTACCGCAGAGACAGAAGAAACAGCAAACTGTG
```

❑E region Sequence

```
ACTATTACCAGCTGTACTCAACTCAATTGAGTACAGACACTGGTGTTGAACATGTTAC
CTTCTTCATCTACAATAAAATTGTTGATGAGCCTGAAGAACATGTCCAAATTCACACA
ATCGACGGTTCATCCGGAGTTGTTAATCCAGTAATGGAACCAATTTATGATGAACCG
ACGACGACTACTAGCGTGCCTTTGTAAGCACAAGCTGATGAGTACGAACTTATGTAC
TCATTCGTTTCGGAAGAGACAGGTACGTTAATAGTTAATAGCGTACTTCTTTTTCTTGC
TTTCGTGGTATTCTTGCTAGTTACACTAGCCATCCTTACTGCGCTTCGATTGTGTGCGT
ACTGCTGCAATATTGTTAACGTGAGTCTTGTAAAACCTTCTTTTTACGTTTACTCTCGT
GTTAAAAATCTGAATTCTTCTAGAGTTCCTGATCTTCTGGTCTAAACGAACTAAATATT
ATATTAGTTTTTCTGTTTGGAACTTTAATTTTAGCCATGGCAGATTCCAACGGTACTAT
TACCGTTGAAGAGCTTAAAAAGCTCCTTGAACAATGGAACCTAGTAATAGGTTTCCTA
TTCCTTACATGGATT
```

❑N region RT-LAMP primers

F3 2019-nCoV N-gene	AACACAAGCTTTCGGCAG
B3 2019-nCoV N-gene	GAAATTTGGATCTTTGTCATCC
BIP 2019-nCoV N-gene	TGCGGCCAATGTTTGTAATCAGCCAAGGAAATTTTGGGGAC
FIP 2019-nCoV N-gene	CGCATTGGCATGGAAGTCACTTTGATGGCACCTGTGTAG
LF 2019-nCoV N-gene	TTCCTTGTCTGATTAGTTC
LB 2019-nCoV N-gene	ACCTTCGGGAACGTGGTT

❑E region RT-LAMP primers

F3 2019-nCoV E-gene	CCGACGACGACTACTAGC
B3 2019-nCoV E-gene	AGAGTAAACGTAAAAAGAAGGTT
BIP 2019-nCoV E-gene	ACCTGTCTCTTCCGAAACGAATTTGTAAGCACAAGCTGATG
FIP 2019-nCoV E-gene	CTAGCCATCCTTACTGCGCTACTCACGTTAACAATATTGCA
LF 2019-nCoV E-gene	TCGATTGTGTGCGTACTGC
LB 2019-nCoV E-gene	TGAGTACATAAGTTCGTAC

Broughton, James P., et al. "CRISPR–Cas12-based detection of SARS-CoV-2." *Nature Biotechnology* (2020): 1-5.

Figure 2- Genetic sequences of N and E regions, with corresponding primer sequences.

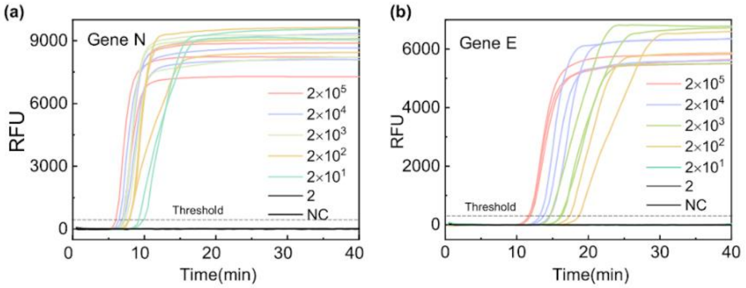


Figure 3- Performance of N-gene (a) and E-gene (b) in RT-LAMP test at varying concentrations (copies/µL).

iNAAT Hardware Design

Components of a Low-Cost RT-LAMP Test

A device conducting a multiplexed RT-LAMP test has several requirements. It must be able to:

- House tubes containing saliva samples
- Block ambient light from penetrating the test chambers
- Regulate the temperature of the test chambers
- Emit light to excite the sample
- Quantify the fluorescent signal
- Switch between channels being observed
- Reliably store information for later use
- Wirelessly send data to a smartphone
- Accomplish all the above in a consistent, reliable, and user-friendly manner

Because assay validation was performed at 65°C, it is the target temperature of the heating element. For rapid prototyping, a 3D-printed shell can enclose the device. With a differential voltage applied, a resistive heating element can dissipate power to achieve a temperature increase. A MOSFET and thermistor-based temperature sensor can provide the negative feedback required to maintain a stable temperature. LEDs emit light to excite the LAMP mix, and RGB color sensors convert a fluorescent signal into a digital one. An embedded computing unit can coordinate the switching of the MOSFET and LEDs, read the information from the color sensors, and store the data in memory. An external view of the iNAAT prototype can be seen in figure 4.

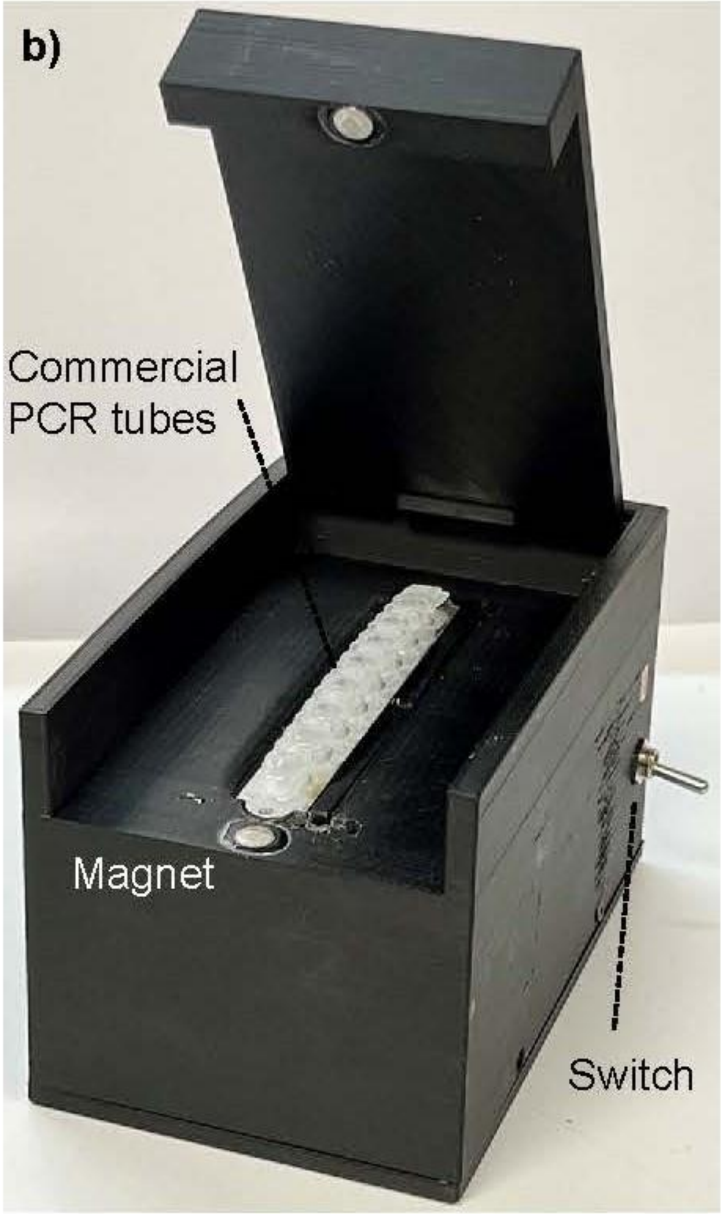


Figure 4- Outer view of iNAAT 3D printed shell.

Selection of Key Components

Vishay Semiconductor's VEML3328 RGBCIR is a photodiode-based color sensor with 16-bit resolution and an inter-integrated circuit (I^2C) communication interface [10]. It has a suitably high spectral responsivity in its red channel, as shown in figure 5. It has configurable gain and integration time via a control register. Despite the LAMP sample emitting green light, the red color sensor is used to quantify the signal because the green sensor will also respond to the blue LEDs.

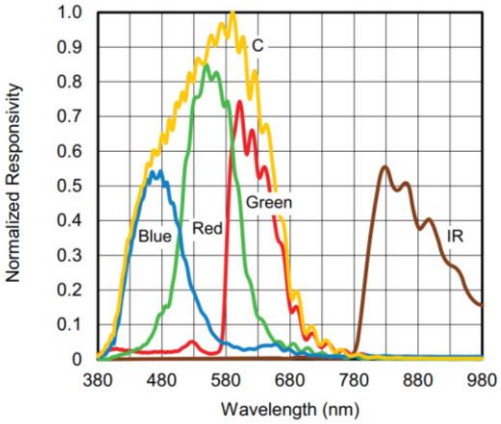


Figure 5- Photoresponsivity characteristics of R, G, B, C, and IR sensors in VEML3328 [10].

Four 1Ω MP725 Surface Mount Power Film Resistors in series with a 9 V applied bias produce the heat necessary to incubate the samples at 65°C. The resistors have a power rating of 25 W, allowing a comfortably large safety margin [11].

A Raspberry Pi Zero W was chosen as the embedded computing platform. It features 26 general purpose input/output (GPIO) pins and one I^2C data and clock bus. Compared to the similarly sized bare-metal Arduino Uno, it has more RAM (512 MB vs 2 KB) and a higher clock speed (1 GHz vs 16 MHz) [12] [13]. Additionally, the Raspberry Pi Zero W is run on a Debian-based operating system. This comes with BlueZ, Linux's implementation of the Bluetooth protocol stack, which allows the device to form a connection with a smartphone and respond to data read and write requests [14].

PCB Design

Two custom printed circuit boards were manufactured that house the electronic components. The schematics and board are shown in Appendix A. Design considerations included having nine mm equidistant spacing of the LEDs and color sensors to match the measurements of the aluminum heating block and making sure the dimensions can fit within the 3D printed shell.

Estimated Hardware Cost

Off the shelf benchtop PCR machines vary in cost from under $1,000 to $10,000 [15] [16]. An estimated bill of materials to produce a single unit is shown in table 1. This estimate neglects the cost of several components:

- The 3D printed case; it was produced with an in-lab unit
- The aluminum case; its per unit cost is difficult to estimate since it was custom manufactured
- All other circuit components (MOSFET and LEDs pull-up resistors), as each have negligible cost

Of note is that the VEML3328 and TCA9548 are not available off the shelf, due to the ongoing supply chain crisis. Instead, MikroElectronika's Color 10 Click sensor and Adafruit's TCA9548 were purchased, which contain the respective integrated circuits that could be desoldered and reused.

Component	Quantity	Total Cost
Raspberry Pi Zero W	1	$10 [12]
Color 10 Click	8	$56 [15]
MP725 Resistor	4	$35 [16]
Adafruit TCA9548	1	$7 [17]
LED PCB	1	$15
Color Sensor PCB	1	$15
Control PCB	1	$30
Total		$168

Table 1- Estimated total cost of iNAAT prototype

Raspberry Pi Software Design

State Diagram of Software Architecture

A state diagram of the Raspberry Pi and iOS app are shown in figures 6 and 7. respectively. The Raspberry Pi's diagram contains the value of its Bluetooth generic attribute (GATT) profile characteristics. Its default state is Idle. It registers an advertisement that can be viewed by any Bluetooth compatible that is scanning. Upon a request to establish connection from an iPhone, the Raspberry Pi will automatically accept. To prevent other devices from pairing and disrupting the test flow, the device will stop advertising, and will only begin again when the device disconnects. When the phone initiates a test, the resistive heating element will be switched on. When it reaches 65°C, it will periodically turn the LEDs on, read the value of the color sensor, then turn the LEDs off. It will repeat until 60 readings have taken place. Test data will not be erased until the phone explicitly tells the Raspberry Pi to end a test. If the device loses power during a test, it will resume the test upon receiving power again.

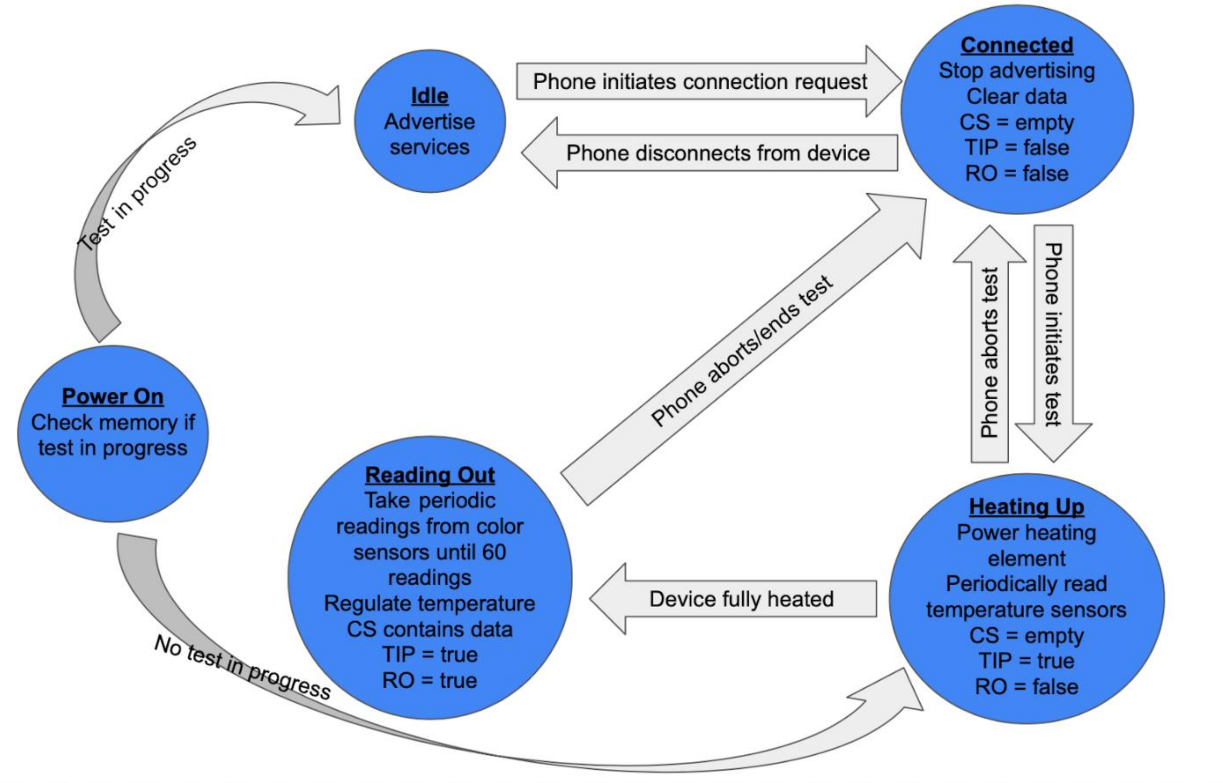


Figure 6- State diagram of Raspberry Pi software. Each state displays actions taken and value of BLE characteristics.

The iPhone has a simpler design. It will specify a device serial number and will scan only for iNAAT devices with the serial number. The phone will determine whether the device is conducting a test. If a test is in progress, it reads the data from its color sensors, only if the device is done heating up. Upon completion of a test, the iPhone stores the data, and tells the

Raspberry Pi to erase its test data.

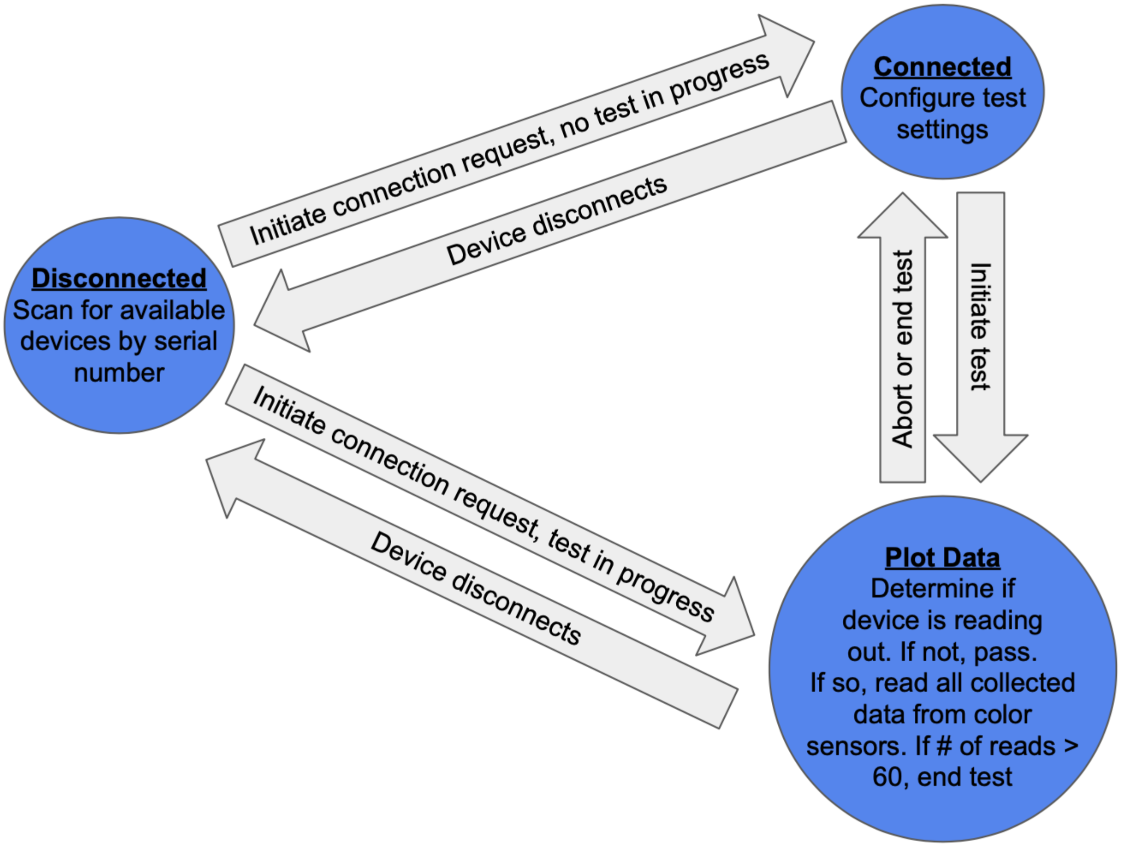

Connected
Configure test settings
Disconnected
Scan for available devices by serial number
Plot Data
Determine if device is reading out. If not, pass. If so, read all collected data from color sensors. If # of reads > 60, end test
Initiate connection request, no test in progress
Device disconnects
Abort or end test
Initiate test
Initiate connection request, test in progress
Device disconnects

Figure 7- State diagram of iOS app software.

Workflow of Raspberry Pi

The iNAAT is designed to operate in a headless state. At boot time, systemd launches a service which runs main.py. Screenshots of all Raspberry Pi files are included in Appendix B. This script creates and registers the advertisement data, and the GATT profile services and characteristics. It then spawns two threads. The first checks whether a test is in progress and the device is heated up. If so, it takes a fluorescence reading. If not, it sleeps. The second checks whether a test is in progress. If so, it begins regulating the temperature. It simply takes a temperature reading, switches the MOSFET on or off if the temperature is less than or greater than 65°C respectively. Once the temperature initially crosses 65°C, the thread signals that the device can begin taking fluorescence readings.

One shared instance BTCharManager, defined in value_manager_class.py, serves as an API to multiple threads that can atomically change and monitor the state of the device. As shown in the state diagram above, the state information consists of whether a test is in progress, whether the device is heated up, and eight arrays of the color sensor readings.

Bluetooth Programming Using BlueZ

BlueZ is Linux's implementation of the Bluetooth protocol stack. It uses Py-DBus, a framework to facilitate inter-process communication with Python bindings. Bluetooth Low Energy (BLE) GATT profiles, advertisements, advertisement managers etc. are modeled as DBus objects. Each object has an interface which defines methods to which it will respond and signals it will emit.

For example, the method call ReadValue sent to a GATT Characteristic will execute a registered callback and transmit the returned value to the requesting device. The method call WriteValue modifies a characteristic's value. Each time it is called on a writeable characteristic, the values are stored using the BTCharManager instance. DBus registers an event loop to process all callbacks asynchronously. Raspberry Pi code can be seen in Appendix B. BlueZ's repository contains examples that strongly influenced many of the scripts on the Raspberry Pi. These are not pictured in Appendix B. See [18] for all examples and source code.

iOS App Software Design

Necessity of an iOS App

As stated previously, the Raspberry Pi is intended to operate in a headless state, with no external intervention. The user should, ideally, not be concerned about the Raspberry Pi's inter-workings. The user must be provided an user-friendly API to control the Raspberry Pi that restricts the actions that the user can take.

Workflow of iOS App

Screenshots of the files involved to make the iOS app can be found in Appendix C. Screenshots of the app itself can be found in Appendix D. When the iOS app is launched, it immediately scans for the previously connected iNAAT device, if any. The user can select the serial number of any iNAAT device that it wishes to scan for and initiate a connection if any devices are found. A user can enter the names or ID of a test sample and specify which channels of the iNAAT will contain a test sample.

When a test is initiated, the device will write 0x01 to the iNAAT's TestInProgress characteristic. The iNAAT's Raspberry Pi will recognize this and initiate a test. When the iOS app reads that the Reading Out characteristic is equal to 0x01, then it begins reading the color sensors. Color sensors are displayed in real time using SwiftUICharts, an external library. At any time, a test can be aborted.

When a test is completed, as distinguished by a read request that returns more than 60 entries per channel, the results of each channel in use are displayed. A positive result is identified by the median of its last five entries being greater than the threshold fluorescence. The user can choose to save the results. It is only at this point that the iOS app will tell the Raspberry Pi to clear its test data.

Constructing iOS App Views

State information dictates the appearance and behavior of each view within an iOS app. This is desirable since it allows the programmer to decide which UI elements show themselves and how each will react to a gesture. In the Swift programming language, structs are immutable value types. In SwiftUI, Apple's newest app development framework, views are structs. As the simplest demonstration of managing state information, if one wishes to modify a view's appearance, it marks some properties as @State. If there is an attempt to set a @State property, a new view will be re-rendered. Other techniques exist to pass consistent data between views and for all views to access a common object [19]. State information is used extensively in the iNAAT iOS app. For example, whenever the app receives new fluorescence signal data, it re-renders the real-time plot.

Each Swift view follows a lifecycle, pictured in figure 8 [20]. Each view's init() method must not access the struct's properties, or else it will give undefined results. When a view is first in sight, its onAppear(perform:) method is triggered, and when a view is no longer in view, its onDisappear(perform:) method is triggered. In between, all changes to @State variables are tracked.

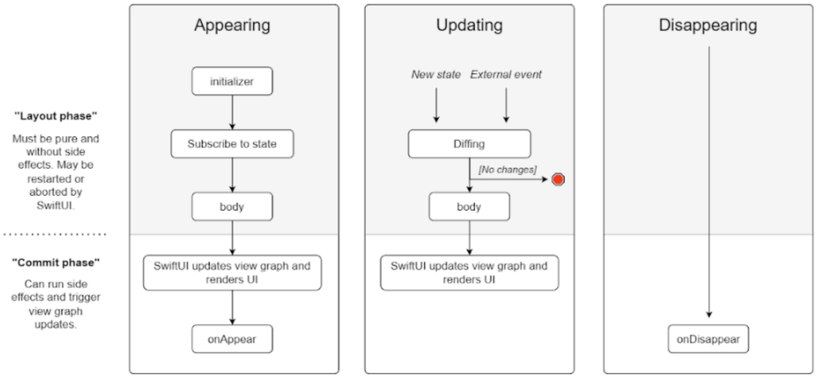

"Layout phase"
Must be pure and without side effects. May be restarted or aborted by SwiftUI.
"Commit phase"
Can run side effects and trigger view graph updates.
Appearing
initializer
Subscribe to state
body
SwiftUI updates view graph and renders UI
onAppear
Updating
New state
External event
Diffing
[No changes]
body
SwiftUI updates view graph and renders UI
Disappearing
onDisappear

Figure 8- iOS view lifecycle and updating of state information.

Bluetooth Programming Using Core Bluetooth

Core Bluetooth is Apple’s Bluetooth Low Energy framework [21]. The iOS app performs the “central” role since it scans for devices and initiates connection. The Raspberry Pi is designated a “peripheral” since it advertises data and waits for a connection attempt.

Upon a new connection, the iOS app will check that all services and characteristics of the Raspberry Pi’s GATT server are properly accessible.

Read and write requests to a peripheral device’s GATT server are non-blocking. If data is successfully returned from a read request, the peripheral(_:didUpdateValueFor:error:) callback will run.

Data Persistence Using Core Data

Fundamental to any point-of-care biomedical device platform is persistent data storage. Core Data is Apple’s API to an SQLite database that stores test data [22]. All state information mentioned above will be reset upon an exit and subsequent restarting of an app. Certain data must be stored indefinitely. The iNAAT stores the following information:

- A list of all previously conducted test results, including the recipient's name/ID, testing data and time, and result status.
- Whether a test is in progress
- The data arrays from the last recorded read to the color sensors
- The names and results (if any) for the current test in progress
- Which test channels are in use for the current test
- The serial number of the connected iNAAT

In the event of the user exiting the app, the storage of all vital information will allow the app to restore itself to its previous state and resume execution where it discontinued.

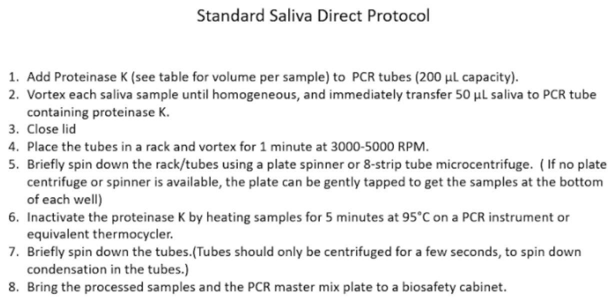
iNAAT Testing and Validation

Experimental Procedure To Compare iNAAT and Benchtop PCR Machine

The FDA has approved saliva as a biomaterial for the detection of SARS-CoV-2 [23]. Due to the difficulty in transporting and preserving saliva samples, the positive saliva samples were manufactured using the Saliva Direct Protocol. Three negative samples and three positive samples spiked at 1024 copies/µL were inputted to the benchtop machine. The procedure followed by GuanLab is diagrammed in figures 9 and 10.

The saliva samples are combined with a LAMP master mix shown in figure 11. Most notably, this mix contains SYTO 9 Green Fluorescent Nucleic Acid Stain, which intercalates in DNA, emitting green light when excited by blue light. As shown in figure 12, its absorption spectra peaks at 485 nm, and its emission spectra peaks at 498 nm [26].

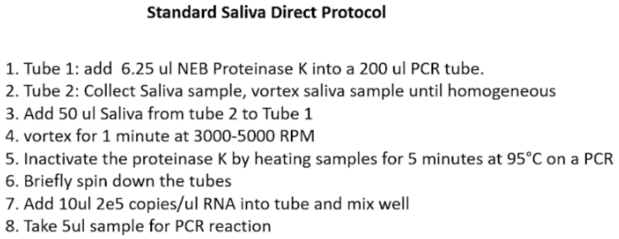
The tests performed used the TCS34725 color sensor. The original plan involved the use of these sensors, but due to their limited availability, the VEML3328 were chosen as a suitable alternative.



Standard Saliva Direct Protocol

1. Add Proteinase K (see table for volume per sample) to PCR tubes (200 µL capacity).
2. Vortex each saliva sample until homogeneous, and immediately transfer 50 µL saliva to PCR tube containing proteinase K.
3. Close lid
4. Place the tubes in a rack and vortex for 1 minute at 3000-5000 RPM.
5. Briefly spin down the rack/tubes using a plate spinner or 8-strip tube microcentrifuge. (If no plate centrifuge or spinner is available, the plate can be gently tapped to get the samples at the bottom of each well)
6. Inactivate the proteinase K by heating samples for 5 minutes at 95°C on a PCR instrument or equivalent thermocycler.
7. Briefly spin down the tubes.(Tubes should only be centrifuged for a few seconds, to spin down condensation in the tubes.)
8. Bring the processed samples and the PCR master mix plate to a biosafety cabinet.

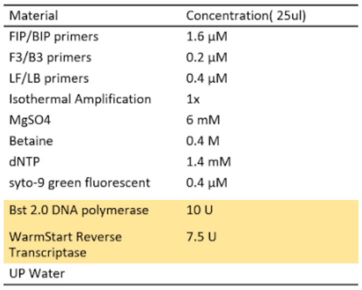
Figure 9- Standard Saliva Direct Protocol followed by GuanLabs pt. 1



Standard Saliva Direct Protocol

1. Tube 1: add 6.25 ul NEB Proteinase K into a 200 ul PCR tube.
2. Tube 2: Collect Saliva sample, vortex saliva sample until homogeneous
3. Add 50 ul Saliva from tube 2 to Tube 1
4. vortex for 1 minute at 3000-5000 RPM
5. Inactivate the proteinase K by heating samples for 5 minutes at 95°C on a PCR
6. Briefly spin down the tubes
7. Add 10ul 2e5 copies/ul RNA into tube and mix well
8. Take 5ul sample for PCR reaction

Figure 10- Standard Saliva Direct Protocol followed by GuanLabs pt. 2



Material	Concentration(25ul)
FIP/BIP primers	1.6 µM
F3/B3 primers	0.2 µM
LF/LB primers	0.4 µM
Isothermal Amplification	1x
MgSO4	6 mM
Betaine	0.4 M
dNTP	1.4 mM
syto-9 green fluorescent	0.4 µM
Bst 2.0 DNA polymerase	10 U
WarmStart Reverse Transcriptase	7.5 U
UP Water	

Figure 11- RT-LAMP master mix combined with saliva for use in RT-LAMP

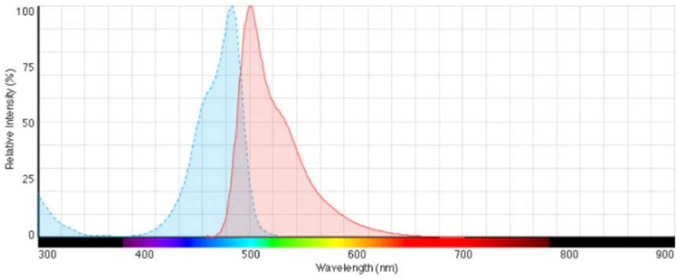


Figure 12- Absorption (left) and emission (right) spectrum of SYTO 9 Green Fluorescent Nucleic Acid Stain [26].

Experimental Results

By using simple electronic components and a computing unit, it is possible to create a significantly less expensive SARS-CoV-2 screening device with comparable performance to a benchtop PCR machine. A comparison of the test performance metrics can be seen in table 2. A plot of relative fluorescence versus time can be seen in figure 13 and 14.

	Benchtop PCR	iNAAT
# of Negative Samples	3	3
# of Positive Samples	3	5
# of True Positives	3	5
# of False Positives	2	1
# of True Negative	1	2
# of False Negatives	0	0

Table 2- Comparison of iNAAT and benchtop PCR machine performance

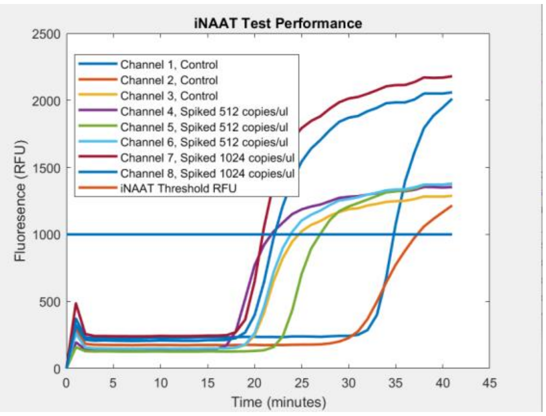


Figure 13- Performance of iNAAT

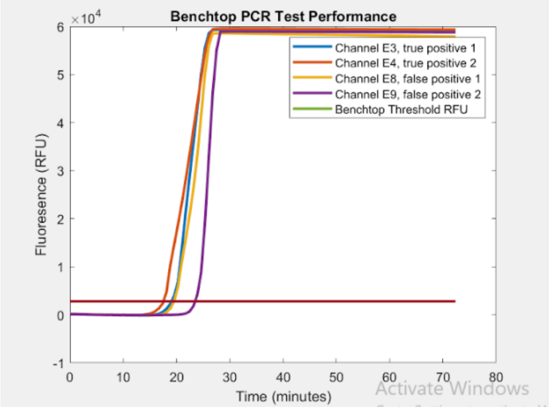


Figure 14- Performance of benchtop PCR machine

Work To Be Done

Experiment on Real Saliva Samples

The tests described above were spiked with SARS-CoV-2 RNA samples. To validate the performance of the iNAAT, it must be tested using real saliva samples.

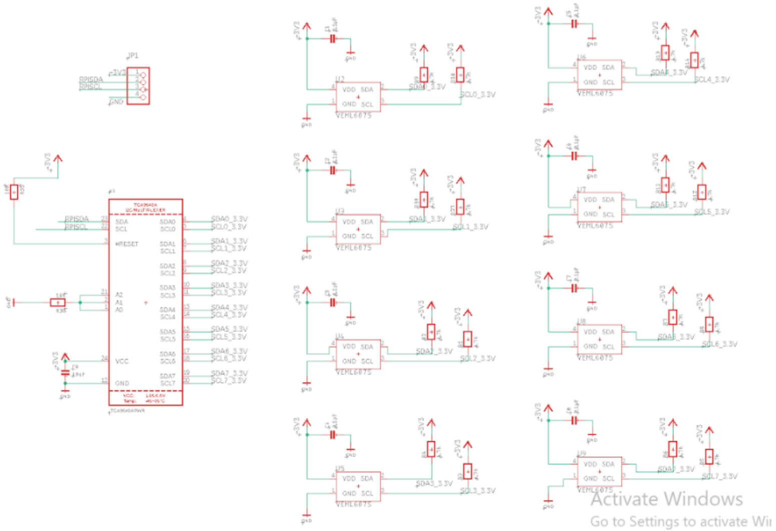
Optimize Testing Parameters

As seen in figure 12, if the iNAAT test were cut at minute 25, there would be only one measured false positive. If it were cut at minute 42, there would be three measured false positives. The time that a test will run must be optimized to capture low concentration positive samples, yet also prevent the amplification of negative samples. Additionally, the threshold RFU level must be optimized for the same reasons. Lastly, since the VEML3328 color sensor gain and integration time are configurable, further experiments must be performed to find the optimal settings.

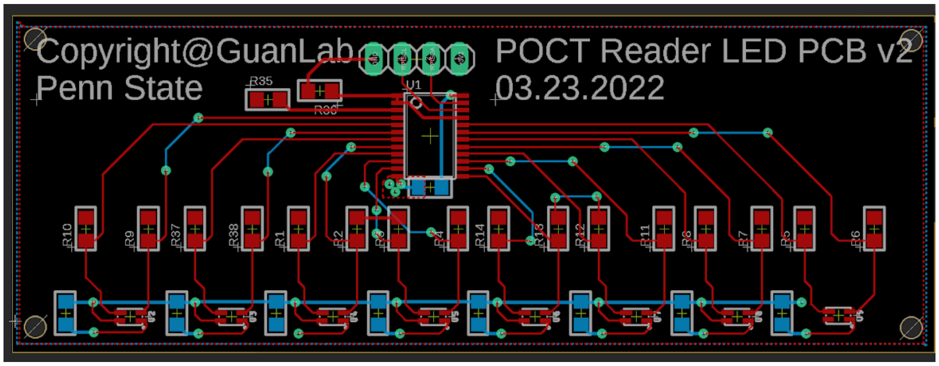
Appendix A

Eagle PCB Files

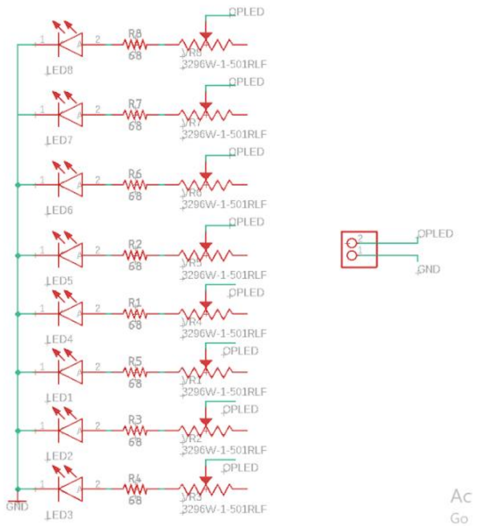
8plex_ColorSensorModule.sch



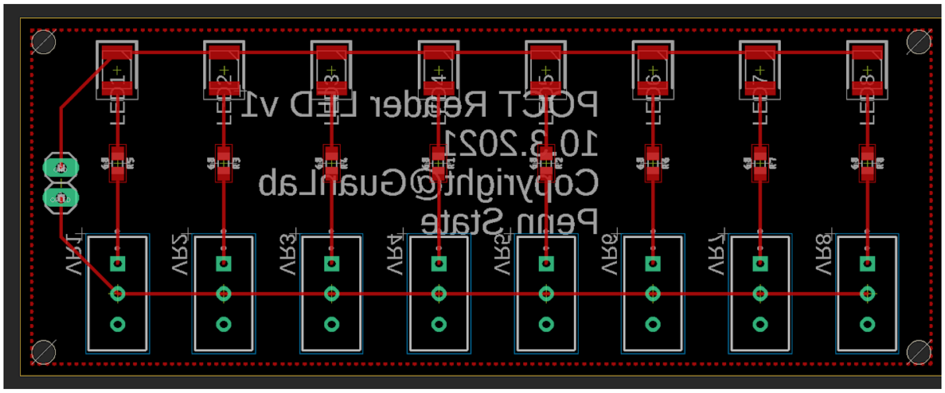
8plex_ColorSensorModule.brd



8plex_LEDModule.sch



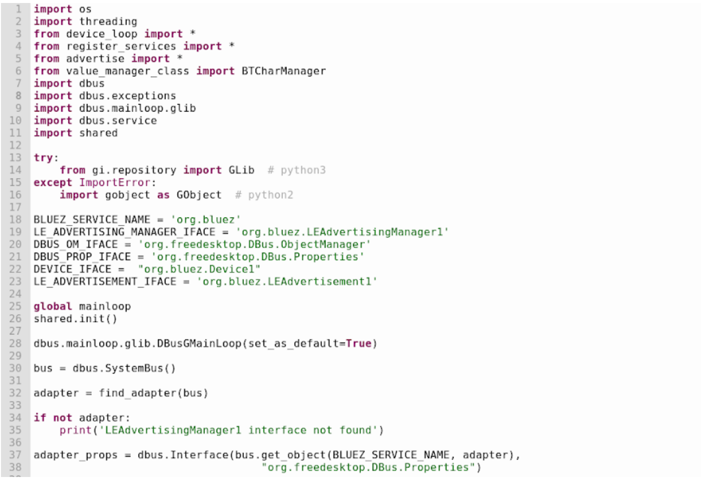
8plex_LEDModule.brd



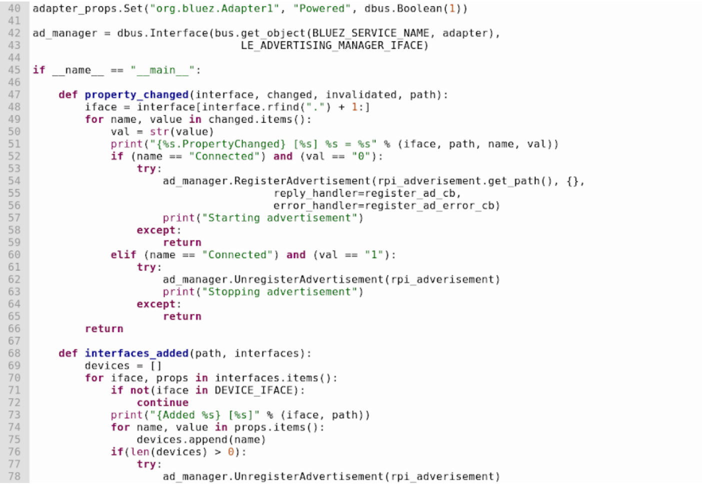
Appendix B

iNAAT Hardware Python Scripts

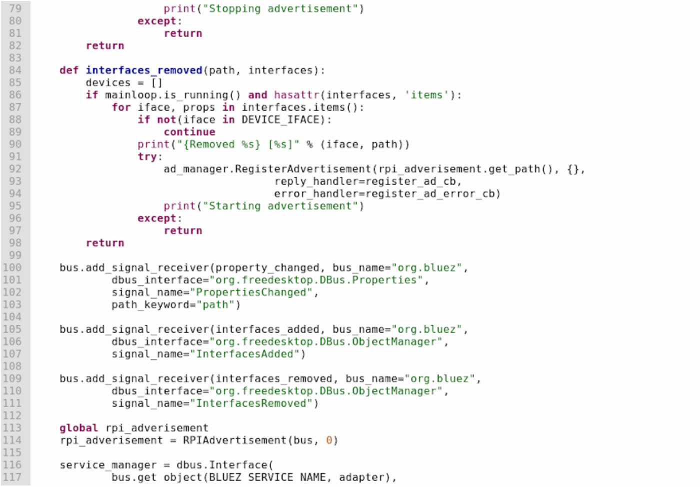
main.py



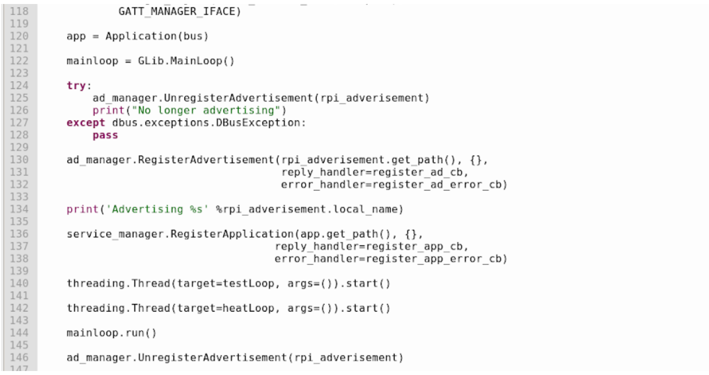
```python
import os
import threading
from device_loop import *
from register_services import *
from advertise import *
from value_manager_class import BTCharManager
import dbus
import dbus.exceptions
import dbus.mainloop.glib
import dbus.service
import shared

try:
    from gi.repository import GLib  # python3
except ImportError:
    import gobject as GObject  # python2

BLUEZ_SERVICE_NAME = 'org.bluez'
LE_ADVERTISING_MANAGER_IFACE = 'org.bluez.LEAdvertisingManager1'
DBUS_OM_IFACE = 'org.freedesktop.DBus.ObjectManager'
DBUS_PROP_IFACE = 'org.freedesktop.DBus.Properties'
DEVICE_IFACE =  "org.bluez.Device1"
LE_ADVERTISEMENT_IFACE = 'org.bluez.LEAdvertisement1'

global mainloop
shared.init()

dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)

bus = dbus.SystemBus()

adapter = find_adapter(bus)

if not adapter:
    print('LEAdvertisingManager1 interface not found')

adapter_props = dbus.Interface(bus.get_object(BLUEZ_SERVICE_NAME, adapter),
                                    "org.freedesktop.DBus.Properties")
```



```python
adapter_props.Set("org.bluez.Adapter1", "Powered", dbus.Boolean(1))

ad_manager = dbus.Interface(bus.get_object(BLUEZ_SERVICE_NAME, adapter),
                                LE_ADVERTISING_MANAGER_IFACE)

if __name__ == "__main__":

    def property_changed(interface, changed, invalidated, path):
        iface = interface[interface.rfind(".") + 1:]
        for name, value in changed.items():
            val = str(value)
            print("{%s.PropertyChanged} [%s] %s = %s" % (iface, path, name, val))
            if (name == "Connected") and (val == "0"):
                try:
                    ad_manager.RegisterAdvertisement(rpi_adverisement.get_path(), {},
                                     reply_handler=register_ad_cb,
                                     error_handler=register_ad_error_cb)
                    print("Starting advertisement")
                except:
                    return
            elif (name == "Connected") and (val == "1"):
                try:
                    ad_manager.UnregisterAdvertisement(rpi_adverisement)
                    print("Stopping advertisement")
                except:
                    return
        return

    def interfaces_added(path, interfaces):
        devices = []
        for iface, props in interfaces.items():
            if not(iface in DEVICE_IFACE):
                continue
            print("{Added %s} [%s]" % (iface, path))
            for name, value in props.items():
                devices.append(name)
            if(len(devices) > 0):
                try:
                    ad_manager.UnregisterAdvertisement(rpi_adverisement)
```

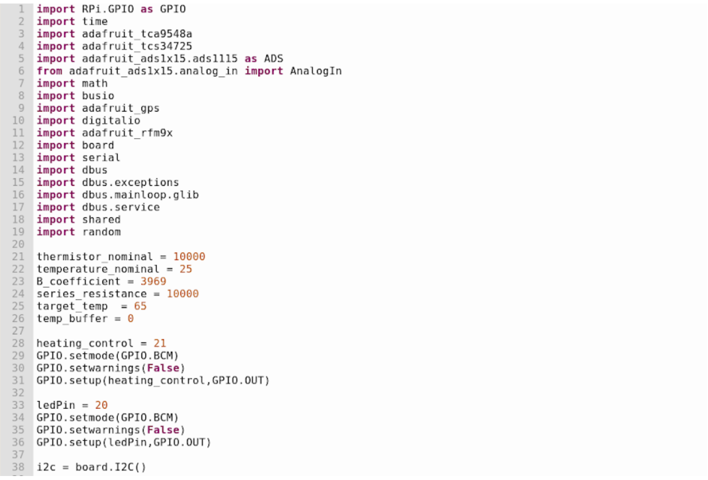


```python
                    print("Stopping advertisement")
                except:
                    return
        return

    def interfaces_removed(path, interfaces):
        devices = []
        if mainloop.is_running() and hasattr(interfaces, 'items'):
            for iface, props in interfaces.items():
                if not(iface in DEVICE_IFACE):
                    continue
                print("{Removed %s} [%s]" % (iface, path))
                try:
                    ad_manager.RegisterAdvertisement(rpi_adverisement.get_path(), {},
                                     reply_handler=register_ad_cb,
                                     error_handler=register_ad_error_cb)
                    print("Starting advertisement")
                except:
                    return
        return

    bus.add_signal_receiver(property_changed, bus_name="org.bluez",
            dbus_interface="org.freedesktop.DBus.Properties",
            signal_name="PropertiesChanged",
            path_keyword="path")

    bus.add_signal_receiver(interfaces_added, bus_name="org.bluez",
            dbus_interface="org.freedesktop.DBus.ObjectManager",
            signal_name="InterfacesAdded")

    bus.add_signal_receiver(interfaces_removed, bus_name="org.bluez",
            dbus_interface="org.freedesktop.DBus.ObjectManager",
            signal_name="InterfacesRemoved")

    global rpi_adverisement
    rpi_adverisement = RPIAdvertisement(bus, 0)

    service_manager = dbus.Interface(
            bus.get object(BLUEZ SERVICE NAME, adapter),
```

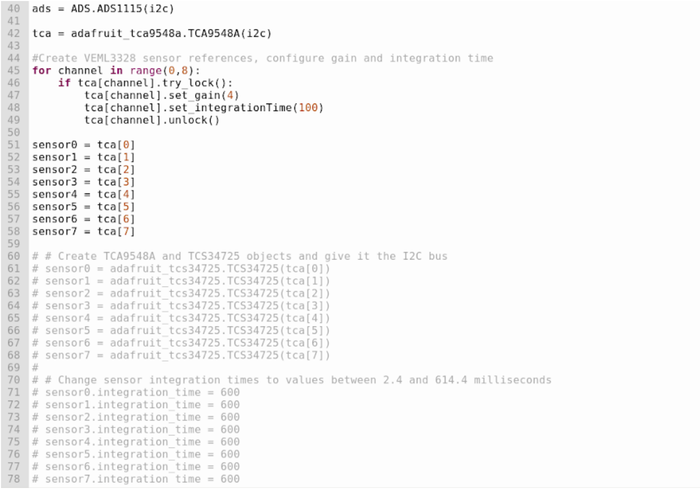


```python
            GATT_MANAGER_IFACE)

    app = Application(bus)

    mainloop = GLib.MainLoop()

    try:
        ad_manager.UnregisterAdvertisement(rpi_adverisement)
        print("No longer advertising")
    except dbus.exceptions.DBusException:
        pass

    ad_manager.RegisterAdvertisement(rpi_adverisement.get_path(), {},
                                     reply_handler=register_ad_cb,
                                     error_handler=register_ad_error_cb)

    print('Advertising %s' %rpi_adverisement.local_name)

    service_manager.RegisterApplication(app.get_path(), {},
                                    reply_handler=register_app_cb,
                                    error_handler=register_app_error_cb)

    threading.Thread(target=testLoop, args=()).start()

    threading.Thread(target=heatLoop, args=()).start()

    mainloop.run()

    ad_manager.UnregisterAdvertisement(rpi_adverisement)
```

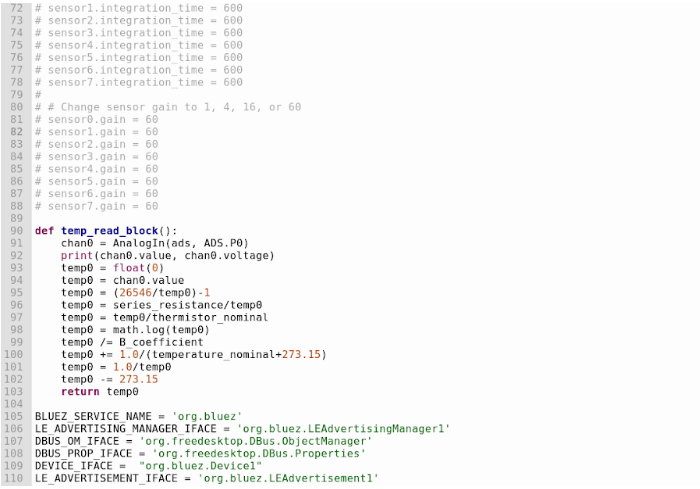
device_loop.py



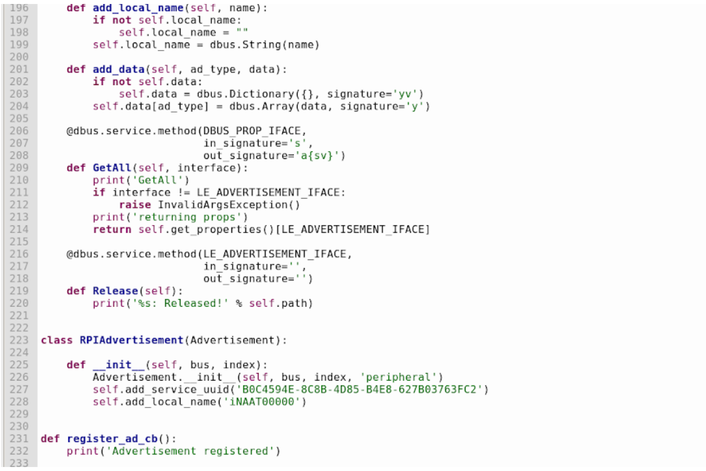
```python
import RPi.GPIO as GPIO
import time
import adafruit_tca9548a
import adafruit_tcs34725
import adafruit_ads1x15.ads1115 as ADS
from adafruit_ads1x15.analog_in import AnalogIn
import math
import busio
import adafruit_gps
import digitalio
import adafruit_rfm9x
import board
import serial
import dbus
import dbus.exceptions
import dbus.mainloop.glib
import dbus.service
import shared
import random

thermistor_nominal = 10000
temperature_nominal = 25
B_coefficient = 3969
series_resistance = 10000
target_temp  = 65
temp_buffer = 0

heating_control = 21
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(heating_control,GPIO.OUT)

ledPin = 20
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(ledPin,GPIO.OUT)

i2c = board.I2C()
```



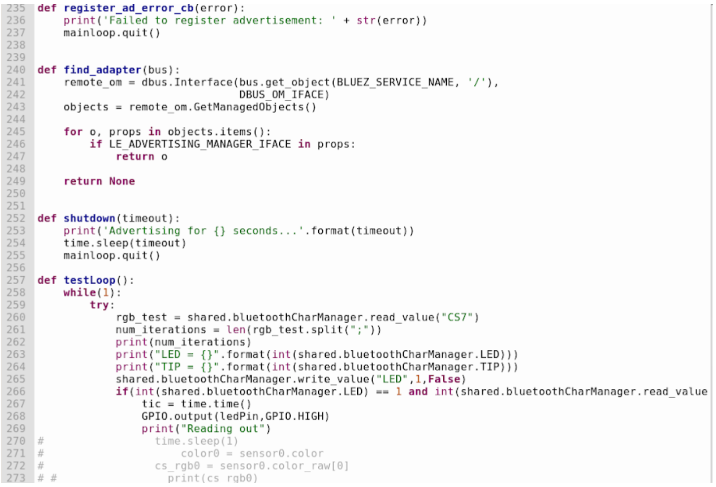
```python
ads = ADS.ADS1115(i2c)

tca = adafruit_tca9548a.TCA9548A(i2c)

#Create VEML3328 sensor references, configure gain and integration time
for channel in range(0,8):
    if tca[channel].try_lock():
        tca[channel].set_gain(4)
        tca[channel].set_integrationTime(100)
        tca[channel].unlock()

sensor0 = tca[0]
sensor1 = tca[1]
sensor2 = tca[2]
sensor3 = tca[3]
sensor4 = tca[4]
sensor5 = tca[5]
sensor6 = tca[6]
sensor7 = tca[7]

# # Create TCA9548A and TCS34725 objects and give it the I2C bus
# sensor0 = adafruit_tcs34725.TCS34725(tca[0])
# sensor1 = adafruit_tcs34725.TCS34725(tca[1])
# sensor2 = adafruit_tcs34725.TCS34725(tca[2])
# sensor3 = adafruit_tcs34725.TCS34725(tca[3])
# sensor4 = adafruit_tcs34725.TCS34725(tca[4])
# sensor5 = adafruit_tcs34725.TCS34725(tca[5])
# sensor6 = adafruit_tcs34725.TCS34725(tca[6])
# sensor7 = adafruit_tcs34725.TCS34725(tca[7])
#
# # Change sensor integration times to values between 2.4 and 614.4 milliseconds
# sensor0.integration_time = 600
# sensor1.integration_time = 600
# sensor2.integration_time = 600
# sensor3.integration_time = 600
# sensor4.integration_time = 600
# sensor5.integration_time = 600
# sensor6.integration_time = 600
# sensor7.integration_time = 600
```



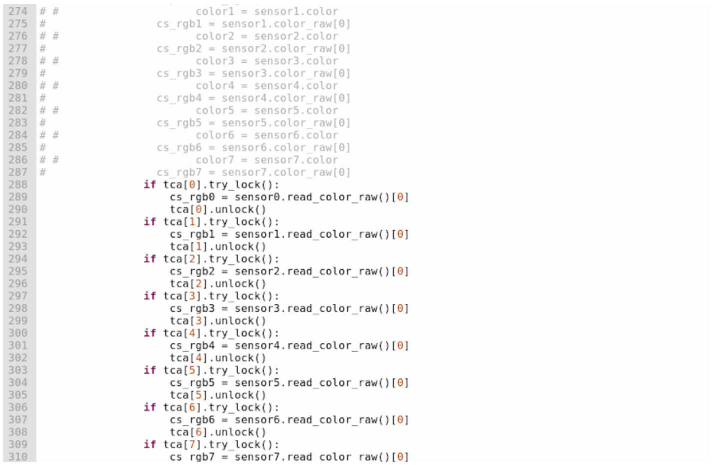
```python
# sensor1.integration_time = 600
# sensor2.integration_time = 600
# sensor3.integration_time = 600
# sensor4.integration_time = 600
# sensor5.integration_time = 600
# sensor6.integration_time = 600
# sensor7.integration_time = 600
#
# # Change sensor gain to 1, 4, 16, or 60
# sensor0.gain = 60
# sensor1.gain = 60
# sensor2.gain = 60
# sensor3.gain = 60
# sensor4.gain = 60
# sensor5.gain = 60
# sensor6.gain = 60
# sensor7.gain = 60

def temp_read_block():
    chan0 = AnalogIn(ads, ADS.P0)
    print(chan0.value, chan0.voltage)
    temp0 = float(0)
    temp0 = chan0.value
    temp0 = (26546/temp0)-1
    temp0 = series_resistance/temp0
    temp0 = temp0/thermistor_nominal
    temp0 = math.log(temp0)
    temp0 /= B_coefficient
    temp0 += 1.0/(temperature_nominal+273.15)
    temp0 = 1.0/temp0
    temp0 -= 273.15
    return temp0

BLUEZ_SERVICE_NAME = 'org.bluez'
LE_ADVERTISING_MANAGER_IFACE = 'org.bluez.LEAdvertisingManager1'
DBUS_OM_IFACE = 'org.freedesktop.DBus.ObjectManager'
DBUS_PROP_IFACE = 'org.freedesktop.DBus.Properties'
DEVICE_IFACE =  "org.bluez.Device1"
LE_ADVERTISEMENT_IFACE = 'org.bluez.LEAdvertisement1'
```



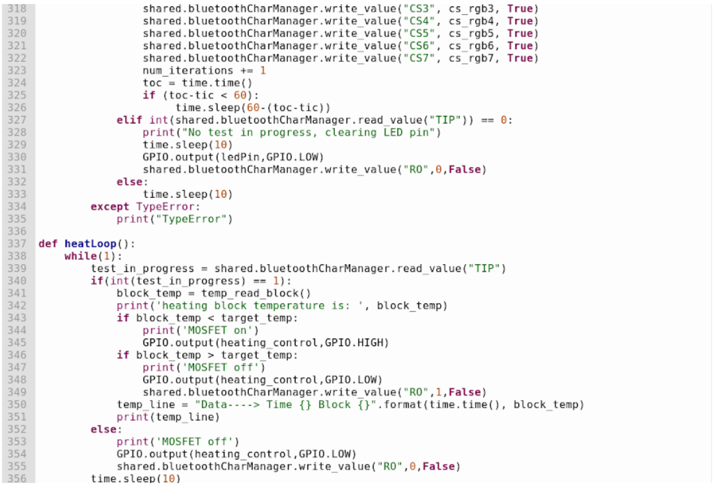
```python
    def add_local_name(self, name):
        if not self.local_name:
            self.local_name = ""
        self.local_name = dbus.String(name)

    def add_data(self, ad_type, data):
        if not self.data:
            self.data = dbus.Dictionary({}, signature='yv')
        self.data[ad_type] = dbus.Array(data, signature='y')

    @dbus.service.method(DBUS_PROP_IFACE,
                         in_signature='s',
                         out_signature='a{sv}')
    def GetAll(self, interface):
        print('GetAll')
        if interface != LE_ADVERTISEMENT_IFACE:
            raise InvalidArgsException()
        print('returning props')
        return self.get_properties()[LE_ADVERTISEMENT_IFACE]

    @dbus.service.method(LE_ADVERTISEMENT_IFACE,
                         in_signature='',
                         out_signature='')
    def Release(self):
        print('%s: Released!' % self.path)


class RPIAdvertisement(Advertisement):

    def __init__(self, bus, index):
        Advertisement.__init__(self, bus, index, 'peripheral')
        self.add_service_uuid('B0C4594E-8C8B-4D85-B4E8-627B03763FC2')
        self.add_local_name('iNAAT00000')


def register_ad_cb():
    print('Advertisement registered')

```



```
def register_ad_error_cb(error):
    print('Failed to register advertisement: ' + str(error))
    mainloop.quit()


def find_adapter(bus):
    remote_om = dbus.Interface(bus.get_object(BLUEZ_SERVICE_NAME, '/'),
                               DBUS_OM_IFACE)
    objects = remote_om.GetManagedObjects()

    for o, props in objects.items():
        if LE_ADVERTISING_MANAGER_IFACE in props:
            return o

    return None


def shutdown(timeout):
    print('Advertising for {} seconds...'.format(timeout))
    time.sleep(timeout)
    mainloop.quit()

def testLoop():
    while(1):
        try:
            rgb_test = shared.bluetoothCharManager.read_value("CS7")
            num_iterations = len(rgb_test.split(";"))
            print(num_iterations)
            print("LED = {}".format(int(shared.bluetoothCharManager.LED)))
            print("TIP = {}".format(int(shared.bluetoothCharManager.TIP)))
            shared.bluetoothCharManager.write_value("LED",1,False)
            if(int(shared.bluetoothCharManager.LED) == 1 and int(shared.bluetoothCharManager.read_value
                tic = time.time()
                GPIO.output(ledPin,GPIO.HIGH)
                print("Reading out")
#                 time.sleep(1)
#                     color0 = sensor0.color
#                 cs_rgb0 = sensor0.color_raw[0]
# #                 print(cs_rgb0)
```

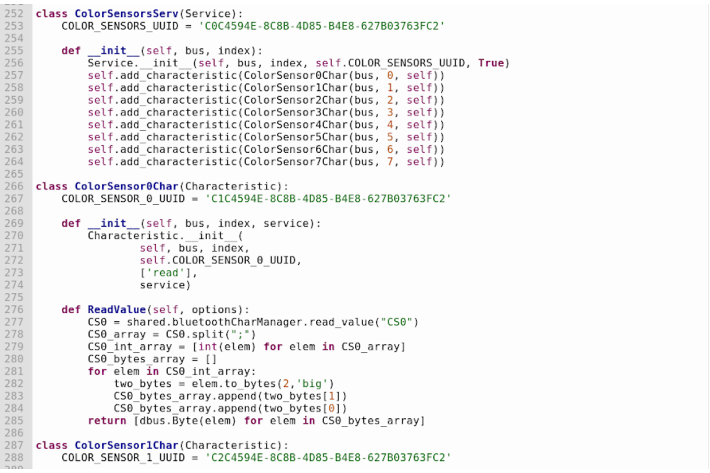


```
# #                     color1 = sensor1.color
#                 cs_rgb1 = sensor1.color_raw[0]
# #                     color2 = sensor2.color
#                 cs_rgb2 = sensor2.color_raw[0]
# #                     color3 = sensor3.color
#                 cs_rgb3 = sensor3.color_raw[0]
# #                     color4 = sensor4.color
#                 cs_rgb4 = sensor4.color_raw[0]
# #                     color5 = sensor5.color
#                 cs_rgb5 = sensor5.color_raw[0]
# #                     color6 = sensor6.color
#                 cs_rgb6 = sensor6.color_raw[0]
# #                     color7 = sensor7.color
#                 cs_rgb7 = sensor7.color_raw[0]
                if tca[0].try_lock():
                    cs_rgb0 = sensor0.read_color_raw()[0]
                    tca[0].unlock()
                if tca[1].try_lock():
                    cs_rgb1 = sensor1.read_color_raw()[0]
                    tca[1].unlock()
                if tca[2].try_lock():
                    cs_rgb2 = sensor2.read_color_raw()[0]
                    tca[2].unlock()
                if tca[3].try_lock():
                    cs_rgb3 = sensor3.read_color_raw()[0]
                    tca[3].unlock()
                if tca[4].try_lock():
                    cs_rgb4 = sensor4.read_color_raw()[0]
                    tca[4].unlock()
                if tca[5].try_lock():
                    cs_rgb5 = sensor5.read_color_raw()[0]
                    tca[5].unlock()
                if tca[6].try_lock():
                    cs_rgb6 = sensor6.read_color_raw()[0]
                    tca[6].unlock()
                if tca[7].try_lock():
                    cs_rgb7 = sensor7.read_color_raw()[0]
```

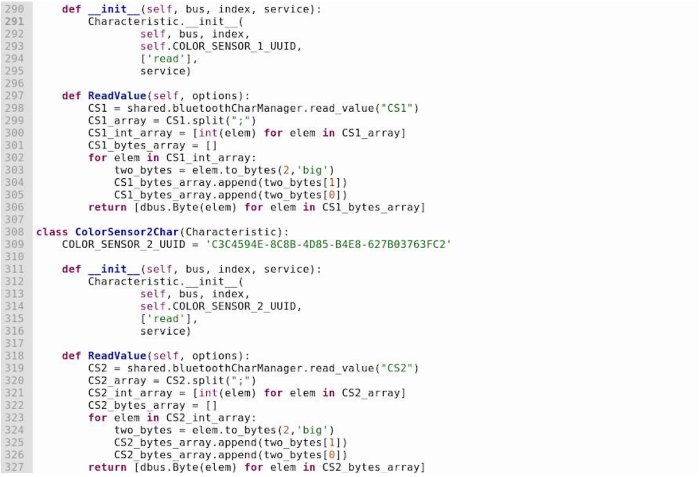


```
                shared.bluetoothCharManager.write_value("CS3", cs_rgb3, True)
                shared.bluetoothCharManager.write_value("CS4", cs_rgb4, True)
                shared.bluetoothCharManager.write_value("CS5", cs_rgb5, True)
                shared.bluetoothCharManager.write_value("CS6", cs_rgb6, True)
                shared.bluetoothCharManager.write_value("CS7", cs_rgb7, True)
                num_iterations += 1
                toc = time.time()
                if (toc-tic < 60):
                     time.sleep(60-(toc-tic))
            elif int(shared.bluetoothCharManager.read_value("TIP")) == 0:
                print("No test in progress, clearing LED pin")
                time.sleep(10)
                GPIO.output(ledPin,GPIO.LOW)
                shared.bluetoothCharManager.write_value("RO",0,False)
            else:
                time.sleep(10)
        except TypeError:
            print("TypeError")

def heatLoop():
    while(1):
        test_in_progress = shared.bluetoothCharManager.read_value("TIP")
        if(int(test_in_progress) == 1):
            block_temp = temp_read_block()
            print('heating block temperature is: ', block_temp)
            if block_temp < target_temp:
                print('MOSFET on')
                GPIO.output(heating_control,GPIO.HIGH)
            if block_temp > target_temp:
                print('MOSFET off')
                GPIO.output(heating_control,GPIO.LOW)
                shared.bluetoothCharManager.write_value("RO",1,False)
            temp_line = "Data----> Time {} Block {}".format(time.time(), block_temp)
            print(temp_line)
        else:
            print('MOSFET off')
            GPIO.output(heating_control,GPIO.LOW)
            shared.bluetoothCharManager.write_value("RO",0,False)
        time.sleep(10)
```

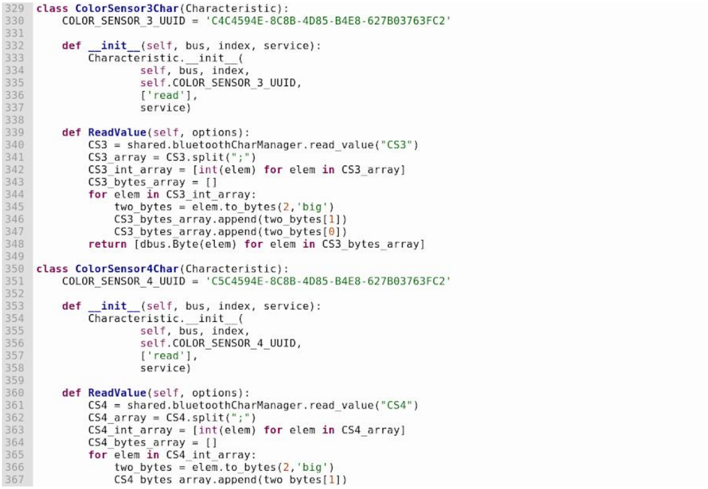
register_services.py



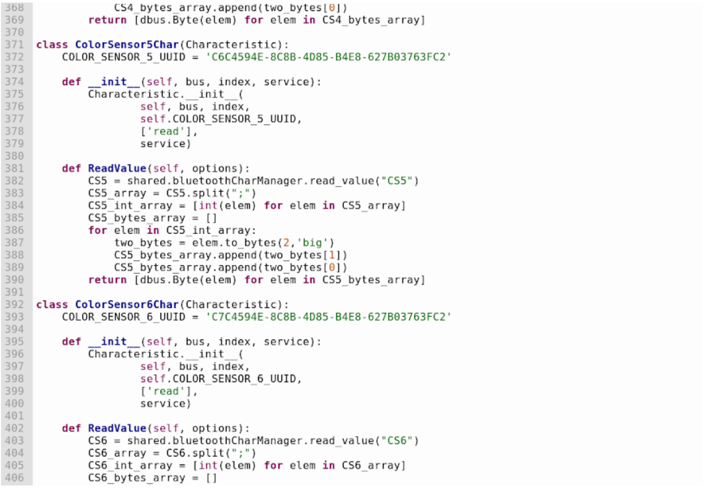
```python
class ColorSensorsServ(Service):
    COLOR_SENSORS_UUID = 'C0C4594E-8C8B-4D85-B4E8-627B03763FC2'

    def __init__(self, bus, index):
        Service.__init__(self, bus, index, self.COLOR_SENSORS_UUID, True)
        self.add_characteristic(ColorSensor0Char(bus, 0, self))
        self.add_characteristic(ColorSensor1Char(bus, 1, self))
        self.add_characteristic(ColorSensor2Char(bus, 2, self))
        self.add_characteristic(ColorSensor3Char(bus, 3, self))
        self.add_characteristic(ColorSensor4Char(bus, 4, self))
        self.add_characteristic(ColorSensor5Char(bus, 5, self))
        self.add_characteristic(ColorSensor6Char(bus, 6, self))
        self.add_characteristic(ColorSensor7Char(bus, 7, self))

class ColorSensor0Char(Characteristic):
    COLOR_SENSOR_0_UUID = 'C1C4594E-8C8B-4D85-B4E8-627B03763FC2'

    def __init__(self, bus, index, service):
        Characteristic.__init__(
                self, bus, index,
                self.COLOR_SENSOR_0_UUID,
                ['read'],
                service)

    def ReadValue(self, options):
        CS0 = shared.bluetoothCharManager.read_value("CS0")
        CS0_array = CS0.split(";")
        CS0_int_array = [int(elem) for elem in CS0_array]
        CS0_bytes_array = []
        for elem in CS0_int_array:
            two_bytes = elem.to_bytes(2,'big')
            CS0_bytes_array.append(two_bytes[1])
            CS0_bytes_array.append(two_bytes[0])
        return [dbus.Byte(elem) for elem in CS0_bytes_array]

class ColorSensor1Char(Characteristic):
    COLOR_SENSOR_1_UUID = 'C2C4594E-8C8B-4D85-B4E8-627B03763FC2'
```



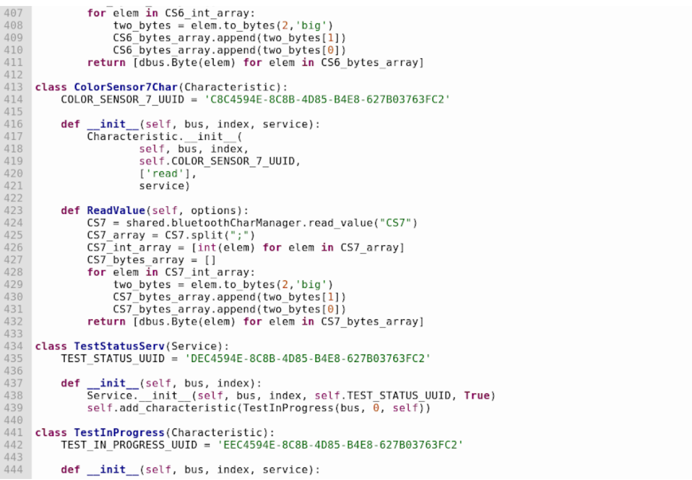
```python
    def __init__(self, bus, index, service):
        Characteristic.__init__(
                self, bus, index,
                self.COLOR_SENSOR_1_UUID,
                ['read'],
                service)

    def ReadValue(self, options):
        CS1 = shared.bluetoothCharManager.read_value("CS1")
        CS1_array = CS1.split(";")
        CS1_int_array = [int(elem) for elem in CS1_array]
        CS1_bytes_array = []
        for elem in CS1_int_array:
            two_bytes = elem.to_bytes(2,'big')
            CS1_bytes_array.append(two_bytes[1])
            CS1_bytes_array.append(two_bytes[0])
        return [dbus.Byte(elem) for elem in CS1_bytes_array]

class ColorSensor2Char(Characteristic):
    COLOR_SENSOR_2_UUID = 'C3C4594E-8C8B-4D85-B4E8-627B03763FC2'

    def __init__(self, bus, index, service):
        Characteristic.__init__(
                self, bus, index,
                self.COLOR_SENSOR_2_UUID,
                ['read'],
                service)

    def ReadValue(self, options):
        CS2 = shared.bluetoothCharManager.read_value("CS2")
        CS2_array = CS2.split(";")
        CS2_int_array = [int(elem) for elem in CS2_array]
        CS2_bytes_array = []
        for elem in CS2_int_array:
            two_bytes = elem.to_bytes(2,'big')
            CS2_bytes_array.append(two_bytes[1])
            CS2_bytes_array.append(two_bytes[0])
        return [dbus.Byte(elem) for elem in CS2_bytes_array]
```



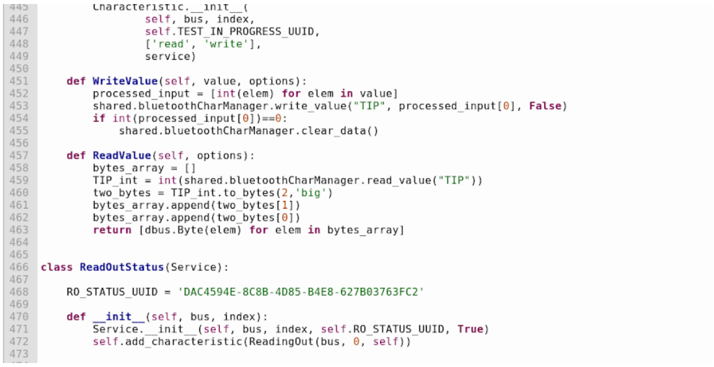
```python
class ColorSensor3Char(Characteristic):
    COLOR_SENSOR_3_UUID = 'C4C4594E-8C8B-4D85-B4E8-627B03763FC2'

    def __init__(self, bus, index, service):
        Characteristic.__init__(
                self, bus, index,
                self.COLOR_SENSOR_3_UUID,
                ['read'],
                service)

    def ReadValue(self, options):
        CS3 = shared.bluetoothCharManager.read_value("CS3")
        CS3_array = CS3.split(";")
        CS3_int_array = [int(elem) for elem in CS3_array]
        CS3_bytes_array = []
        for elem in CS3_int_array:
            two_bytes = elem.to_bytes(2,'big')
            CS3_bytes_array.append(two_bytes[1])
            CS3_bytes_array.append(two_bytes[0])
        return [dbus.Byte(elem) for elem in CS3_bytes_array]

class ColorSensor4Char(Characteristic):
    COLOR_SENSOR_4_UUID = 'C5C4594E-8C8B-4D85-B4E8-627B03763FC2'

    def __init__(self, bus, index, service):
        Characteristic.__init__(
                self, bus, index,
                self.COLOR_SENSOR_4_UUID,
                ['read'],
                service)

    def ReadValue(self, options):
        CS4 = shared.bluetoothCharManager.read_value("CS4")
        CS4_array = CS4.split(";")
        CS4_int_array = [int(elem) for elem in CS4_array]
        CS4_bytes_array = []
        for elem in CS4_int_array:
            two_bytes = elem.to_bytes(2,'big')
            CS4_bytes_array.append(two_bytes[1])
```



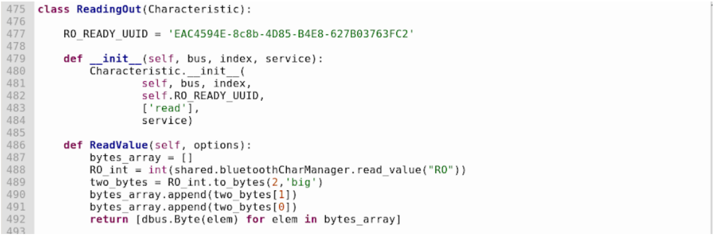
```python
            CS4_bytes_array.append(two_bytes[0])
        return [dbus.Byte(elem) for elem in CS4_bytes_array]

class ColorSensor5Char(Characteristic):
    COLOR_SENSOR_5_UUID = 'C6C4594E-8C8B-4D85-B4E8-627B03763FC2'

    def __init__(self, bus, index, service):
        Characteristic.__init__(
                self, bus, index,
                self.COLOR_SENSOR_5_UUID,
                ['read'],
                service)

    def ReadValue(self, options):
        CS5 = shared.bluetoothCharManager.read_value("CS5")
        CS5_array = CS5.split(";")
        CS5_int_array = [int(elem) for elem in CS5_array]
        CS5_bytes_array = []
        for elem in CS5_int_array:
            two_bytes = elem.to_bytes(2,'big')
            CS5_bytes_array.append(two_bytes[1])
            CS5_bytes_array.append(two_bytes[0])
        return [dbus.Byte(elem) for elem in CS5_bytes_array]

class ColorSensor6Char(Characteristic):
    COLOR_SENSOR_6_UUID = 'C7C4594E-8C8B-4D85-B4E8-627B03763FC2'

    def __init__(self, bus, index, service):
        Characteristic.__init__(
                self, bus, index,
                self.COLOR_SENSOR_6_UUID,
                ['read'],
                service)

    def ReadValue(self, options):
        CS6 = shared.bluetoothCharManager.read_value("CS6")
        CS6_array = CS6.split(";")
        CS6_int_array = [int(elem) for elem in CS6_array]
        CS6_bytes_array = []
```



```python
        for elem in CS6_int_array:
            two_bytes = elem.to_bytes(2,'big')
            CS6_bytes_array.append(two_bytes[1])
            CS6_bytes_array.append(two_bytes[0])
        return [dbus.Byte(elem) for elem in CS6_bytes_array]

class ColorSensor7Char(Characteristic):
    COLOR_SENSOR_7_UUID = 'C8C4594E-8C8B-4D85-B4E8-627B03763FC2'

    def __init__(self, bus, index, service):
        Characteristic.__init__(
                self, bus, index,
                self.COLOR_SENSOR_7_UUID,
                ['read'],
                service)

    def ReadValue(self, options):
        CS7 = shared.bluetoothCharManager.read_value("CS7")
        CS7_array = CS7.split(";")
        CS7_int_array = [int(elem) for elem in CS7_array]
        CS7_bytes_array = []
        for elem in CS7_int_array:
            two_bytes = elem.to_bytes(2,'big')
            CS7_bytes_array.append(two_bytes[1])
            CS7_bytes_array.append(two_bytes[0])
        return [dbus.Byte(elem) for elem in CS7_bytes_array]

class TestStatusServ(Service):
    TEST_STATUS_UUID = 'DEC4594E-8C8B-4D85-B4E8-627B03763FC2'

    def __init__(self, bus, index):
        Service.__init__(self, bus, index, self.TEST_STATUS_UUID, True)
        self.add_characteristic(TestInProgress(bus, 0, self))

class TestInProgress(Characteristic):
    TEST_IN_PROGRESS_UUID = 'EEC4594E-8C8B-4D85-B4E8-627B03763FC2'

    def __init__(self, bus, index, service):
```

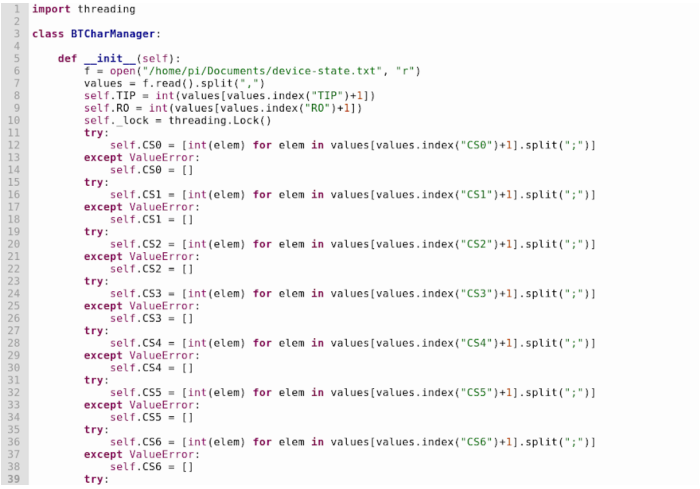


```python
        Characteristic.__init__(
                self, bus, index,
                self.TEST_IN_PROGRESS_UUID,
                ['read', 'write'],
                service)

    def WriteValue(self, value, options):
        processed_input = [int(elem) for elem in value]
        shared.bluetoothCharManager.write_value("TIP", processed_input[0], False)
        if int(processed_input[0])==0:
            shared.bluetoothCharManager.clear_data()

    def ReadValue(self, options):
        bytes_array = []
        TIP_int = int(shared.bluetoothCharManager.read_value("TIP"))
        two_bytes = TIP_int.to_bytes(2,'big')
        bytes_array.append(two_bytes[1])
        bytes_array.append(two_bytes[0])
        return [dbus.Byte(elem) for elem in bytes_array]


class ReadOutStatus(Service):

    RO_STATUS_UUID = 'DAC4594E-8C8B-4D85-B4E8-627B03763FC2'

    def __init__(self, bus, index):
        Service.__init__(self, bus, index, self.RO_STATUS_UUID, True)
        self.add_characteristic(ReadingOut(bus, 0, self))

```

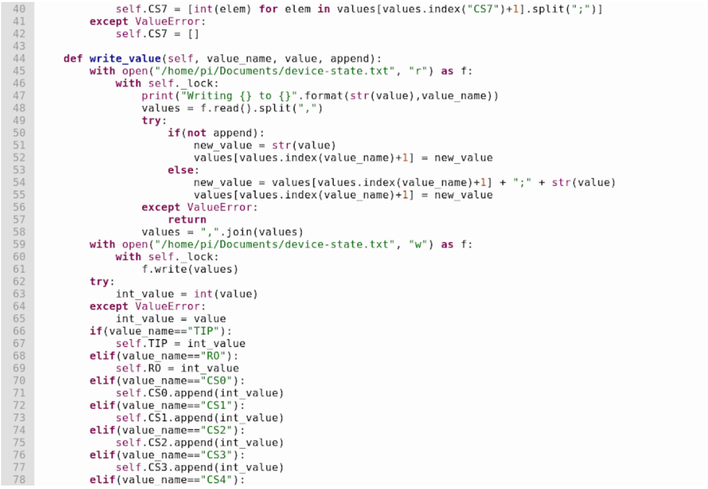


```
class ReadingOut(Characteristic):

    RO_READY_UUID = 'EAC4594E-8c8b-4D85-B4E8-627B03763FC2'

    def __init__(self, bus, index, service):
        Characteristic.__init__(
                self, bus, index,
                self.RO_READY_UUID,
                ['read'],
                service)

    def ReadValue(self, options):
        bytes_array = []
        RO_int = int(shared.bluetoothCharManager.read_value("RO"))
        two_bytes = RO_int.to_bytes(2,'big')
        bytes_array.append(two_bytes[1])
        bytes_array.append(two_bytes[0])
        return [dbus.Byte(elem) for elem in bytes_array]
```

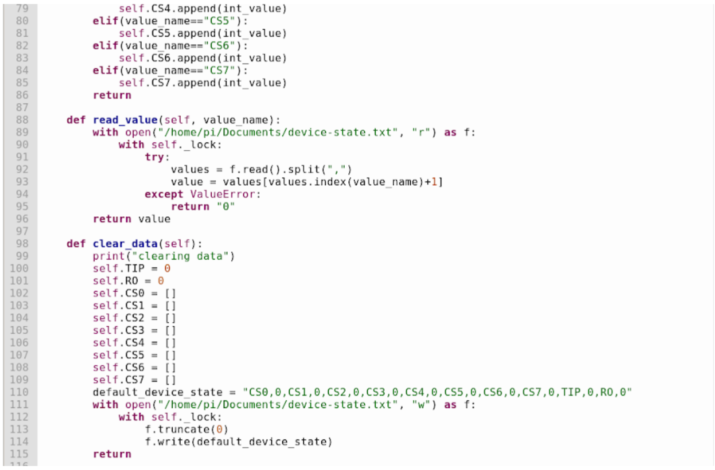
value_manager_class.py



```
import threading

class BTCharManager:

    def __init__(self):
        f = open("/home/pi/Documents/device-state.txt", "r")
        values = f.read().split(",")
        self.TIP = int(values[values.index("TIP")+1])
        self.RO = int(values[values.index("RO")+1])
        self._lock = threading.Lock()
        try:
            self.CS0 = [int(elem) for elem in values[values.index("CS0")+1].split(";")]
        except ValueError:
            self.CS0 = []
        try:
            self.CS1 = [int(elem) for elem in values[values.index("CS1")+1].split(";")]
        except ValueError:
            self.CS1 = []
        try:
            self.CS2 = [int(elem) for elem in values[values.index("CS2")+1].split(";")]
        except ValueError:
            self.CS2 = []
        try:
            self.CS3 = [int(elem) for elem in values[values.index("CS3")+1].split(";")]
        except ValueError:
            self.CS3 = []
        try:
            self.CS4 = [int(elem) for elem in values[values.index("CS4")+1].split(";")]
        except ValueError:
            self.CS4 = []
        try:
            self.CS5 = [int(elem) for elem in values[values.index("CS5")+1].split(";")]
        except ValueError:
            self.CS5 = []
        try:
            self.CS6 = [int(elem) for elem in values[values.index("CS6")+1].split(";")]
        except ValueError:
            self.CS6 = []
        try:
```

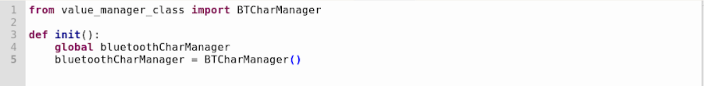


```
            self.CS7 = [int(elem) for elem in values[values.index("CS7")+1].split(";")]
        except ValueError:
            self.CS7 = []

    def write_value(self, value_name, value, append):
        with open("/home/pi/Documents/device-state.txt", "r") as f:
            with self._lock:
                print("Writing {} to {}".format(str(value),value_name))
                values = f.read().split(",")
                try:
                    if(not append):
                        new_value = str(value)
                        values[values.index(value_name)+1] = new_value
                    else:
                        new_value = values[values.index(value_name)+1] + ";" + str(value)
                        values[values.index(value_name)+1] = new_value
                except ValueError:
                    return
                values = ",".join(values)
        with open("/home/pi/Documents/device-state.txt", "w") as f:
            with self._lock:
                f.write(values)
        try:
            int_value = int(value)
        except ValueError:
            int_value = value
        if(value_name=="TIP"):
            self.TIP = int_value
        elif(value_name=="RO"):
            self.RO = int_value
        elif(value_name=="CS0"):
            self.CS0.append(int_value)
        elif(value_name=="CS1"):
            self.CS1.append(int_value)
        elif(value_name=="CS2"):
            self.CS2.append(int_value)
        elif(value_name=="CS3"):
            self.CS3.append(int_value)
        elif(value_name=="CS4"):
```



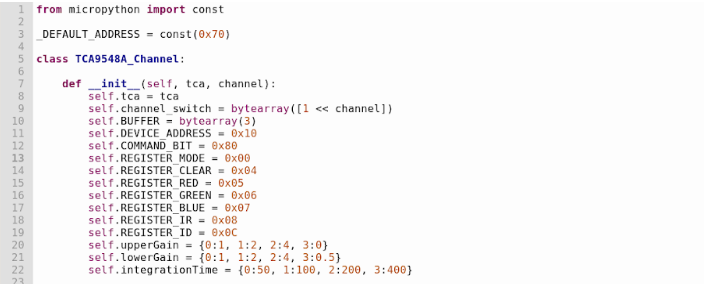
```
            self.CS4.append(int_value)
        elif(value_name=="CS5"):
            self.CS5.append(int_value)
        elif(value_name=="CS6"):
            self.CS6.append(int_value)
        elif(value_name=="CS7"):
            self.CS7.append(int_value)
        return

    def read_value(self, value_name):
        with open("/home/pi/Documents/device-state.txt", "r") as f:
            with self._lock:
                try:
                    values = f.read().split(",")
                    value = values[values.index(value_name)+1]
                except ValueError:
                    return "0"
        return value

    def clear_data(self):
        print("clearing data")
        self.TIP = 0
        self.RO = 0
        self.CS0 = []
        self.CS1 = []
        self.CS2 = []
        self.CS3 = []
        self.CS4 = []
        self.CS5 = []
        self.CS6 = []
        self.CS7 = []
        default_device_state = "CS0,0,CS1,0,CS2,0,CS3,0,CS4,0,CS5,0,CS6,0,CS7,0,TIP,0,RO,0"
        with open("/home/pi/Documents/device-state.txt", "w") as f:
            with self._lock:
                f.truncate(0)
                f.write(default_device_state)
        return
```

shared.py

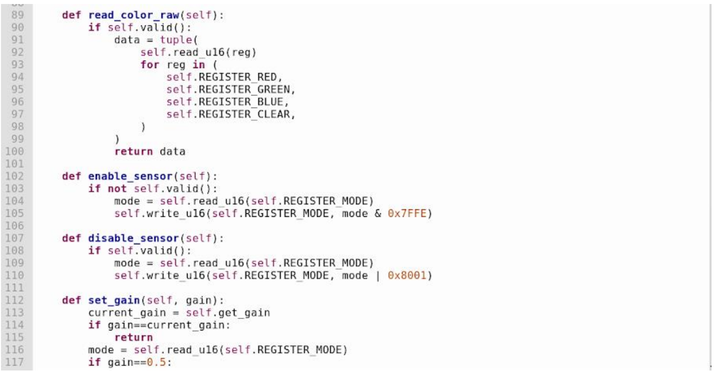


```
from value_manager_class import BTCharManager

def init():
    global bluetoothCharManager
    bluetoothCharManager = BTCharManager()
```

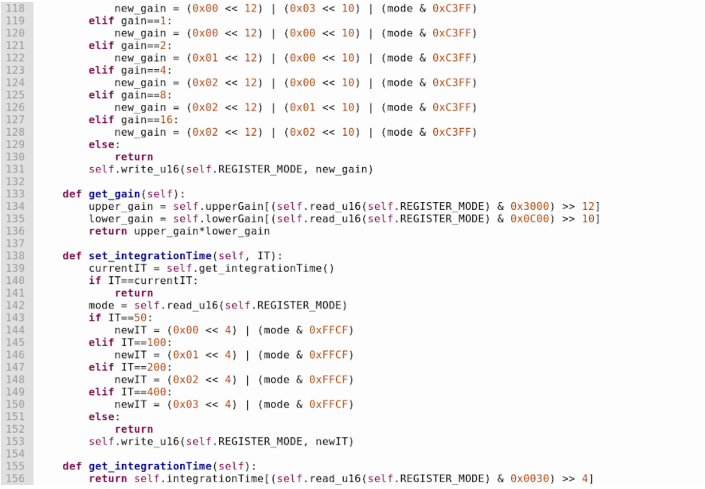
tca9458a.py



```
from micropython import const

_DEFAULT_ADDRESS = const(0x70)

class TCA9548A_Channel:

    def __init__(self, tca, channel):
        self.tca = tca
        self.channel_switch = bytearray([1 << channel])
        self.BUFFER = bytearray(3)
        self.DEVICE_ADDRESS = 0x10
        self.COMMAND_BIT = 0x80
        self.REGISTER_MODE = 0x00
        self.REGISTER_CLEAR = 0x04
        self.REGISTER_RED = 0x05
        self.REGISTER_GREEN = 0x06
        self.REGISTER_BLUE = 0x07
        self.REGISTER_IR = 0x08
        self.REGISTER_ID = 0x0C
        self.upperGain = {0:1, 1:2, 2:4, 3:0}
        self.lowerGain = {0:1, 1:2, 2:4, 3:0.5}
        self.integrationTime = {0:50, 1:100, 2:200, 3:400}
```



```python
    def read_color_raw(self):
        if self.valid():
            data = tuple(
                self.read_u16(reg)
                for reg in (
                    self.REGISTER_RED,
                    self.REGISTER_GREEN,
                    self.REGISTER_BLUE,
                    self.REGISTER_CLEAR,
                )
            )
            return data

    def enable_sensor(self):
        if not self.valid():
            mode = self.read_u16(self.REGISTER_MODE)
            self.write_u16(self.REGISTER_MODE, mode & 0x7FFE)

    def disable_sensor(self):
        if self.valid():
            mode = self.read_u16(self.REGISTER_MODE)
            self.write_u16(self.REGISTER_MODE, mode | 0x8001)

    def set_gain(self, gain):
        current_gain = self.get_gain
        if gain==current_gain:
            return
        mode = self.read_u16(self.REGISTER_MODE)
        if gain==0.5:
```

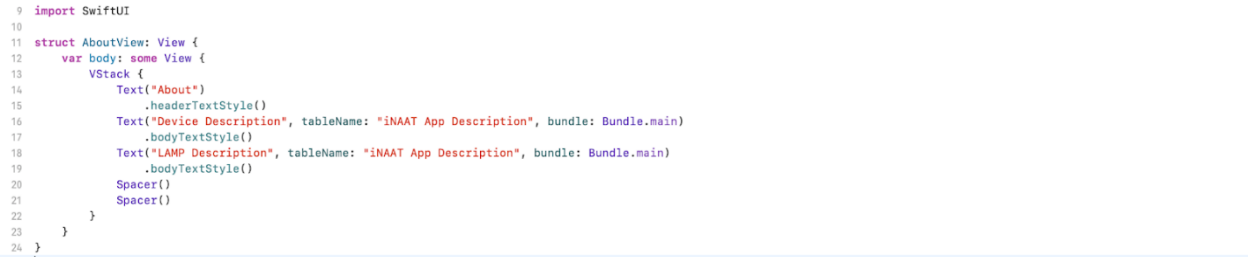


```python
            new_gain = (0x00 << 12) | (0x03 << 10) | (mode & 0xC3FF)
        elif gain==1:
            new_gain = (0x00 << 12) | (0x00 << 10) | (mode & 0xC3FF)
        elif gain==2:
            new_gain = (0x01 << 12) | (0x00 << 10) | (mode & 0xC3FF)
        elif gain==4:
            new_gain = (0x02 << 12) | (0x00 << 10) | (mode & 0xC3FF)
        elif gain==8:
            new_gain = (0x02 << 12) | (0x01 << 10) | (mode & 0xC3FF)
        elif gain==16:
            new_gain = (0x02 << 12) | (0x02 << 10) | (mode & 0xC3FF)
        else:
            return
        self.write_u16(self.REGISTER_MODE, new_gain)

    def get_gain(self):
        upper_gain = self.upperGain[(self.read_u16(self.REGISTER_MODE) & 0x3000) >> 12]
        lower_gain = self.lowerGain[(self.read_u16(self.REGISTER_MODE) & 0x0C00) >> 10]
        return upper_gain*lower_gain

    def set_integrationTime(self, IT):
        currentIT = self.get_integrationTime()
        if IT==currentIT:
            return
        mode = self.read_u16(self.REGISTER_MODE)
        if IT==50:
            newIT = (0x00 << 4) | (mode & 0xFFCF)
        elif IT==100:
            newIT = (0x01 << 4) | (mode & 0xFFCF)
        elif IT==200:
            newIT = (0x02 << 4) | (mode & 0xFFCF)
        elif IT==400:
            newIT = (0x03 << 4) | (mode & 0xFFCF)
        else:
            return
        self.write_u16(self.REGISTER_MODE, newIT)

    def get_integrationTime(self):
        return self.integrationTime[(self.read_u16(self.REGISTER_MODE) & 0x0030) >> 4]
```

Appendix C

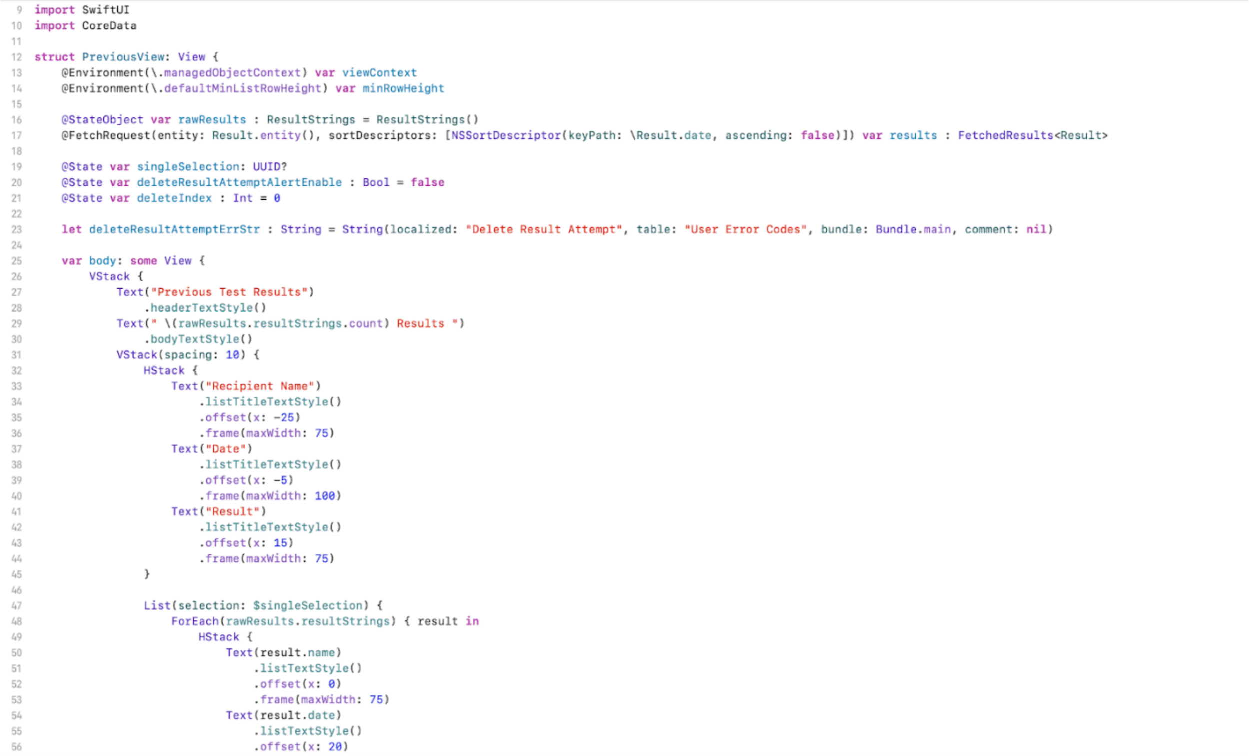
iNAAT iOS App Swift Files

AboutView.swift

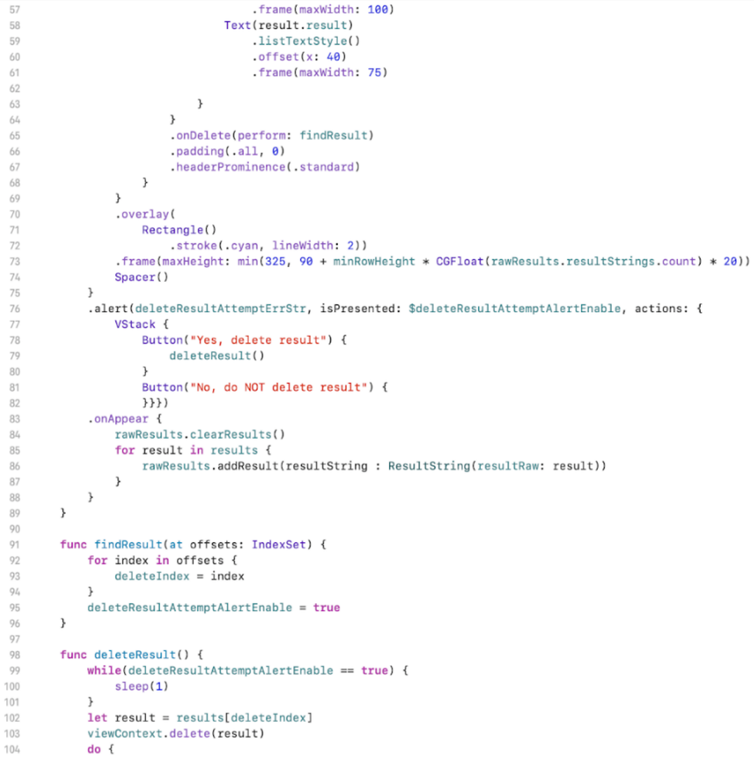


```swift
import SwiftUI

struct AboutView: View {
    var body: some View {
        VStack {
            Text("About")
                .headerTextStyle()
            Text("Device Description", tableName: "iNAAT App Description", bundle: Bundle.main)
                .bodyTextStyle()
            Text("LAMP Description", tableName: "iNAAT App Description", bundle: Bundle.main)
                .bodyTextStyle()
            Spacer()
            Spacer()
        }
    }
}
```

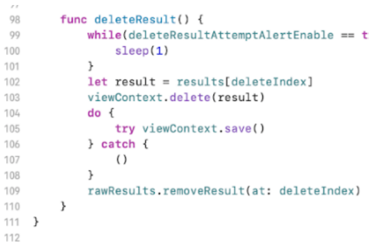
PreviousView.swift



```swift
import SwiftUI
import CoreData

struct PreviousView: View {
    @Environment(\.managedObjectContext) var viewContext
    @Environment(\.defaultMinListRowHeight) var minRowHeight

    @StateObject var rawResults : ResultStrings = ResultStrings()
    @FetchRequest(entity: Result.entity(), sortDescriptors: [NSSortDescriptor(keyPath: \Result.date, ascending: false)]) var results : FetchedResults<Result>

    @State var singleSelection: UUID?
    @State var deleteResultAttemptAlertEnable : Bool = false
    @State var deleteIndex : Int = 0

    let deleteResultAttemptErrStr : String = String(localized: "Delete Result Attempt", table: "User Error Codes", bundle: Bundle.main, comment: nil)

    var body: some View {
        VStack {
            Text("Previous Test Results")
                .headerTextStyle()
            Text(" \(rawResults.resultStrings.count) Results ")
                .bodyTextStyle()
            VStack(spacing: 10) {
                HStack {
                    Text("Recipient Name")
                        .listTitleTextStyle()
                        .offset(x: -25)
                        .frame(maxWidth: 75)
                    Text("Date")
                        .listTitleTextStyle()
                        .offset(x: -5)
                        .frame(maxWidth: 100)
                    Text("Result")
                        .listTitleTextStyle()
                        .offset(x: 15)
                        .frame(maxWidth: 75)
                }

                List(selection: $singleSelection) {
                    ForEach(rawResults.resultStrings) { result in
                        HStack {
                            Text(result.name)
                                .listTextStyle()
                                .offset(x: 0)
                                .frame(maxWidth: 75)
                            Text(result.date)
                                .listTextStyle()
                                .offset(x: 20)
```

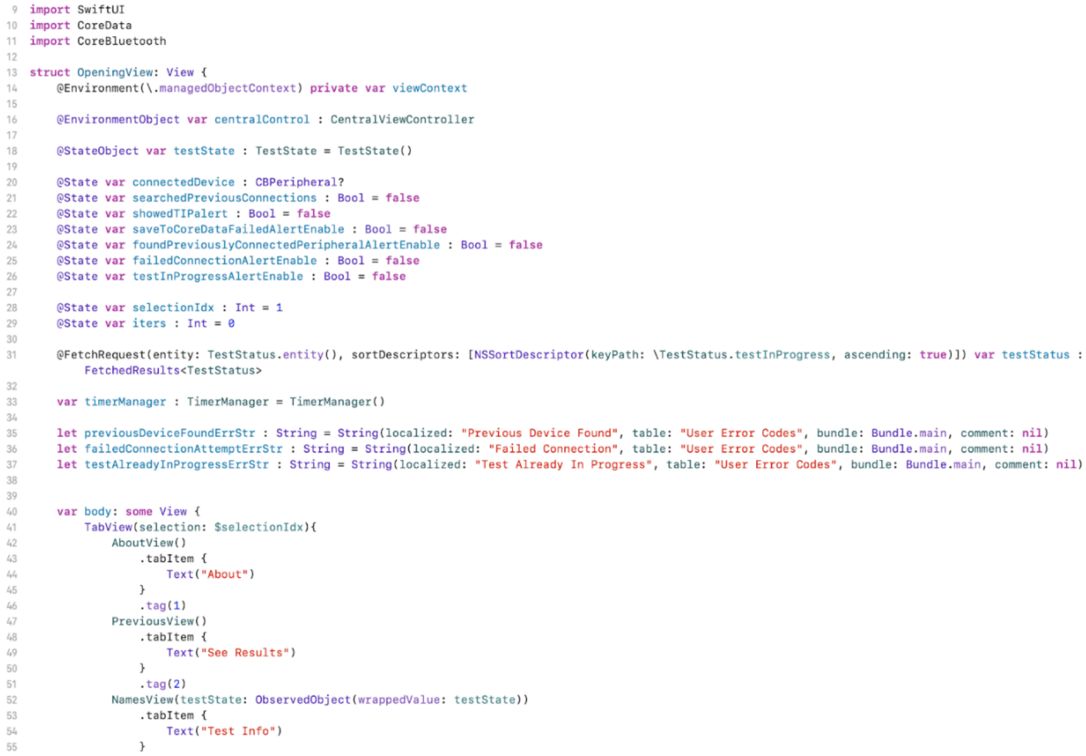


```
                                .frame(maxWidth: 100)
                            Text(result.result)
                                .listTextStyle()
                                .offset(x: 40)
                                .frame(maxWidth: 75)

                        }
                    }
                    .onDelete(perform: findResult)
                    .padding(.all, 0)
                    .headerProminence(.standard)
                }
            }
            .overlay(
                Rectangle()
                    .stroke(.cyan, lineWidth: 2))
            .frame(maxHeight: min(325, 90 + minRowHeight * CGFloat(rawResults.resultStrings.count) * 20))
            Spacer()
        }
        .alert(deleteResultAttemptErrStr, isPresented: $deleteResultAttemptAlertEnable, actions: {
            VStack {
                Button("Yes, delete result") {
                    deleteResult()
                }
                Button("No, do NOT delete result") {
                }}})
        .onAppear {
            rawResults.clearResults()
            for result in results {
                rawResults.addResult(resultString : ResultString(resultRaw: result))
            }
        }
    }

    func findResult(at offsets: IndexSet) {
        for index in offsets {
            deleteIndex = index
        }
        deleteResultAttemptAlertEnable = true
    }

    func deleteResult() {
        while(deleteResultAttemptAlertEnable == true) {
            sleep(1)
        }
        let result = results[deleteIndex]
        viewContext.delete(result)
        do {
```

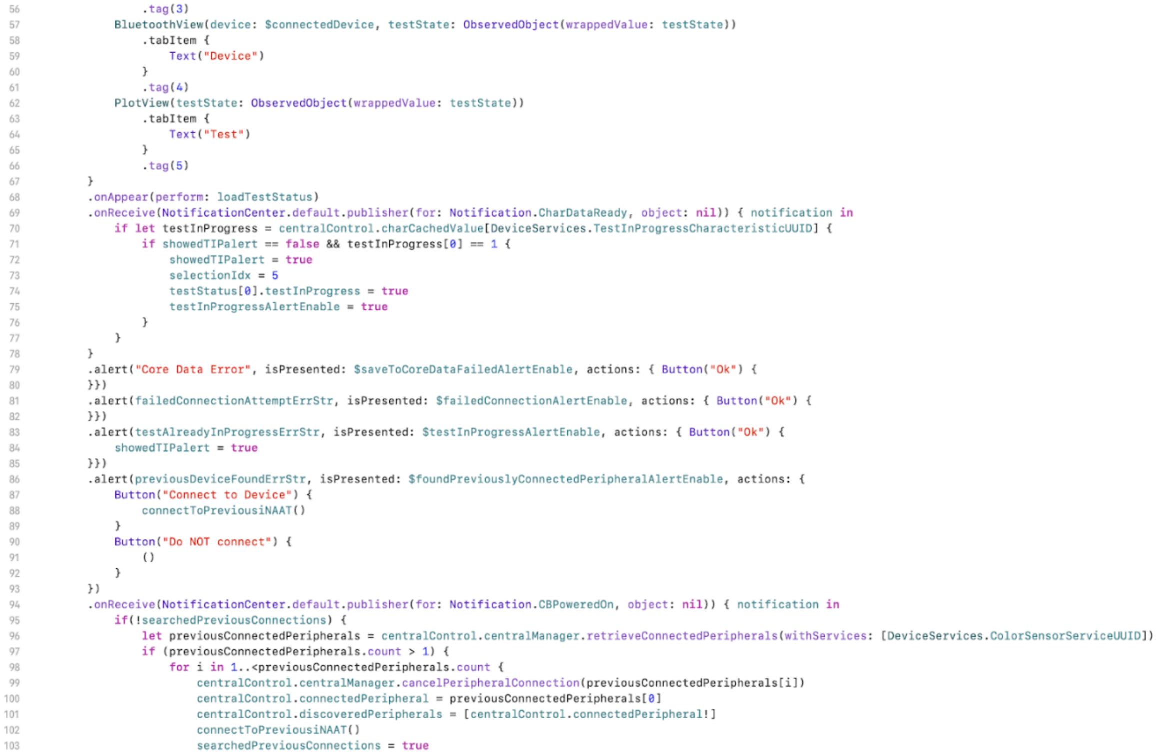


```
    func deleteResult() {
        while(deleteResultAttemptAlertEnable == true) {
            sleep(1)
        }
        let result = results[deleteIndex]
        viewContext.delete(result)
        do {
            try viewContext.save()
        } catch {
            ()
        }
        rawResults.removeResult(at: deleteIndex)
    }
}
```

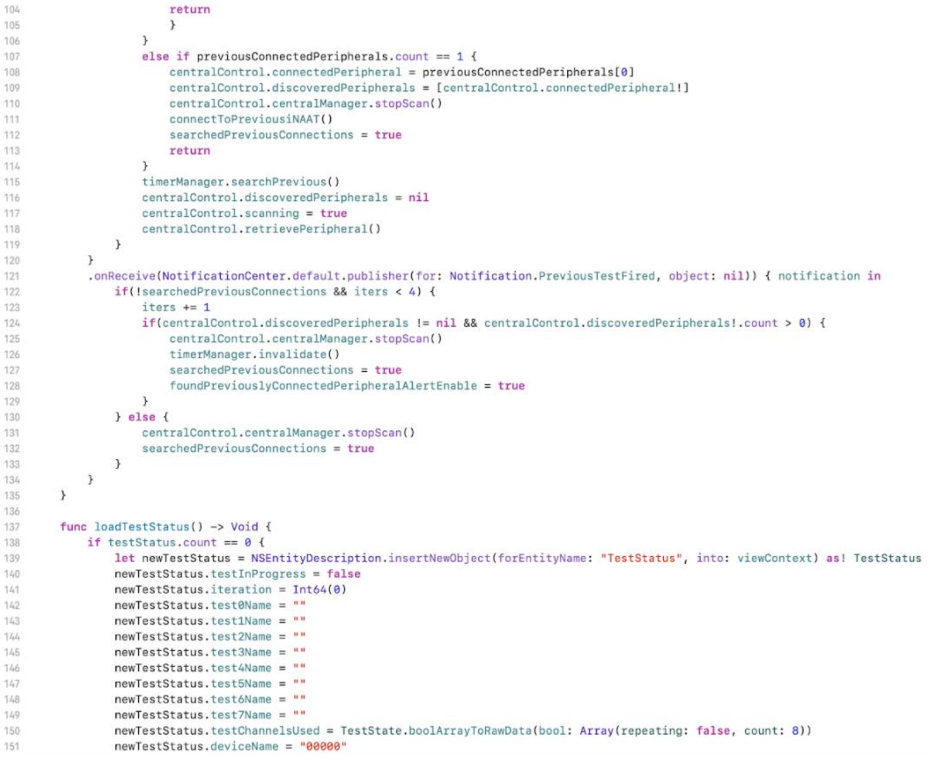
OpeningView.swift



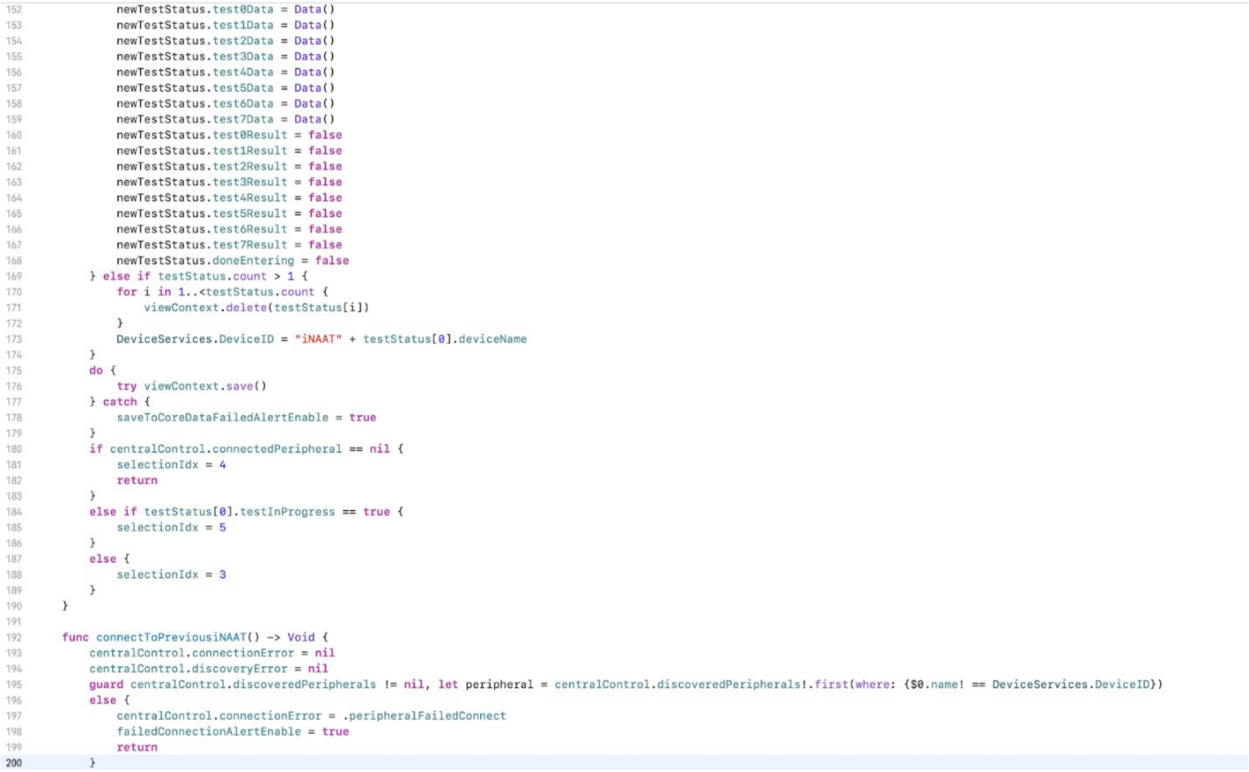
```swift
import SwiftUI
import CoreData
import CoreBluetooth

struct OpeningView: View {
    @Environment(\.managedObjectContext) private var viewContext

    @EnvironmentObject var centralControl : CentralViewController

    @StateObject var testState : TestState = TestState()

    @State var connectedDevice : CBPeripheral?
    @State var searchedPreviousConnections : Bool = false
    @State var showedTIPalert : Bool = false
    @State var saveToCoreDataFailedAlertEnable : Bool = false
    @State var foundPreviouslyConnectedPeripheralAlertEnable : Bool = false
    @State var failedConnectionAlertEnable : Bool = false
    @State var testInProgressAlertEnable : Bool = false

    @State var selectionIdx : Int = 1
    @State var iters : Int = 0

    @FetchRequest(entity: TestStatus.entity(), sortDescriptors: [NSSortDescriptor(keyPath: \TestStatus.testInProgress, ascending: true)]) var testStatus :
        FetchedResults<TestStatus>

    var timerManager : TimerManager = TimerManager()

    let previousDeviceFoundErrStr : String = String(localized: "Previous Device Found", table: "User Error Codes", bundle: Bundle.main, comment: nil)
    let failedConnectionAttemptErrStr : String = String(localized: "Failed Connection", table: "User Error Codes", bundle: Bundle.main, comment: nil)
    let testAlreadyInProgressErrStr : String = String(localized: "Test Already In Progress", table: "User Error Codes", bundle: Bundle.main, comment: nil)


    var body: some View {
        TabView(selection: $selectionIdx){
            AboutView()
                .tabItem {
                    Text("About")
                }
                .tag(1)
            PreviousView()
                .tabItem {
                    Text("See Results")
                }
                .tag(2)
            NamesView(testState: ObservedObject(wrappedValue: testState))
                .tabItem {
                    Text("Test Info")
                }
```



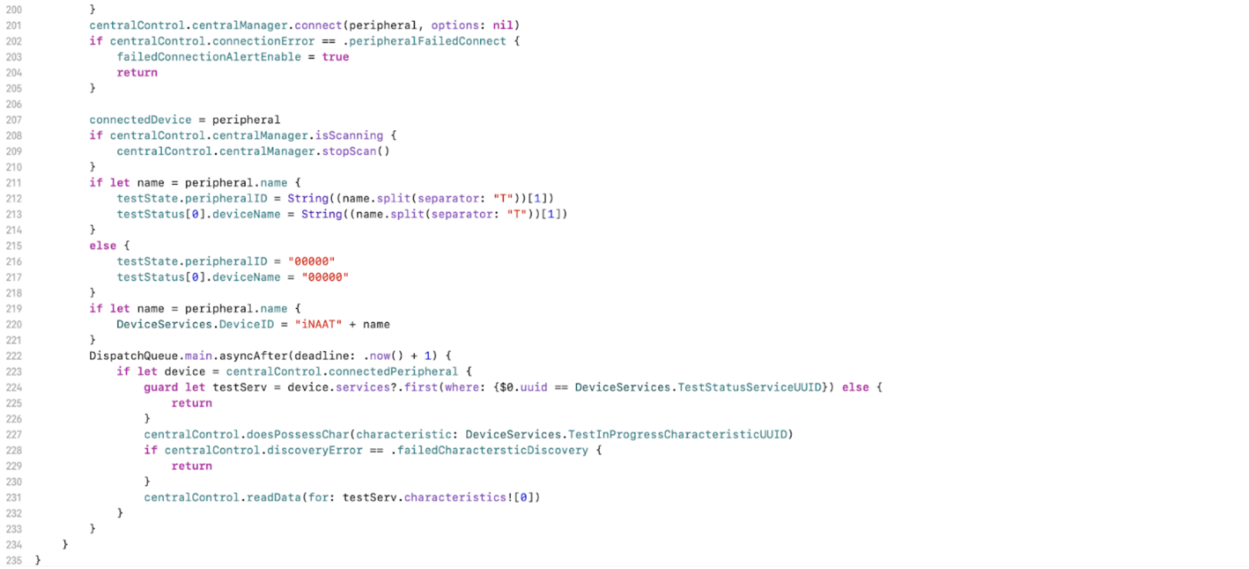
```
                .tag(3)
            BluetoothView(device: $connectedDevice, testState: ObservedObject(wrappedValue: testState))
                .tabItem {
                    Text("Device")
                }
                .tag(4)
            PlotView(testState: ObservedObject(wrappedValue: testState))
                .tabItem {
                    Text("Test")
                }
                .tag(5)
        }
        .onAppear(perform: loadTestStatus)
        .onReceive(NotificationCenter.default.publisher(for: Notification.CharDataReady, object: nil)) { notification in
            if let testInProgress = centralControl.charCachedValue[DeviceServices.TestInProgressCharacteristicUUID] {
                if showedTIPalert == false && testInProgress[0] == 1 {
                    showedTIPalert = true
                    selectionIdx = 5
                    testStatus[0].testInProgress = true
                    testInProgressAlertEnable = true
                }
            }
        }
        .alert("Core Data Error", isPresented: $saveToCoreDataFailedAlertEnable, actions: { Button("Ok") {
        }})
        .alert(failedConnectionAttemptErrStr, isPresented: $failedConnectionAlertEnable, actions: { Button("Ok") {
        }})
        .alert(testAlreadyInProgressErrStr, isPresented: $testInProgressAlertEnable, actions: { Button("Ok") {
            showedTIPalert = true
        }})
        .alert(previousDeviceFoundErrStr, isPresented: $foundPreviouslyConnectedPeripheralAlertEnable, actions: {
            Button("Connect to Device") {
                connectToPreviousiNAAT()
            }
            Button("Do NOT connect") {
                ()
            }
        })
        .onReceive(NotificationCenter.default.publisher(for: Notification.CBPoweredOn, object: nil)) { notification in
            if(!searchedPreviousConnections) {
                let previousConnectedPeripherals = centralControl.centralManager.retrieveConnectedPeripherals(withServices: [DeviceServices.ColorSensorServiceUUID])
                if (previousConnectedPeripherals.count > 1) {
                    for i in 1..<previousConnectedPeripherals.count {
                        centralControl.centralManager.cancelPeripheralConnection(previousConnectedPeripherals[i])
                        centralControl.connectedPeripheral = previousConnectedPeripherals[0]
                        centralControl.discoveredPeripherals = [centralControl.connectedPeripheral!]
                        connectToPreviousiNAAT()
                        searchedPreviousConnections = true
```



```
                    return
                    }
                }
                else if previousConnectedPeripherals.count == 1 {
                    centralControl.connectedPeripheral = previousConnectedPeripherals[0]
                    centralControl.discoveredPeripherals = [centralControl.connectedPeripheral!]
                    centralControl.centralManager.stopScan()
                    connectToPreviousiNAAT()
                    searchedPreviousConnections = true
                    return
                }
                timerManager.searchPrevious()
                centralControl.discoveredPeripherals = nil
                centralControl.scanning = true
                centralControl.retrievePeripheral()
            }
        }
        .onReceive(NotificationCenter.default.publisher(for: Notification.PreviousTestFired, object: nil)) { notification in
            if(!searchedPreviousConnections && iters < 4) {
                iters += 1
                if(centralControl.discoveredPeripherals != nil && centralControl.discoveredPeripherals!.count > 0) {
                    centralControl.centralManager.stopScan()
                    timerManager.invalidate()
                    searchedPreviousConnections = true
                    foundPreviouslyConnectedPeripheralAlertEnable = true
                }
            } else {
                centralControl.centralManager.stopScan()
                searchedPreviousConnections = true
            }
        }
    }

    func loadTestStatus() -> Void {
        if testStatus.count == 0 {
            let newTestStatus = NSEntityDescription.insertNewObject(forEntityName: "TestStatus", into: viewContext) as! TestStatus
            newTestStatus.testInProgress = false
            newTestStatus.iteration = Int64(0)
            newTestStatus.test0Name = ""
            newTestStatus.test1Name = ""
            newTestStatus.test2Name = ""
            newTestStatus.test3Name = ""
            newTestStatus.test4Name = ""
            newTestStatus.test5Name = ""
            newTestStatus.test6Name = ""
            newTestStatus.test7Name = ""
            newTestStatus.testChannelsUsed = TestState.boolArrayToRawData(bool: Array(repeating: false, count: 8))
            newTestStatus.deviceName = "00000"
```

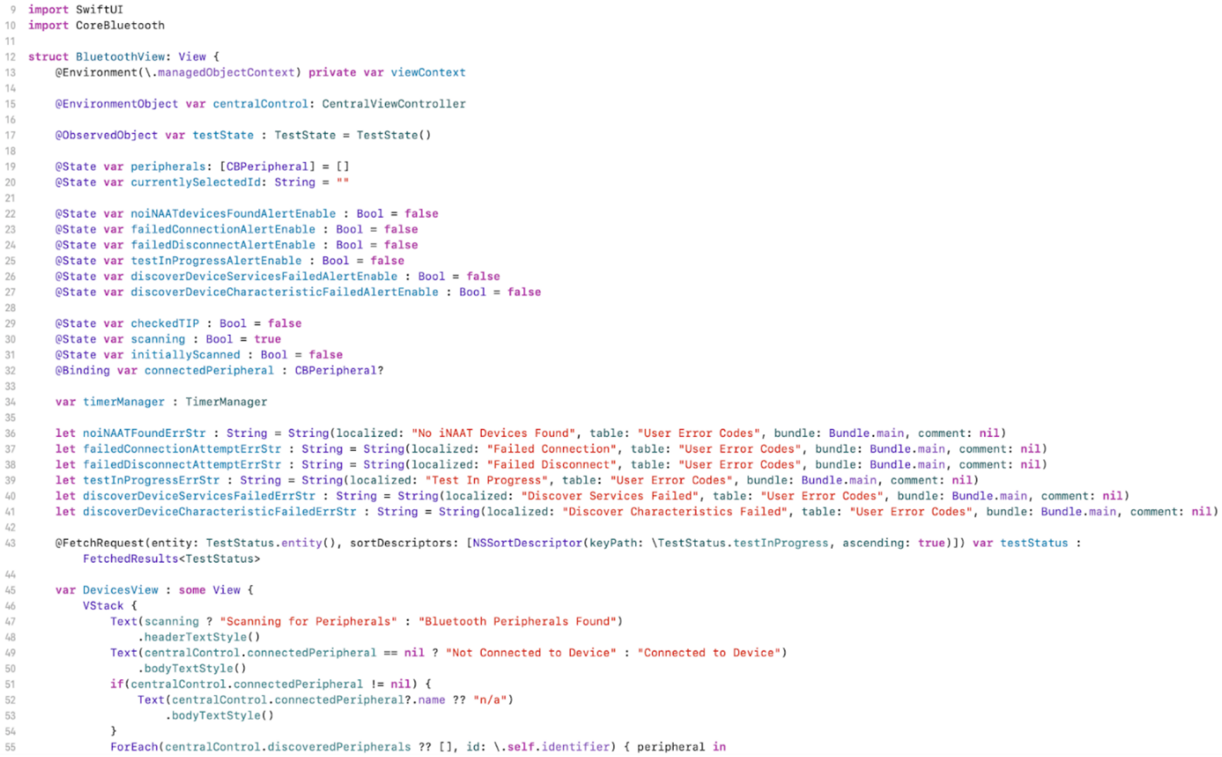


```swift
            newTestStatus.test0Data = Data()
            newTestStatus.test1Data = Data()
            newTestStatus.test2Data = Data()
            newTestStatus.test3Data = Data()
            newTestStatus.test4Data = Data()
            newTestStatus.test5Data = Data()
            newTestStatus.test6Data = Data()
            newTestStatus.test7Data = Data()
            newTestStatus.test0Result = false
            newTestStatus.test1Result = false
            newTestStatus.test2Result = false
            newTestStatus.test3Result = false
            newTestStatus.test4Result = false
            newTestStatus.test5Result = false
            newTestStatus.test6Result = false
            newTestStatus.test7Result = false
            newTestStatus.doneEntering = false
        } else if testStatus.count > 1 {
            for i in 1..<testStatus.count {
                viewContext.delete(testStatus[i])
            }
            DeviceServices.DeviceID = "iNAAT" + testStatus[0].deviceName
        }
        do {
            try viewContext.save()
        } catch {
            saveToCoreDataFailedAlertEnable = true
        }
        if centralControl.connectedPeripheral == nil {
            selectionIdx = 4
            return
        }
        else if testStatus[0].testInProgress == true {
            selectionIdx = 5
        }
        else {
            selectionIdx = 3
        }
    }

    func connectToPreviousiNAAT() -> Void {
        centralControl.connectionError = nil
        centralControl.discoveryError = nil
        guard centralControl.discoveredPeripherals != nil, let peripheral = centralControl.discoveredPeripherals!.first(where: {$0.name! == DeviceServices.DeviceID})
        else {
            centralControl.connectionError = .peripheralFailedConnect
            failedConnectionAlertEnable = true
            return
        }
```

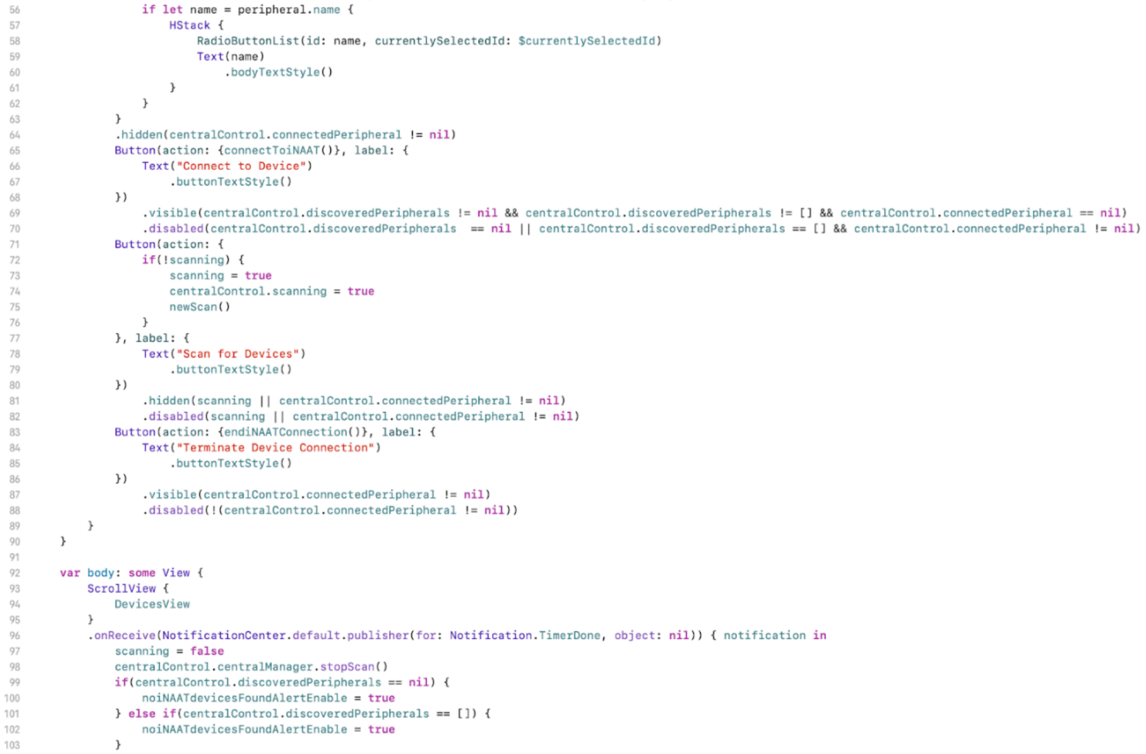


```swift
        }
        centralControl.centralManager.connect(peripheral, options: nil)
        if centralControl.connectionError == .peripheralFailedConnect {
            failedConnectionAlertEnable = true
            return
        }

        connectedDevice = peripheral
        if centralControl.centralManager.isScanning {
            centralControl.centralManager.stopScan()
        }
        if let name = peripheral.name {
            testState.peripheralID = String((name.split(separator: "T"))[1])
            testStatus[0].deviceName = String((name.split(separator: "T"))[1])
        }
        else {
            testState.peripheralID = "00000"
            testStatus[0].deviceName = "00000"
        }
        if let name = peripheral.name {
            DeviceServices.DeviceID = "iNAAT" + name
        }
        DispatchQueue.main.asyncAfter(deadline: .now() + 1) {
            if let device = centralControl.connectedPeripheral {
                guard let testServ = device.services?.first(where: {$0.uuid == DeviceServices.TestStatusServiceUUID}) else {
                    return
                }
                centralControl.doesPossessChar(characteristic: DeviceServices.TestInProgressCharacteristicUUID)
                if centralControl.discoveryError == .failedCharactersticDiscovery {
                    return
                }
                centralControl.readData(for: testServ.characteristics![0])
            }
        }
    }
}
```

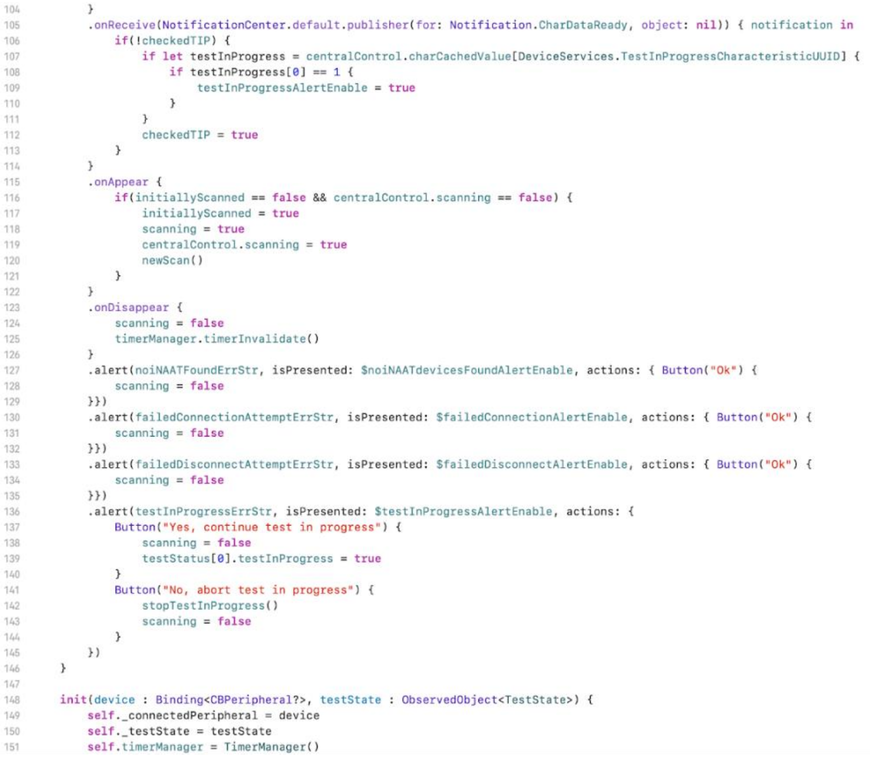
BluetoothView.swift



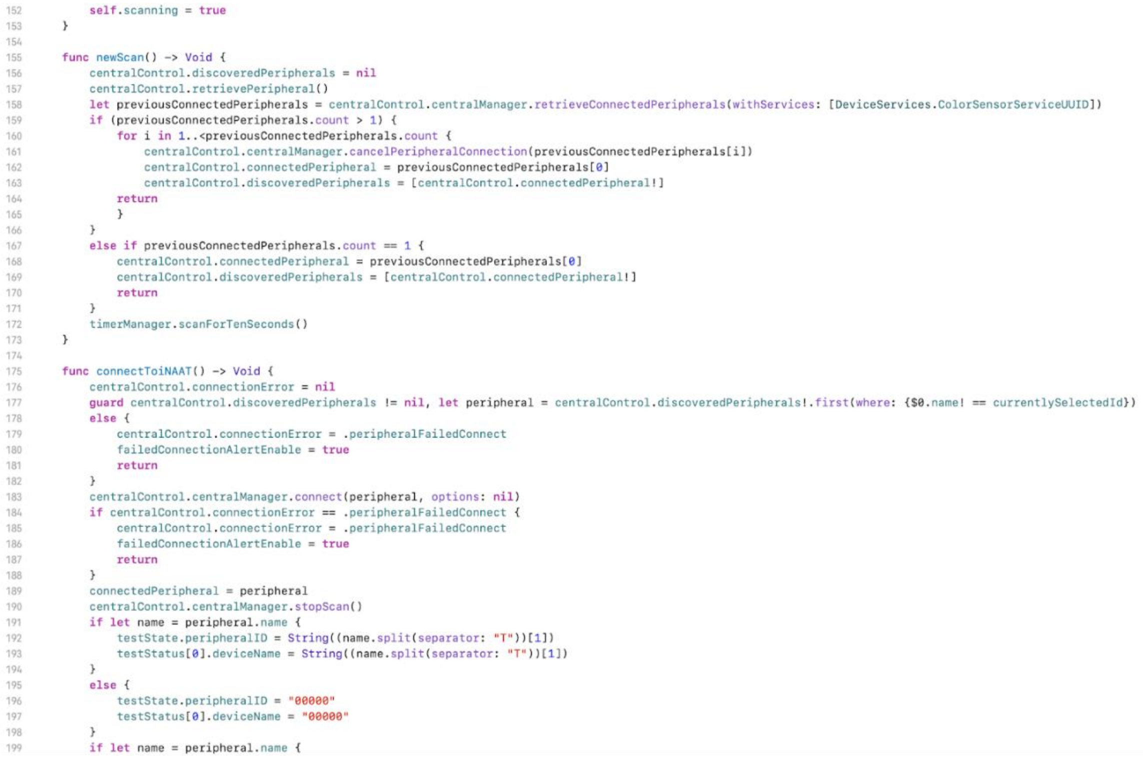
```
import SwiftUI
import CoreBluetooth

struct BluetoothView: View {
    @Environment(\.managedObjectContext) private var viewContext

    @EnvironmentObject var centralControl: CentralViewController

    @ObservedObject var testState : TestState = TestState()

    @State var peripherals: [CBPeripheral] = []
    @State var currentlySelectedId: String = ""

    @State var noiNAATdevicesFoundAlertEnable : Bool = false
    @State var failedConnectionAlertEnable : Bool = false
    @State var failedDisconnectAlertEnable : Bool = false
    @State var testInProgressAlertEnable : Bool = false
    @State var discoverDeviceServicesFailedAlertEnable : Bool = false
    @State var discoverDeviceCharacteristicFailedAlertEnable : Bool = false

    @State var checkedTIP : Bool = false
    @State var scanning : Bool = true
    @State var initiallyScanned : Bool = false
    @Binding var connectedPeripheral : CBPeripheral?

    var timerManager : TimerManager

    let noiNAATFoundErrStr : String = String(localized: "No iNAAT Devices Found", table: "User Error Codes", bundle: Bundle.main, comment: nil)
    let failedConnectionAttemptErrStr : String = String(localized: "Failed Connection", table: "User Error Codes", bundle: Bundle.main, comment: nil)
    let failedDisconnectAttemptErrStr : String = String(localized: "Failed Disconnect", table: "User Error Codes", bundle: Bundle.main, comment: nil)
    let testInProgressErrStr : String = String(localized: "Test In Progress", table: "User Error Codes", bundle: Bundle.main, comment: nil)
    let discoverDeviceServicesFailedErrStr : String = String(localized: "Discover Services Failed", table: "User Error Codes", bundle: Bundle.main, comment: nil)
    let discoverDeviceCharacteristicFailedErrStr : String = String(localized: "Discover Characteristics Failed", table: "User Error Codes", bundle: Bundle.main, comment: nil)

    @FetchRequest(entity: TestStatus.entity(), sortDescriptors: [NSSortDescriptor(keyPath: \TestStatus.testInProgress, ascending: true)]) var testStatus :
        FetchedResults<TestStatus>

    var DevicesView : some View {
        VStack {
            Text(scanning ? "Scanning for Peripherals" : "Bluetooth Peripherals Found")
                .headerTextStyle()
            Text(centralControl.connectedPeripheral == nil ? "Not Connected to Device" : "Connected to Device")
                .bodyTextStyle()
            if(centralControl.connectedPeripheral != nil) {
                Text(centralControl.connectedPeripheral?.name ?? "n/a")
                    .bodyTextStyle()
            }
            ForEach(centralControl.discoveredPeripherals ?? [], id: \.self.identifier) { peripheral in
```



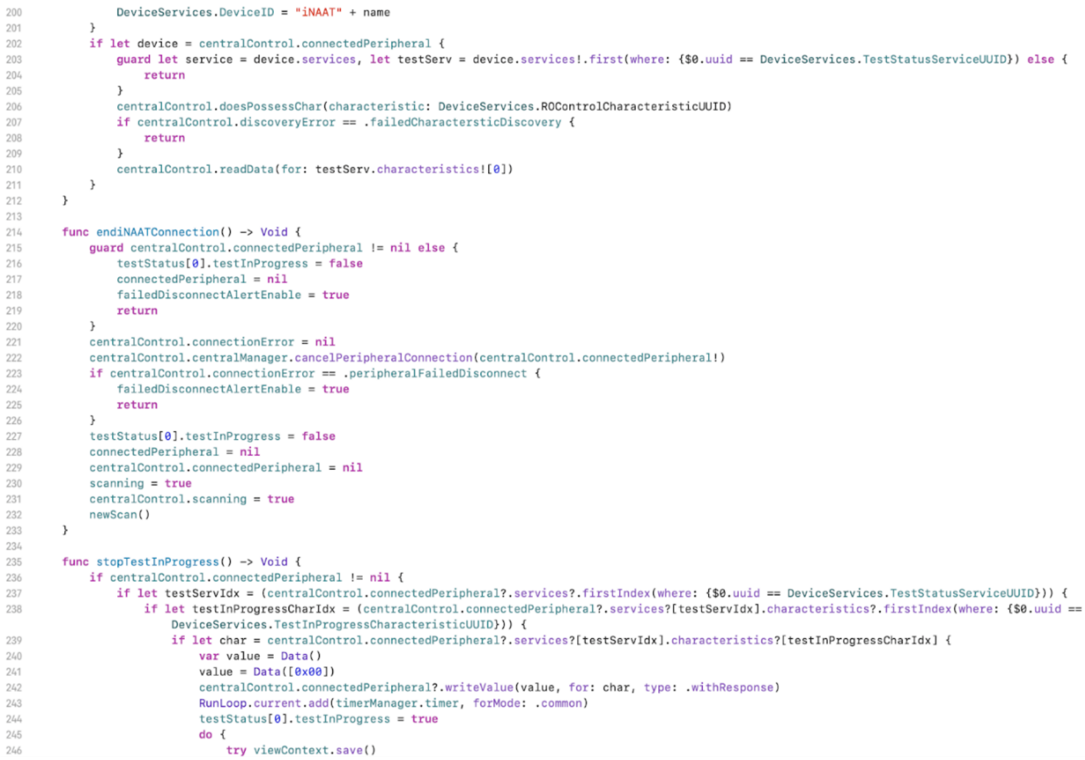
```
                if let name = peripheral.name {
                    HStack {
                        RadioButtonList(id: name, currentlySelectedId: $currentlySelectedId)
                        Text(name)
                            .bodyTextStyle()
                    }
                }
            }
            .hidden(centralControl.connectedPeripheral != nil)
            Button(action: {connectToiNAAT()}, label: {
                Text("Connect to Device")
                    .buttonTextStyle()
            })
                .visible(centralControl.discoveredPeripherals != nil && centralControl.discoveredPeripherals != [] && centralControl.connectedPeripheral == nil)
                .disabled(centralControl.discoveredPeripherals  == nil || centralControl.discoveredPeripherals == [] && centralControl.connectedPeripheral != nil)
            Button(action: {
                if(!scanning) {
                    scanning = true
                    centralControl.scanning = true
                    newScan()
                }
            }, label: {
                Text("Scan for Devices")
                    .buttonTextStyle()
            })
                .hidden(scanning || centralControl.connectedPeripheral != nil)
                .disabled(scanning || centralControl.connectedPeripheral != nil)
            Button(action: {endiNAATConnection()}, label: {
                Text("Terminate Device Connection")
                    .buttonTextStyle()
            })
                .visible(centralControl.connectedPeripheral != nil)
                .disabled(!(centralControl.connectedPeripheral != nil))
        }
    }

    var body: some View {
        ScrollView {
            DevicesView
        }
        .onReceive(NotificationCenter.default.publisher(for: Notification.TimerDone, object: nil)) { notification in
            scanning = false
            centralControl.centralManager.stopScan()
            if(centralControl.discoveredPeripherals == nil) {
                noiNAATdevicesFoundAlertEnable = true
            } else if(centralControl.discoveredPeripherals == []) {
                noiNAATdevicesFoundAlertEnable = true
            }
```



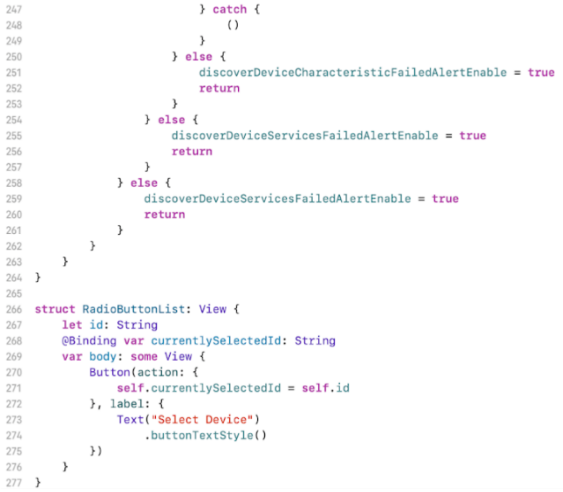
```
        }
        .onReceive(NotificationCenter.default.publisher(for: Notification.CharDataReady, object: nil)) { notification in
            if(!checkedTIP) {
                if let testInProgress = centralControl.charCachedValue[DeviceServices.TestInProgressCharacteristicUUID] {
                    if testInProgress[0] == 1 {
                        testInProgressAlertEnable = true
                    }
                }
                checkedTIP = true
            }
        }
        .onAppear {
            if(initiallyScanned == false && centralControl.scanning == false) {
                initiallyScanned = true
                scanning = true
                centralControl.scanning = true
                newScan()
            }
        }
        .onDisappear {
            scanning = false
            timerManager.timerInvalidate()
        }
        .alert(noiNAATFoundErrStr, isPresented: $noiNAATdevicesFoundAlertEnable, actions: { Button("Ok") {
            scanning = false
        }})
        .alert(failedConnectionAttemptErrStr, isPresented: $failedConnectionAlertEnable, actions: { Button("Ok") {
            scanning = false
        }})
        .alert(failedDisconnectAttemptErrStr, isPresented: $failedDisconnectAlertEnable, actions: { Button("Ok") {
            scanning = false
        }})
        .alert(testInProgressErrStr, isPresented: $testInProgressAlertEnable, actions: {
            Button("Yes, continue test in progress") {
                scanning = false
                testStatus[0].testInProgress = true
            }
            Button("No, abort test in progress") {
                stopTestInProgress()
                scanning = false
            }
        })
    }

    init(device : Binding<CBPeripheral?>, testState : ObservedObject<TestState>) {
        self._connectedPeripheral = device
        self._testState = testState
        self.timerManager = TimerManager()
```



```
        self.scanning = true
    }

    func newScan() -> Void {
        centralControl.discoveredPeripherals = nil
        centralControl.retrievePeripheral()
        let previousConnectedPeripherals = centralControl.centralManager.retrieveConnectedPeripherals(withServices: [DeviceServices.ColorSensorServiceUUID])
        if (previousConnectedPeripherals.count > 1) {
            for i in 1..<previousConnectedPeripherals.count {
                centralControl.centralManager.cancelPeripheralConnection(previousConnectedPeripherals[i])
                centralControl.connectedPeripheral = previousConnectedPeripherals[0]
                centralControl.discoveredPeripherals = [centralControl.connectedPeripheral!]
            return
            }
        }
        else if previousConnectedPeripherals.count == 1 {
            centralControl.connectedPeripheral = previousConnectedPeripherals[0]
            centralControl.discoveredPeripherals = [centralControl.connectedPeripheral!]
            return
        }
        timerManager.scanForTenSeconds()
    }

    func connectToiNAAT() -> Void {
        centralControl.connectionError = nil
        guard centralControl.discoveredPeripherals != nil, let peripheral = centralControl.discoveredPeripherals!.first(where: {$0.name! == currentlySelectedId})
        else {
            centralControl.connectionError = .peripheralFailedConnect
            failedConnectionAlertEnable = true
            return
        }
        centralControl.centralManager.connect(peripheral, options: nil)
        if centralControl.connectionError == .peripheralFailedConnect {
            centralControl.connectionError = .peripheralFailedConnect
            failedConnectionAlertEnable = true
            return
        }
        connectedPeripheral = peripheral
        centralControl.centralManager.stopScan()
        if let name = peripheral.name {
            testState.peripheralID = String((name.split(separator: "T"))[1])
            testStatus[0].deviceName = String((name.split(separator: "T"))[1])
        }
        else {
            testState.peripheralID = "00000"
            testStatus[0].deviceName = "00000"
        }
        if let name = peripheral.name {
```

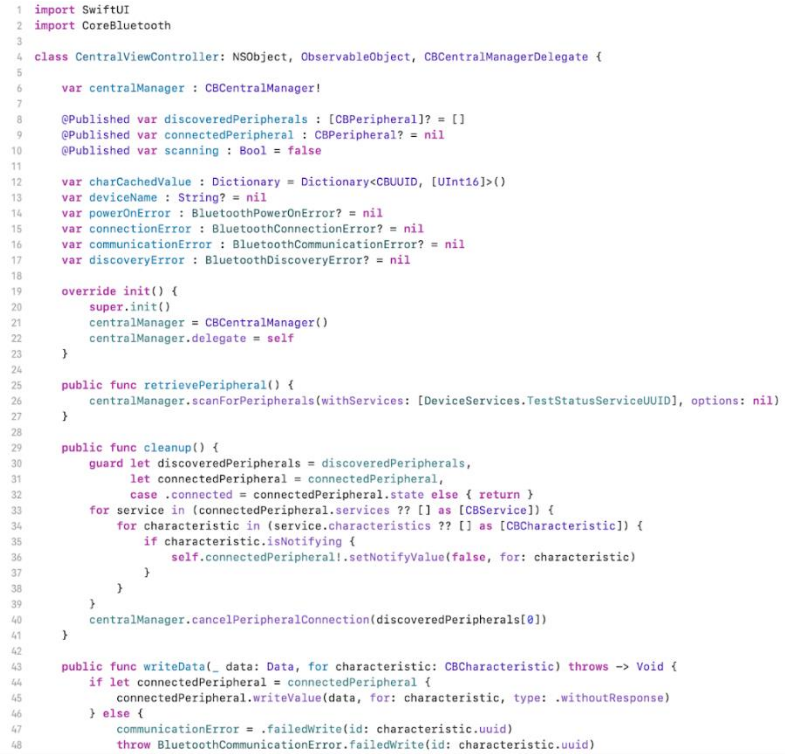


```
            DeviceServices.DeviceID = "iNAAT" + name
        }
        if let device = centralControl.connectedPeripheral {
            guard let service = device.services, let testServ = device.services!.first(where: {$0.uuid == DeviceServices.TestStatusServiceUUID}) else {
                return
            }
            centralControl.doesPossessChar(characteristic: DeviceServices.ROControlCharacteristicUUID)
            if centralControl.discoveryError == .failedCharactersticDiscovery {
                return
            }
            centralControl.readData(for: testServ.characteristics![0])
        }
    }

    func endiNAATConnection() -> Void {
        guard centralControl.connectedPeripheral != nil else {
            testStatus[0].testInProgress = false
            connectedPeripheral = nil
            failedDisconnectAlertEnable = true
            return
        }
        centralControl.connectionError = nil
        centralControl.centralManager.cancelPeripheralConnection(centralControl.connectedPeripheral!)
        if centralControl.connectionError == .peripheralFailedDisconnect {
            failedDisconnectAlertEnable = true
            return
        }
        testStatus[0].testInProgress = false
        connectedPeripheral = nil
        centralControl.connectedPeripheral = nil
        scanning = true
        centralControl.scanning = true
        newScan()
    }

    func stopTestInProgress() -> Void {
        if centralControl.connectedPeripheral != nil {
            if let testServIdx = (centralControl.connectedPeripheral?.services?.firstIndex(where: {$0.uuid == DeviceServices.TestStatusServiceUUID})) {
                if let testInProgressCharIdx = (centralControl.connectedPeripheral?.services?[testServIdx].characteristics?.firstIndex(where: {$0.uuid ==
                    DeviceServices.TestInProgressCharacteristicUUID})) {
                    if let char = centralControl.connectedPeripheral?.services?[testServIdx].characteristics?[testInProgressCharIdx] {
                        var value = Data()
                        value = Data([0x00])
                        centralControl.connectedPeripheral?.writeValue(value, for: char, type: .withResponse)
                        RunLoop.current.add(timerManager.timer, forMode: .common)
                        testStatus[0].testInProgress = true
                        do {
                            try viewContext.save()
```

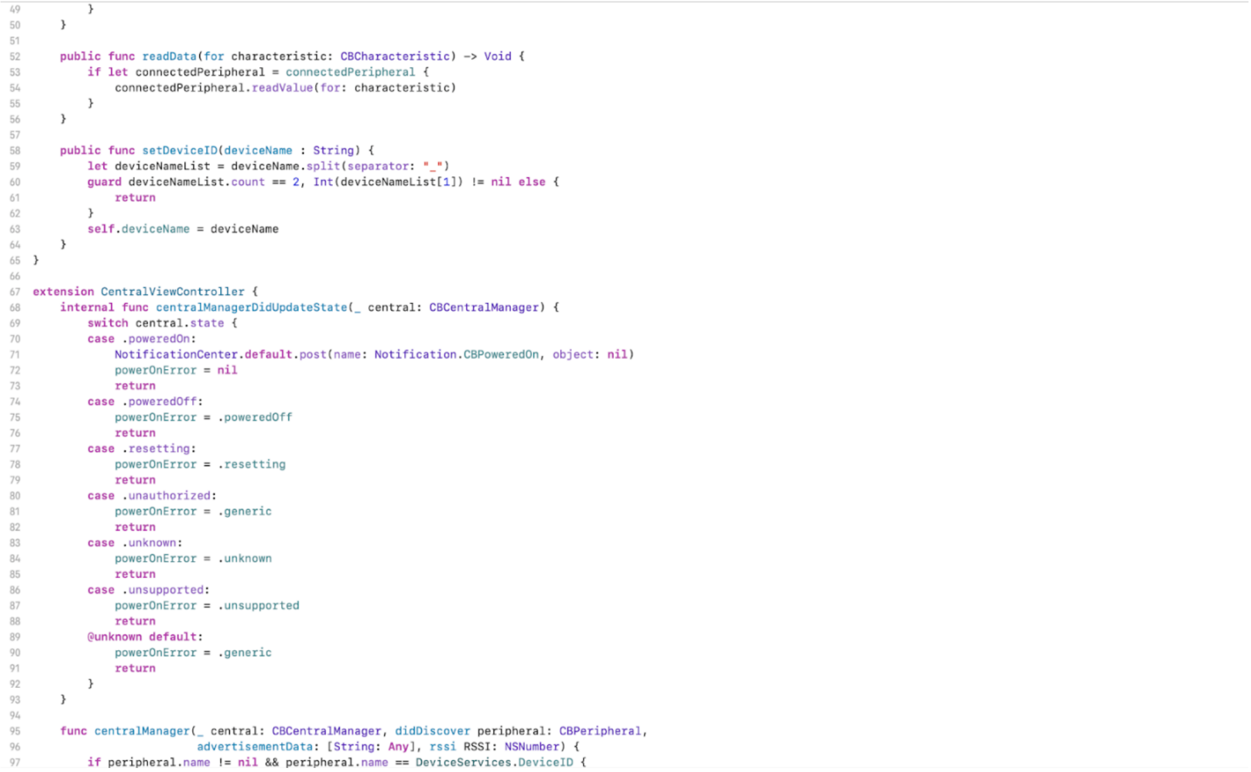


```swift
                        } catch {
                            ()
                        }
                    } else {
                        discoverDeviceCharacteristicFailedAlertEnable = true
                        return
                    }
                } else {
                    discoverDeviceServicesFailedAlertEnable = true
                    return
                }
            } else {
                discoverDeviceServicesFailedAlertEnable = true
                return
            }
        }
    }
}

struct RadioButtonList: View {
    let id: String
    @Binding var currentlySelectedId: String
    var body: some View {
        Button(action: {
            self.currentlySelectedId = self.id
        }, label: {
            Text("Select Device")
                .buttonTextStyle()
        })
    }
}
```

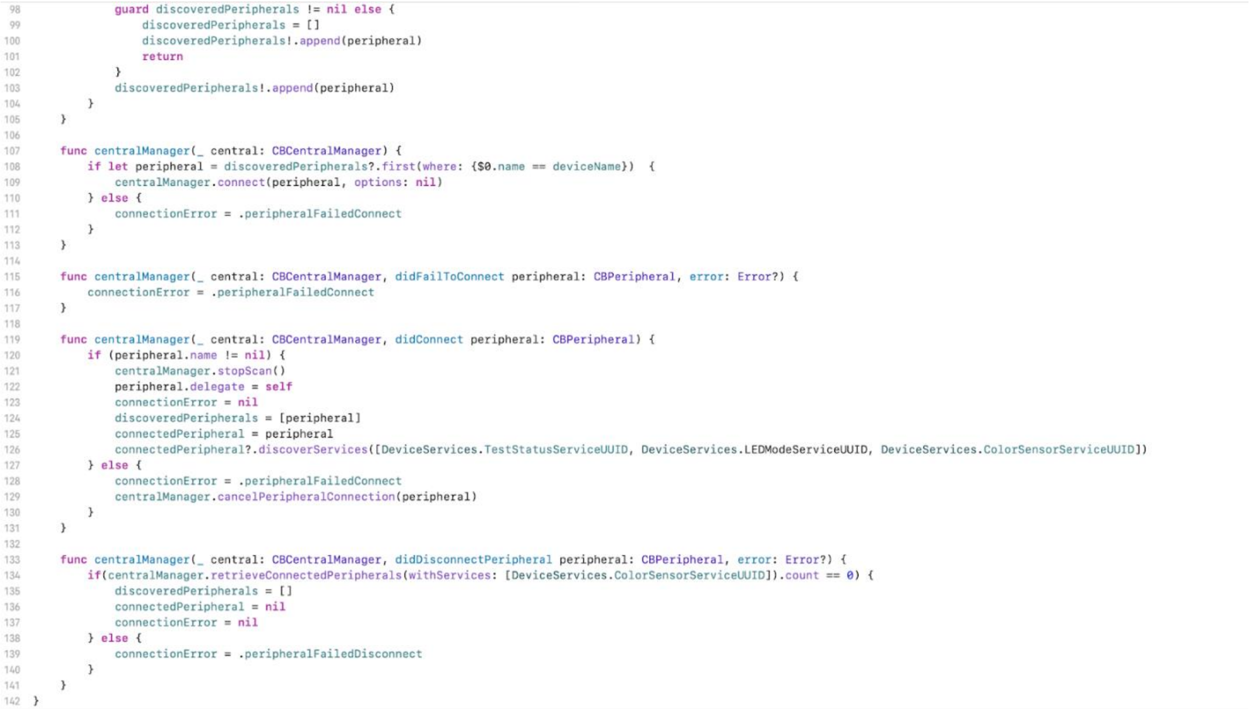
CentralViewController.swift



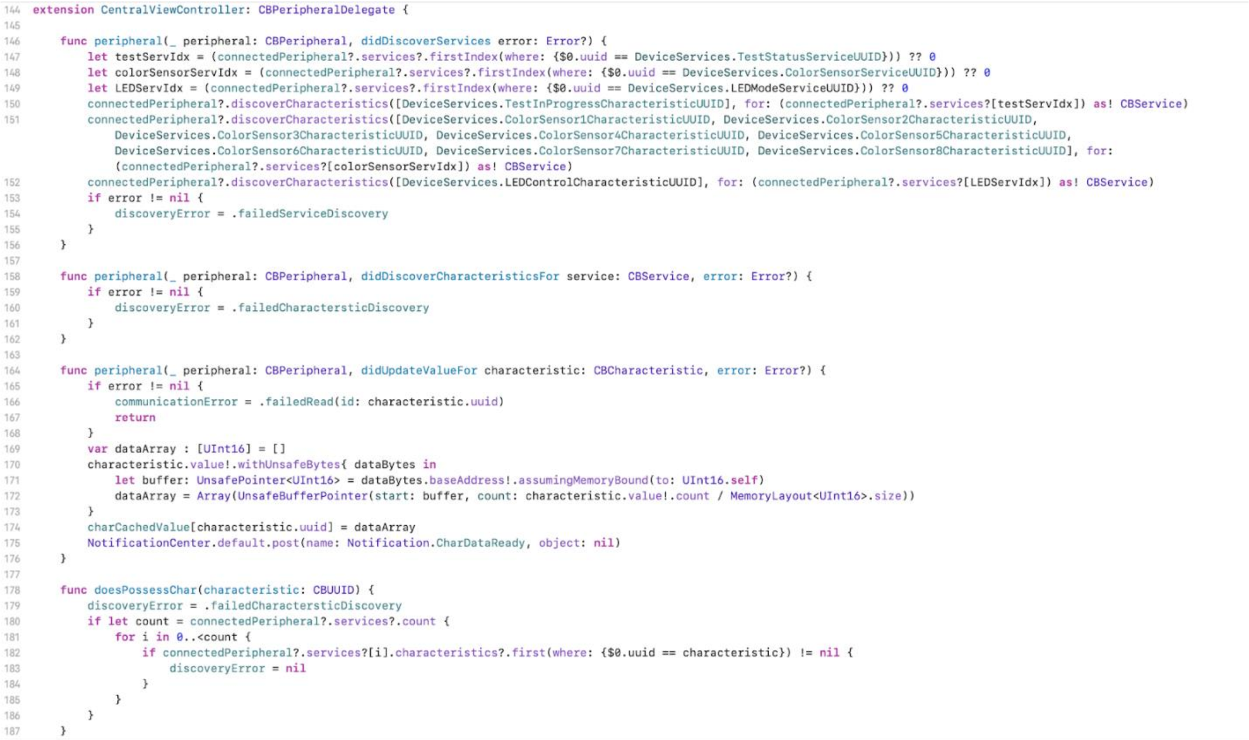
```swift
import SwiftUI
import CoreBluetooth

class CentralViewController: NSObject, ObservableObject, CBCentralManagerDelegate {

    var centralManager : CBCentralManager!

    @Published var discoveredPeripherals : [CBPeripheral]? = []
    @Published var connectedPeripheral : CBPeripheral? = nil
    @Published var scanning : Bool = false

    var charCachedValue : Dictionary = Dictionary<CBUUID, [UInt16]>()
    var deviceName : String? = nil
    var powerOnError : BluetoothPowerOnError? = nil
    var connectionError : BluetoothConnectionError? = nil
    var communicationError : BluetoothCommunicationError? = nil
    var discoveryError : BluetoothDiscoveryError? = nil

    override init() {
        super.init()
        centralManager = CBCentralManager()
        centralManager.delegate = self
    }

    public func retrievePeripheral() {
        centralManager.scanForPeripherals(withServices: [DeviceServices.TestStatusServiceUUID], options: nil)
    }

    public func cleanup() {
        guard let discoveredPeripherals = discoveredPeripherals,
              let connectedPeripheral = connectedPeripheral,
              case .connected = connectedPeripheral.state else { return }
        for service in (connectedPeripheral.services ?? [] as [CBService]) {
            for characteristic in (service.characteristics ?? [] as [CBCharacteristic]) {
                if characteristic.isNotifying {
                    self.connectedPeripheral!.setNotifyValue(false, for: characteristic)
                }
            }
        }
        centralManager.cancelPeripheralConnection(discoveredPeripherals[0])
    }

    public func writeData(_ data: Data, for characteristic: CBCharacteristic) throws -> Void {
        if let connectedPeripheral = connectedPeripheral {
            connectedPeripheral.writeValue(data, for: characteristic, type: .withoutResponse)
        } else {
            communicationError = .failedWrite(id: characteristic.uuid)
            throw BluetoothCommunicationError.failedWrite(id: characteristic.uuid)
```



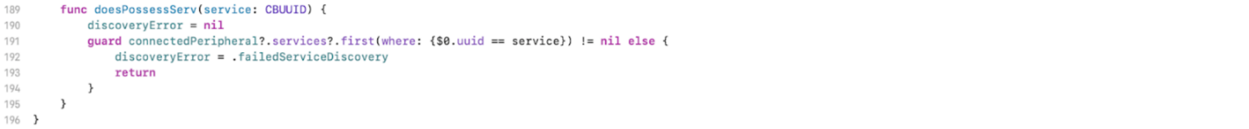
```
        }
    }

    public func readData(for characteristic: CBCharacteristic) -> Void {
        if let connectedPeripheral = connectedPeripheral {
            connectedPeripheral.readValue(for: characteristic)
        }
    }

    public func setDeviceID(deviceName : String) {
        let deviceNameList = deviceName.split(separator: "_")
        guard deviceNameList.count == 2, Int(deviceNameList[1]) != nil else {
            return
        }
        self.deviceName = deviceName
    }
}

extension CentralViewController {
    internal func centralManagerDidUpdateState(_ central: CBCentralManager) {
        switch central.state {
        case .poweredOn:
            NotificationCenter.default.post(name: Notification.CBPoweredOn, object: nil)
            powerOnError = nil
            return
        case .poweredOff:
            powerOnError = .poweredOff
            return
        case .resetting:
            powerOnError = .resetting
            return
        case .unauthorized:
            powerOnError = .generic
            return
        case .unknown:
            powerOnError = .unknown
            return
        case .unsupported:
            powerOnError = .unsupported
            return
        @unknown default:
            powerOnError = .generic
            return
        }
    }

    func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral,
                        advertisementData: [String: Any], rssi RSSI: NSNumber) {
        if peripheral.name != nil && peripheral.name == DeviceServices.DeviceID {
```



```
            guard discoveredPeripherals != nil else {
                discoveredPeripherals = []
                discoveredPeripherals!.append(peripheral)
                return
            }
            discoveredPeripherals!.append(peripheral)
        }
    }

    func centralManager(_ central: CBCentralManager) {
        if let peripheral = discoveredPeripherals?.first(where: {$0.name == deviceName})  {
            centralManager.connect(peripheral, options: nil)
        } else {
            connectionError = .peripheralFailedConnect
        }
    }

    func centralManager(_ central: CBCentralManager, didFailToConnect peripheral: CBPeripheral, error: Error?) {
        connectionError = .peripheralFailedConnect
    }

    func centralManager(_ central: CBCentralManager, didConnect peripheral: CBPeripheral) {
        if (peripheral.name != nil) {
            centralManager.stopScan()
            peripheral.delegate = self
            connectionError = nil
            discoveredPeripherals = [peripheral]
            connectedPeripheral = peripheral
            connectedPeripheral?.discoverServices([DeviceServices.TestStatusServiceUUID, DeviceServices.LEDModeServiceUUID, DeviceServices.ColorSensorServiceUUID])
        } else {
            connectionError = .peripheralFailedConnect
            centralManager.cancelPeripheralConnection(peripheral)
        }
    }

    func centralManager(_ central: CBCentralManager, didDisconnectPeripheral peripheral: CBPeripheral, error: Error?) {
        if(centralManager.retrieveConnectedPeripherals(withServices: [DeviceServices.ColorSensorServiceUUID]).count == 0) {
            discoveredPeripherals = []
            connectedPeripheral = nil
            connectionError = nil
        } else {
            connectionError = .peripheralFailedDisconnect
        }
    }
}
```

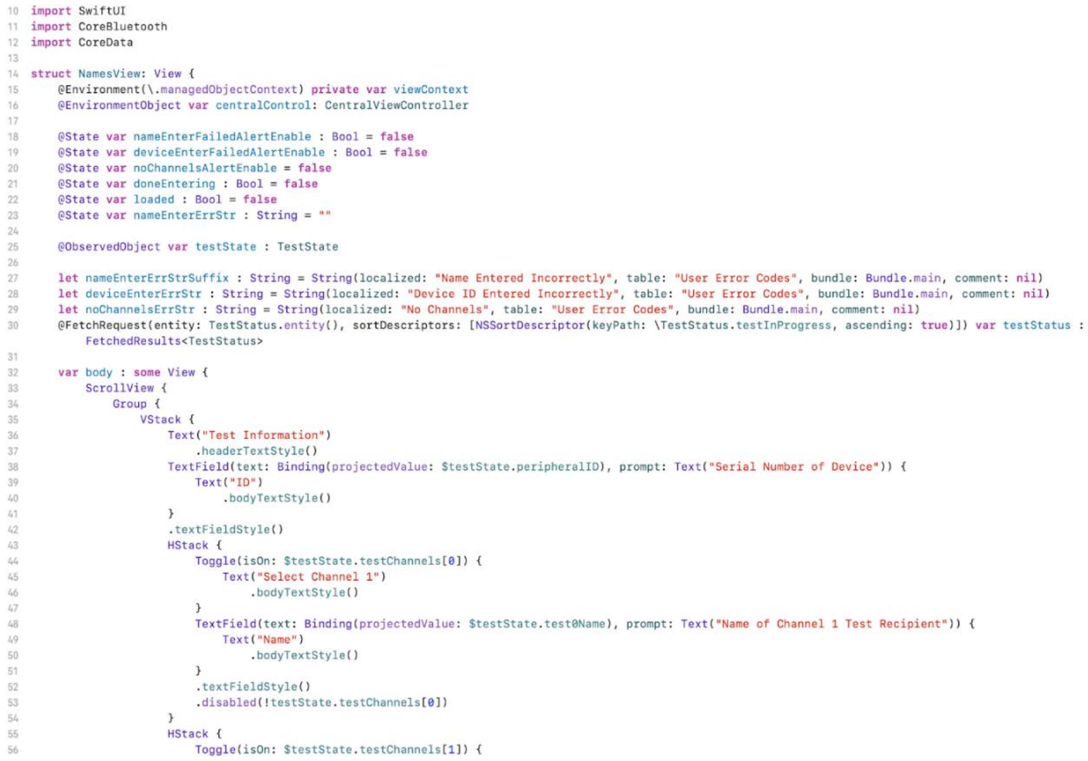


```
extension CentralViewController: CBPeripheralDelegate {

    func peripheral(_ peripheral: CBPeripheral, didDiscoverServices error: Error?) {
        let testServIdx = (connectedPeripheral?.services?.firstIndex(where: {$0.uuid == DeviceServices.TestStatusServiceUUID})) ?? 0
        let colorSensorServIdx = (connectedPeripheral?.services?.firstIndex(where: {$0.uuid == DeviceServices.ColorSensorServiceUUID})) ?? 0
        let LEDServIdx = (connectedPeripheral?.services?.firstIndex(where: {$0.uuid == DeviceServices.LEDModeServiceUUID})) ?? 0
        connectedPeripheral?.discoverCharacteristics([DeviceServices.TestInProgressCharacteristicUUID], for: (connectedPeripheral?.services?[testServIdx]) as! CBService)
        connectedPeripheral?.discoverCharacteristics([DeviceServices.ColorSensor1CharacteristicUUID, DeviceServices.ColorSensor2CharacteristicUUID,
            DeviceServices.ColorSensor3CharacteristicUUID, DeviceServices.ColorSensor4CharacteristicUUID, DeviceServices.ColorSensor5CharacteristicUUID,
            DeviceServices.ColorSensor6CharacteristicUUID, DeviceServices.ColorSensor7CharacteristicUUID, DeviceServices.ColorSensor8CharacteristicUUID], for:
            (connectedPeripheral?.services?[colorSensorServIdx]) as! CBService)
        connectedPeripheral?.discoverCharacteristics([DeviceServices.LEDControlCharacteristicUUID], for: (connectedPeripheral?.services?[LEDServIdx]) as! CBService)
        if error != nil {
            discoveryError = .failedServiceDiscovery
        }
    }

    func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService, error: Error?) {
        if error != nil {
            discoveryError = .failedCharactersticDiscovery
        }
    }

    func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic, error: Error?) {
        if error != nil {
            communicationError = .failedRead(id: characteristic.uuid)
            return
        }
        var dataArray : [UInt16] = []
        characteristic.value!.withUnsafeBytes{ dataBytes in
            let buffer: UnsafePointer<UInt16> = dataBytes.baseAddress!.assumingMemoryBound(to: UInt16.self)
            dataArray = Array(UnsafeBufferPointer(start: buffer, count: characteristic.value!.count / MemoryLayout<UInt16>.size))
        }
        charCachedValue[characteristic.uuid] = dataArray
        NotificationCenter.default.post(name: Notification.CharDataReady, object: nil)
    }

    func doesPossessChar(characteristic: CBUUID) {
        discoveryError = .failedCharactersticDiscovery
        if let count = connectedPeripheral?.services?.count {
            for i in 0..<count {
                if connectedPeripheral?.services?[i].characteristics?.first(where: {$0.uuid == characteristic}) != nil {
                    discoveryError = nil
                }
            }
        }
    }
```

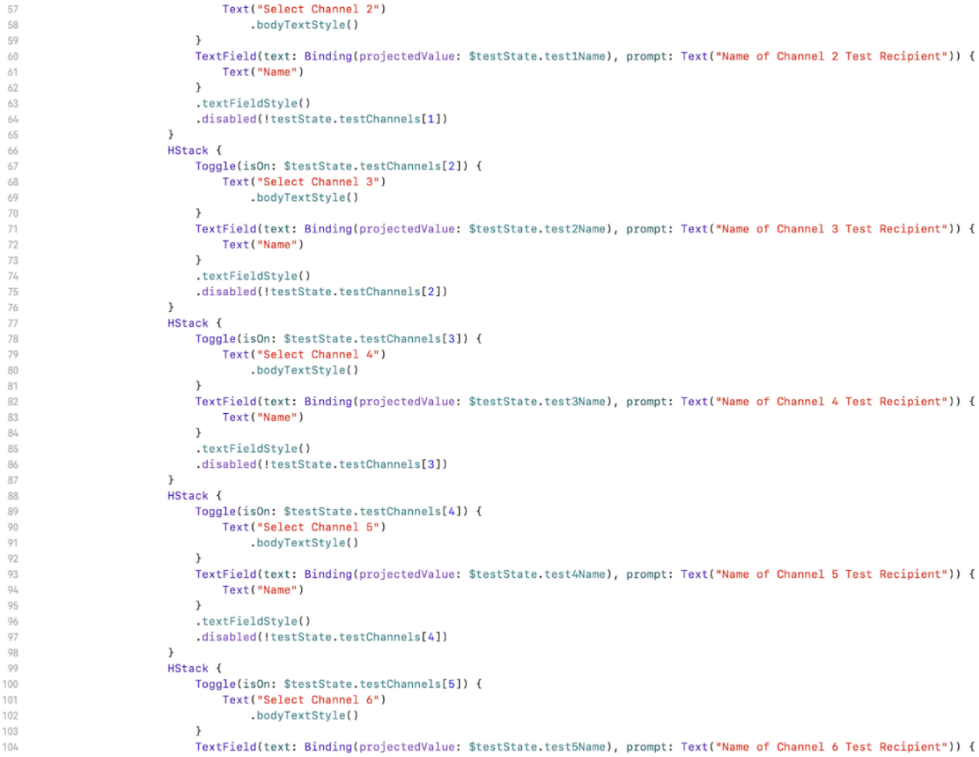


```
    func doesPossessServ(service: CBUUID) {
        discoveryError = nil
        guard connectedPeripheral?.services?.first(where: {$0.uuid == service}) != nil else {
            discoveryError = .failedServiceDiscovery
            return
        }
    }
}
```

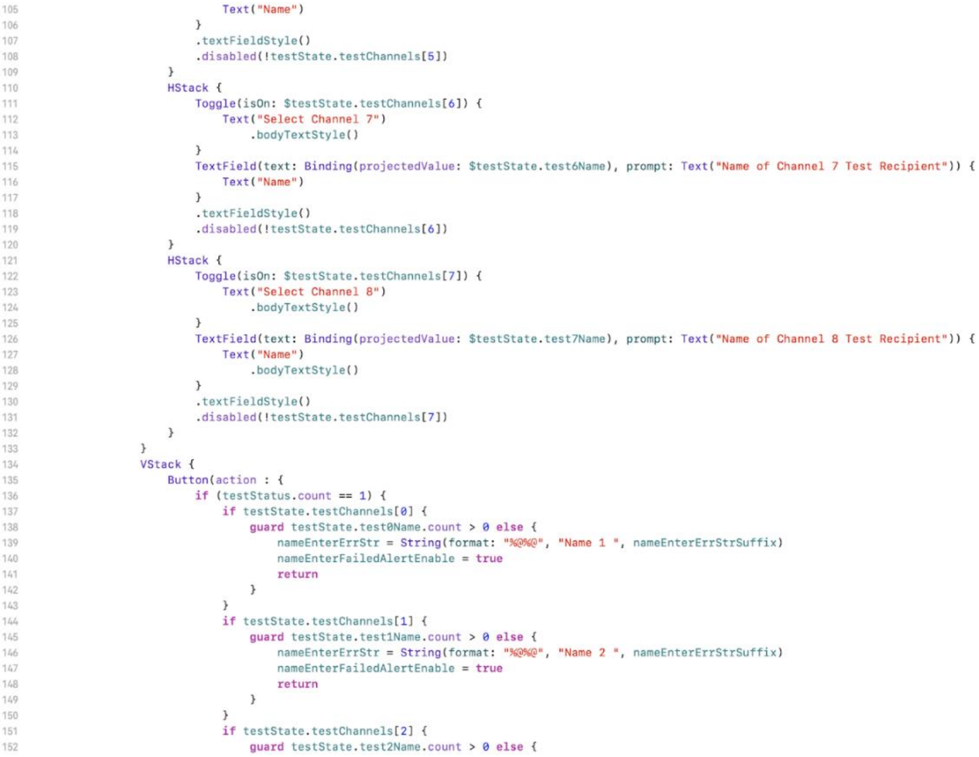
NamesView.swift



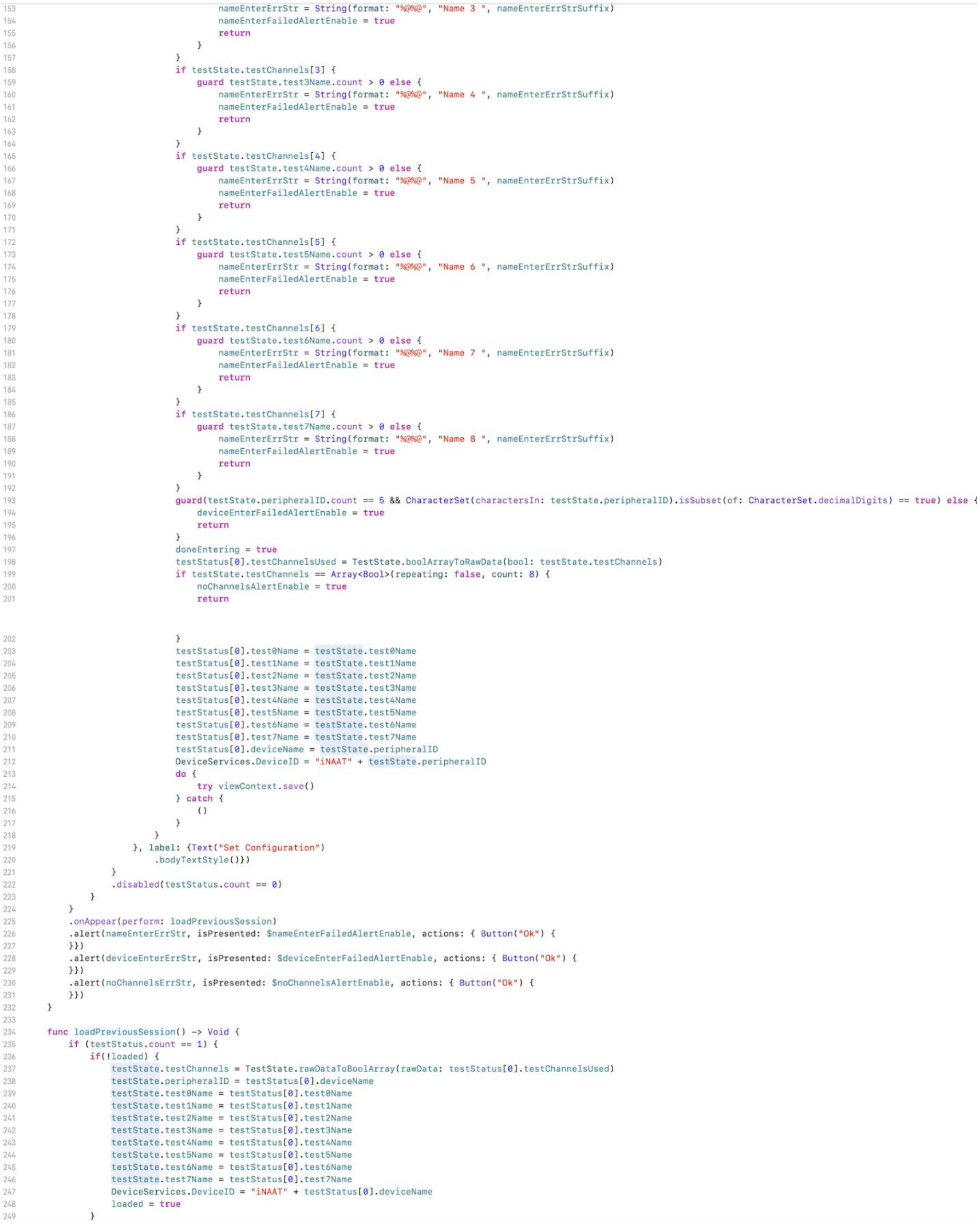
```swift
import SwiftUI
import CoreBluetooth
import CoreData

struct NamesView: View {
    @Environment(\.managedObjectContext) private var viewContext
    @EnvironmentObject var centralControl: CentralViewController

    @State var nameEnterFailedAlertEnable : Bool = false
    @State var deviceEnterFailedAlertEnable : Bool = false
    @State var noChannelsAlertEnable = false
    @State var doneEntering : Bool = false
    @State var loaded : Bool = false
    @State var nameEnterErrStr : String = ""

    @ObservedObject var testState : TestState

    let nameEnterErrStrSuffix : String = String(localized: "Name Entered Incorrectly", table: "User Error Codes", bundle: Bundle.main, comment: nil)
    let deviceEnterErrStr : String = String(localized: "Device ID Entered Incorrectly", table: "User Error Codes", bundle: Bundle.main, comment: nil)
    let noChannelsErrStr : String = String(localized: "No Channels", table: "User Error Codes", bundle: Bundle.main, comment: nil)
    @FetchRequest(entity: TestStatus.entity(), sortDescriptors: [NSSortDescriptor(keyPath: \TestStatus.testInProgress, ascending: true)]) var testStatus :
        FetchedResults<TestStatus>

    var body : some View {
        ScrollView {
            Group {
                VStack {
                    Text("Test Information")
                        .headerTextStyle()
                    TextField(text: Binding(projectedValue: $testState.peripheralID), prompt: Text("Serial Number of Device")) {
                        Text("ID")
                            .bodyTextStyle()
                    }
                    .textFieldStyle()
                    HStack {
                        Toggle(isOn: $testState.testChannels[0]) {
                            Text("Select Channel 1")
                                .bodyTextStyle()
                        }
                        TextField(text: Binding(projectedValue: $testState.test0Name), prompt: Text("Name of Channel 1 Test Recipient")) {
                            Text("Name")
                                .bodyTextStyle()
                        }
                        .textFieldStyle()
                        .disabled(!testState.testChannels[0])
                    }
                    HStack {
                        Toggle(isOn: $testState.testChannels[1]) {
```



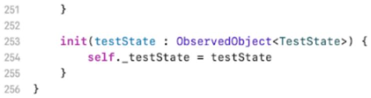
```
            Text("Select Channel 2")
                .bodyTextStyle()
        }
        TextField(text: Binding(projectedValue: $testState.test1Name), prompt: Text("Name of Channel 2 Test Recipient")) {
            Text("Name")
        }
        .textFieldStyle()
        .disabled(!testState.testChannels[1])
    }
    HStack {
        Toggle(isOn: $testState.testChannels[2]) {
            Text("Select Channel 3")
                .bodyTextStyle()
        }
        TextField(text: Binding(projectedValue: $testState.test2Name), prompt: Text("Name of Channel 3 Test Recipient")) {
            Text("Name")
        }
        .textFieldStyle()
        .disabled(!testState.testChannels[2])
    }
    HStack {
        Toggle(isOn: $testState.testChannels[3]) {
            Text("Select Channel 4")
                .bodyTextStyle()
        }
        TextField(text: Binding(projectedValue: $testState.test3Name), prompt: Text("Name of Channel 4 Test Recipient")) {
            Text("Name")
        }
        .textFieldStyle()
        .disabled(!testState.testChannels[3])
    }
    HStack {
        Toggle(isOn: $testState.testChannels[4]) {
            Text("Select Channel 5")
                .bodyTextStyle()
        }
        TextField(text: Binding(projectedValue: $testState.test4Name), prompt: Text("Name of Channel 5 Test Recipient")) {
            Text("Name")
        }
        .textFieldStyle()
        .disabled(!testState.testChannels[4])
    }
    HStack {
        Toggle(isOn: $testState.testChannels[5]) {
            Text("Select Channel 6")
                .bodyTextStyle()
        }
        TextField(text: Binding(projectedValue: $testState.test5Name), prompt: Text("Name of Channel 6 Test Recipient")) {
```



```
            Text("Name")
        }
        .textFieldStyle()
        .disabled(!testState.testChannels[5])
    }
    HStack {
        Toggle(isOn: $testState.testChannels[6]) {
            Text("Select Channel 7")
                .bodyTextStyle()
        }
        TextField(text: Binding(projectedValue: $testState.test6Name), prompt: Text("Name of Channel 7 Test Recipient")) {
            Text("Name")
        }
        .textFieldStyle()
        .disabled(!testState.testChannels[6])
    }
    HStack {
        Toggle(isOn: $testState.testChannels[7]) {
            Text("Select Channel 8")
                .bodyTextStyle()
        }
        TextField(text: Binding(projectedValue: $testState.test7Name), prompt: Text("Name of Channel 8 Test Recipient")) {
            Text("Name")
                .bodyTextStyle()
        }
        .textFieldStyle()
        .disabled(!testState.testChannels[7])
    }
}
VStack {
    Button(action : {
        if (testStatus.count == 1) {
            if testState.testChannels[0] {
                guard testState.test0Name.count > 0 else {
                    nameEnterErrStr = String(format: "%@%@", "Name 1 ", nameEnterErrStrSuffix)
                    nameEnterFailedAlertEnable = true
                    return
                }
            }
            if testState.testChannels[1] {
                guard testState.test1Name.count > 0 else {
                    nameEnterErrStr = String(format: "%@%@", "Name 2 ", nameEnterErrStrSuffix)
                    nameEnterFailedAlertEnable = true
                    return
                }
            }
            if testState.testChannels[2] {
                guard testState.test2Name.count > 0 else {
```

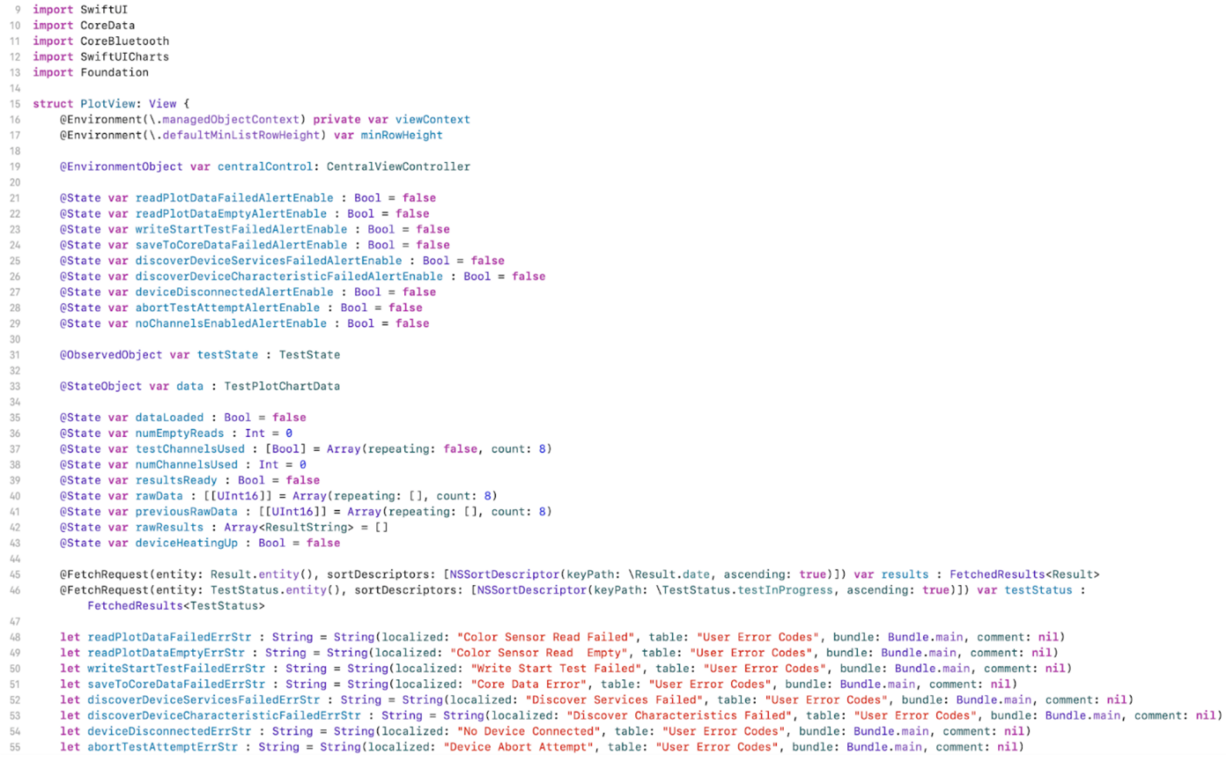


```
                                    nameEnterErrStr = String(format: "%@%@", "Name 3 ", nameEnterErrStrSuffix)
                                    nameEnterFailedAlertEnable = true
                                    return
                                }
                            }
                            if testState.testChannels[3] {
                                guard testState.test3Name.count > 0 else {
                                    nameEnterErrStr = String(format: "%@%@", "Name 4 ", nameEnterErrStrSuffix)
                                    nameEnterFailedAlertEnable = true
                                    return
                                }
                            }
                            if testState.testChannels[4] {
                                guard testState.test4Name.count > 0 else {
                                    nameEnterErrStr = String(format: "%@%@", "Name 5 ", nameEnterErrStrSuffix)
                                    nameEnterFailedAlertEnable = true
                                    return
                                }
                            }
                            if testState.testChannels[5] {
                                guard testState.test5Name.count > 0 else {
                                    nameEnterErrStr = String(format: "%@%@", "Name 6 ", nameEnterErrStrSuffix)
                                    nameEnterFailedAlertEnable = true
                                    return
                                }
                            }
                            if testState.testChannels[6] {
                                guard testState.test6Name.count > 0 else {
                                    nameEnterErrStr = String(format: "%@%@", "Name 7 ", nameEnterErrStrSuffix)
                                    nameEnterFailedAlertEnable = true
                                    return
                                }
                            }
                            if testState.testChannels[7] {
                                guard testState.test7Name.count > 0 else {
                                    nameEnterErrStr = String(format: "%@%@", "Name 8 ", nameEnterErrStrSuffix)
                                    nameEnterFailedAlertEnable = true
                                    return
                                }
                            }
                            guard(testState.peripheralID.count == 5 && CharacterSet(charactersIn: testState.peripheralID).isSubset(of: CharacterSet.decimalDigits) == true) else {
                                deviceEnterFailedAlertEnable = true
                                return
                            }
                            doneEntering = true
                            testStatus[0].testChannelsUsed = TestState.boolArrayToRawData(bool: testState.testChannels)
                            if testState.testChannels == Array<Bool>(repeating: false, count: 8) {
                                noChannelsAlertEnable = true
                                return


                            }
                            testStatus[0].test0Name = testState.test0Name
                            testStatus[0].test1Name = testState.test1Name
                            testStatus[0].test2Name = testState.test2Name
                            testStatus[0].test3Name = testState.test3Name
                            testStatus[0].test4Name = testState.test4Name
                            testStatus[0].test5Name = testState.test5Name
                            testStatus[0].test6Name = testState.test6Name
                            testStatus[0].test7Name = testState.test7Name
                            testStatus[0].deviceName = testState.peripheralID
                            DeviceServices.DeviceID = "iNAAT" + testState.peripheralID
                            do {
                                try viewContext.save()
                            } catch {
                                ()
                            }
                        }
                    }, label: {Text("Set Configuration")
                        .bodyTextStyle()})
                }
                .disabled(testStatus.count == 0)
            }
        }
        .onAppear(perform: loadPreviousSession)
        .alert(nameEnterErrStr, isPresented: $nameEnterFailedAlertEnable, actions: { Button("Ok") {
        }})
        .alert(deviceEnterErrStr, isPresented: $deviceEnterFailedAlertEnable, actions: { Button("Ok") {
        }})
        .alert(noChannelsErrStr, isPresented: $noChannelsAlertEnable, actions: { Button("Ok") {
        }})
    }

    func loadPreviousSession() -> Void {
        if (testStatus.count == 1) {
            if(!loaded) {
                testState.testChannels = TestState.rawDataToBoolArray(rawData: testStatus[0].testChannelsUsed)
                testState.peripheralID = testStatus[0].deviceName
                testState.test0Name = testStatus[0].test0Name
                testState.test1Name = testStatus[0].test1Name
                testState.test2Name = testStatus[0].test2Name
                testState.test3Name = testStatus[0].test3Name
                testState.test4Name = testStatus[0].test4Name
                testState.test5Name = testStatus[0].test5Name
                testState.test6Name = testStatus[0].test6Name
                testState.test7Name = testStatus[0].test7Name
                DeviceServices.DeviceID = "iNAAT" + testStatus[0].deviceName
                loaded = true
            }
```

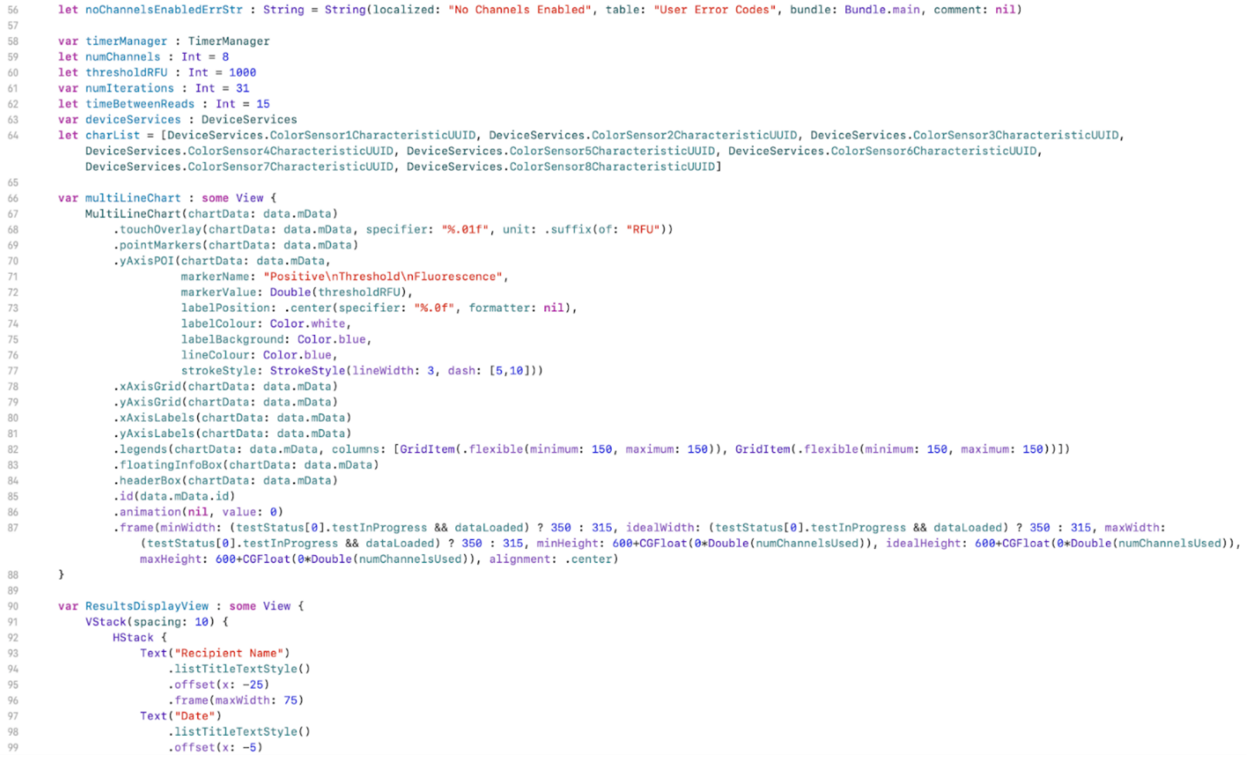


```
    }

    init(testState : ObservedObject<TestState>) {
        self._testState = testState
    }
}
```

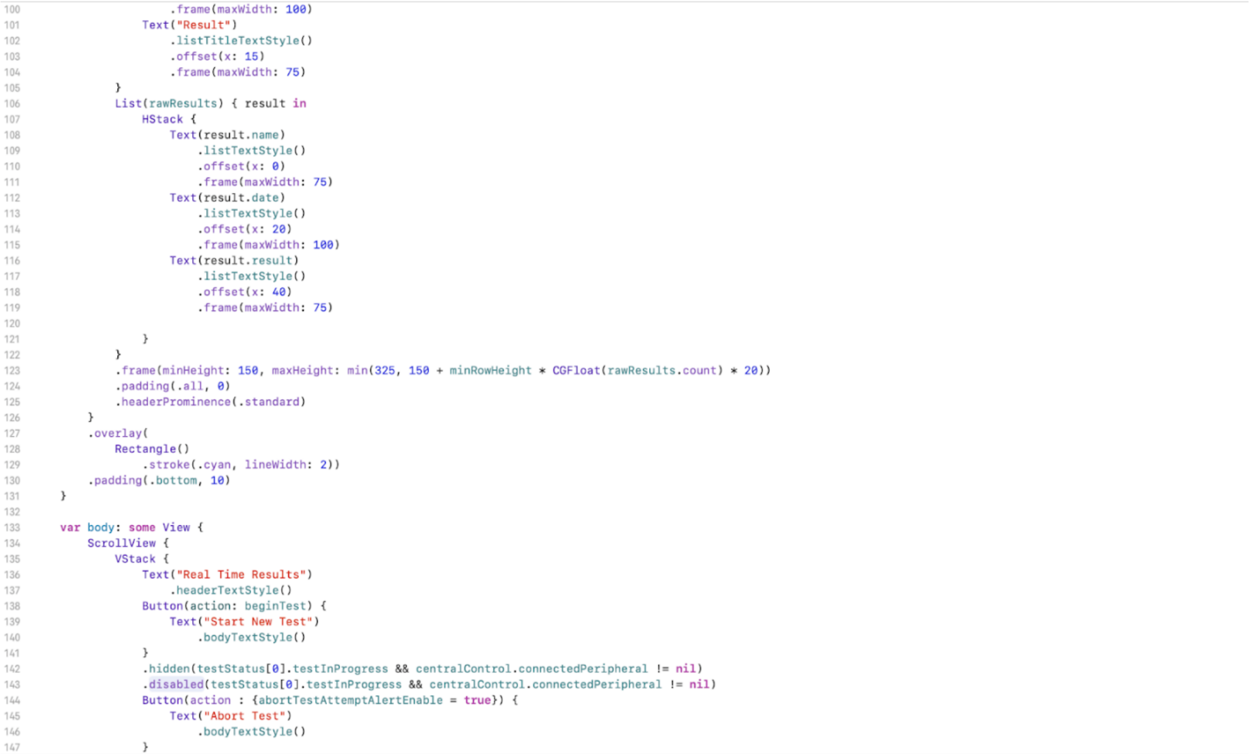
PlotView.swift



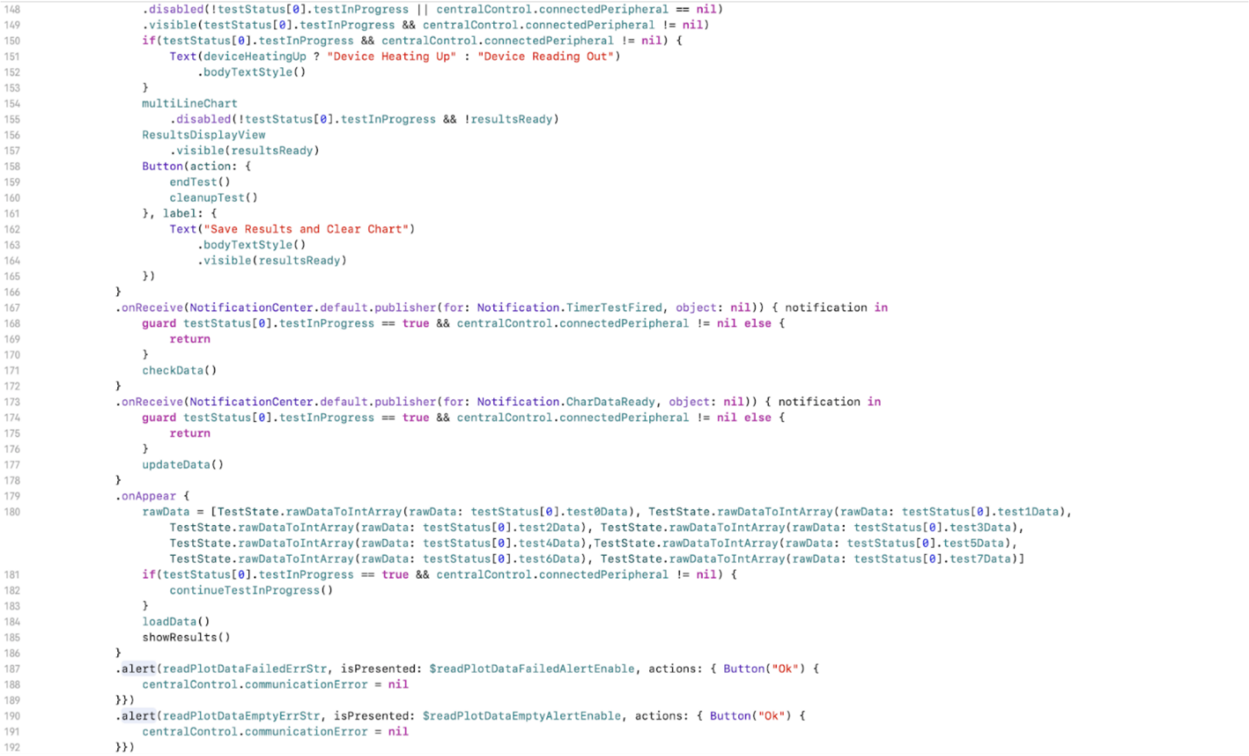
```
import SwiftUI
import CoreData
import CoreBluetooth
import SwiftUICharts
import Foundation

struct PlotView: View {
    @Environment(\.managedObjectContext) private var viewContext
    @Environment(\.defaultMinListRowHeight) var minRowHeight

    @EnvironmentObject var centralControl: CentralViewController

    @State var readPlotDataFailedAlertEnable : Bool = false
    @State var readPlotDataEmptyAlertEnable : Bool = false
    @State var writeStartTestFailedAlertEnable : Bool = false
    @State var saveToCoreDataFailedAlertEnable : Bool = false
    @State var discoverDeviceServicesFailedAlertEnable : Bool = false
    @State var discoverDeviceCharacteristicFailedAlertEnable : Bool = false
    @State var deviceDisconnectedAlertEnable : Bool = false
    @State var abortTestAttemptAlertEnable : Bool = false
    @State var noChannelsEnabledAlertEnable : Bool = false

    @ObservedObject var testState : TestState

    @StateObject var data : TestPlotChartData

    @State var dataLoaded : Bool = false
    @State var numEmptyReads : Int = 0
    @State var testChannelsUsed : [Bool] = Array(repeating: false, count: 8)
    @State var numChannelsUsed : Int = 0
    @State var resultsReady : Bool = false
    @State var rawData : [[UInt16]] = Array(repeating: [], count: 8)
    @State var previousRawData : [[UInt16]] = Array(repeating: [], count: 8)
    @State var rawResults : Array<ResultString> = []
    @State var deviceHeatingUp : Bool = false

    @FetchRequest(entity: Result.entity(), sortDescriptors: [NSSortDescriptor(keyPath: \Result.date, ascending: true)]) var results : FetchedResults<Result>
    @FetchRequest(entity: TestStatus.entity(), sortDescriptors: [NSSortDescriptor(keyPath: \TestStatus.testInProgress, ascending: true)]) var testStatus :
        FetchedResults<TestStatus>

    let readPlotDataFailedErrStr : String = String(localized: "Color Sensor Read Failed", table: "User Error Codes", bundle: Bundle.main, comment: nil)
    let readPlotDataEmptyErrStr : String = String(localized: "Color Sensor Read  Empty", table: "User Error Codes", bundle: Bundle.main, comment: nil)
    let writeStartTestFailedErrStr : String = String(localized: "Write Start Test Failed", table: "User Error Codes", bundle: Bundle.main, comment: nil)
    let saveToCoreDataFailedErrStr : String = String(localized: "Core Data Error", table: "User Error Codes", bundle: Bundle.main, comment: nil)
    let discoverDeviceServicesFailedErrStr : String = String(localized: "Discover Services Failed", table: "User Error Codes", bundle: Bundle.main, comment: nil)
    let discoverDeviceCharacteristicFailedErrStr : String = String(localized: "Discover Characteristics Failed", table: "User Error Codes", bundle: Bundle.main, comment: nil)
    let deviceDisconnectedErrStr : String = String(localized: "No Device Connected", table: "User Error Codes", bundle: Bundle.main, comment: nil)
    let abortTestAttemptErrStr : String = String(localized: "Device Abort Attempt", table: "User Error Codes", bundle: Bundle.main, comment: nil)
```



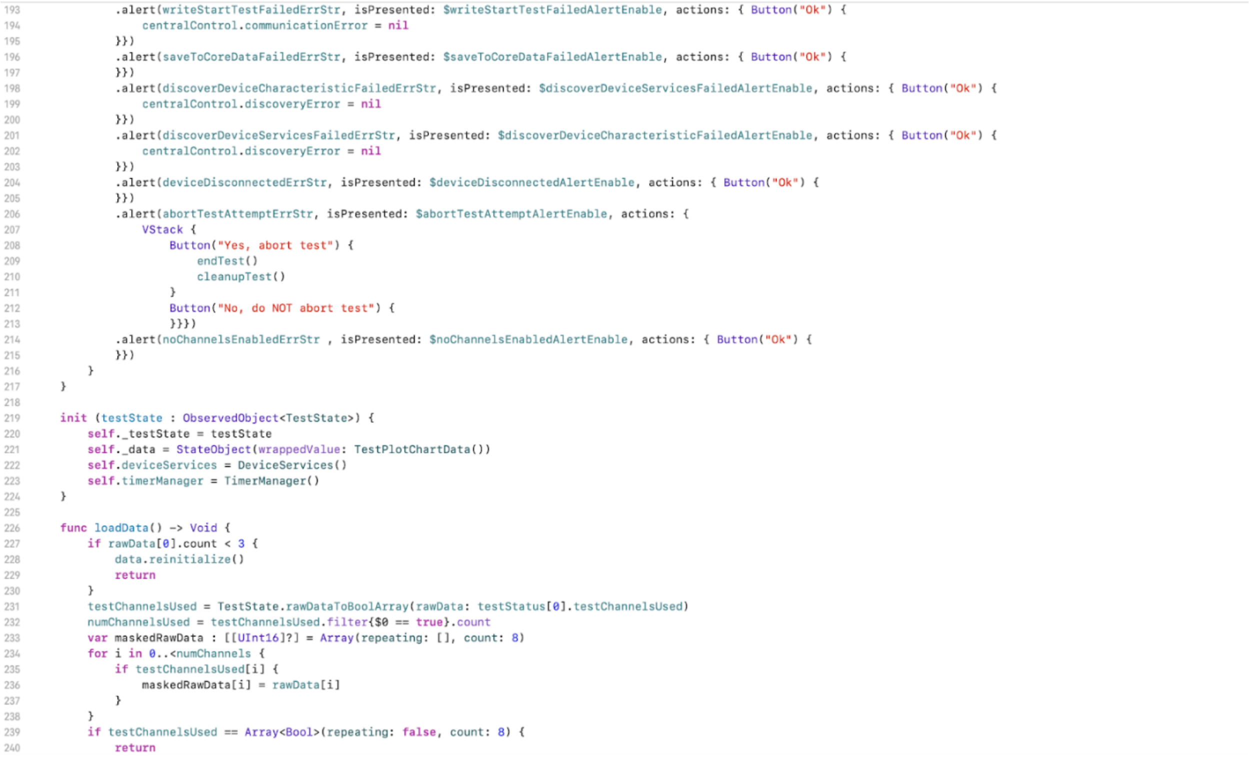
```swift
let noChannelsEnabledErrStr : String = String(localized: "No Channels Enabled", table: "User Error Codes", bundle: Bundle.main, comment: nil)

var timerManager : TimerManager
let numChannels : Int = 8
let thresholdRFU : Int = 1000
var numIterations : Int = 31
let timeBetweenReads : Int = 15
var deviceServices : DeviceServices
let charList = [DeviceServices.ColorSensor1CharacteristicUUID, DeviceServices.ColorSensor2CharacteristicUUID, DeviceServices.ColorSensor3CharacteristicUUID,
    DeviceServices.ColorSensor4CharacteristicUUID, DeviceServices.ColorSensor5CharacteristicUUID, DeviceServices.ColorSensor6CharacteristicUUID,
    DeviceServices.ColorSensor7CharacteristicUUID, DeviceServices.ColorSensor8CharacteristicUUID]

var multiLineChart : some View {
    MultiLineChart(chartData: data.mData)
        .touchOverlay(chartData: data.mData, specifier: "%.01f", unit: .suffix(of: "RFU"))
        .pointMarkers(chartData: data.mData)
        .yAxisPOI(chartData: data.mData,
                  markerName: "Positive\nThreshold\nFluorescence",
                  markerValue: Double(thresholdRFU),
                  labelPosition: .center(specifier: "%.0f", formatter: nil),
                  labelColour: Color.white,
                  labelBackground: Color.blue,
                  lineColour: Color.blue,
                  strokeStyle: StrokeStyle(lineWidth: 3, dash: [5,10]))
        .xAxisGrid(chartData: data.mData)
        .yAxisGrid(chartData: data.mData)
        .xAxisLabels(chartData: data.mData)
        .yAxisLabels(chartData: data.mData)
        .legends(chartData: data.mData, columns: [GridItem(.flexible(minimum: 150, maximum: 150)), GridItem(.flexible(minimum: 150, maximum: 150))])
        .floatingInfoBox(chartData: data.mData)
        .headerBox(chartData: data.mData)
        .id(data.mData.id)
        .animation(nil, value: 0)
        .frame(minWidth: (testStatus[0].testInProgress && dataLoaded) ? 350 : 315, idealWidth: (testStatus[0].testInProgress && dataLoaded) ? 350 : 315, maxWidth:
            (testStatus[0].testInProgress && dataLoaded) ? 350 : 315, minHeight: 600+CGFloat(0*Double(numChannelsUsed)), idealHeight: 600+CGFloat(0*Double(numChannelsUsed)),
            maxHeight: 600+CGFloat(0*Double(numChannelsUsed)), alignment: .center)
}

var ResultsDisplayView : some View {
    VStack(spacing: 10) {
        HStack {
            Text("Recipient Name")
                .listTitleTextStyle()
                .offset(x: -25)
                .frame(maxWidth: 75)
            Text("Date")
                .listTitleTextStyle()
                .offset(x: -5)
```



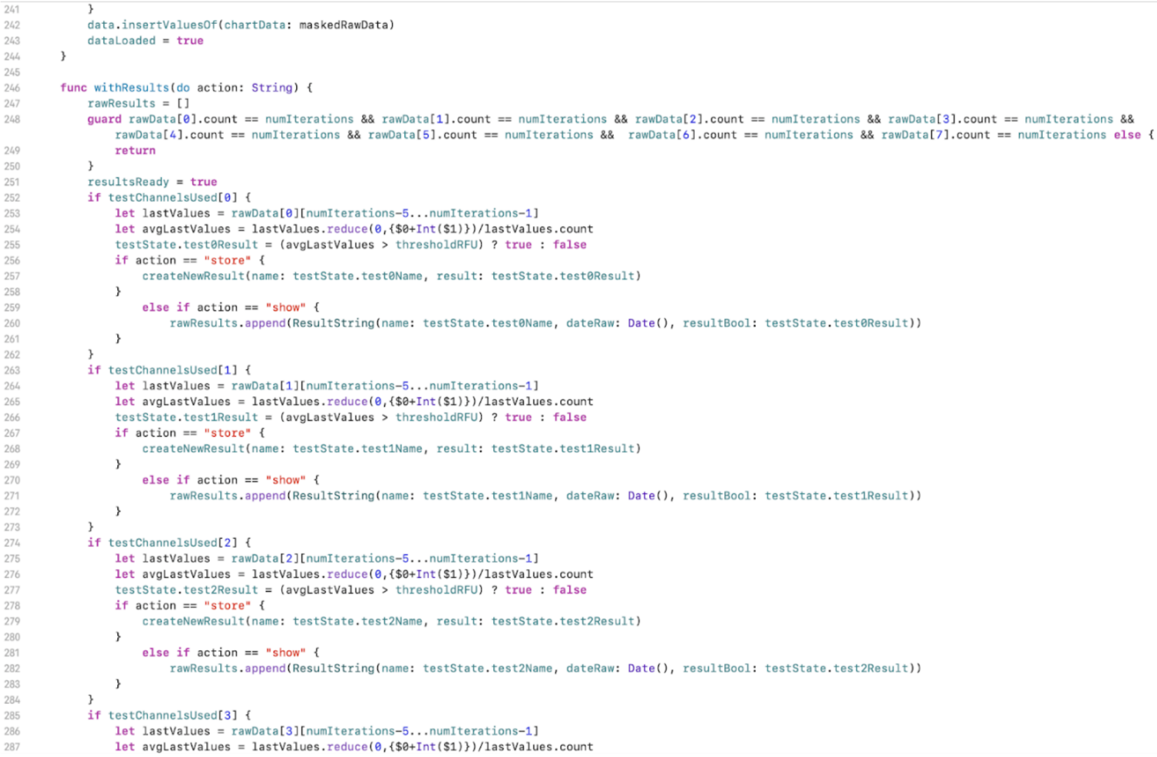
```swift
                .frame(maxWidth: 100)
            Text("Result")
                .listTitleTextStyle()
                .offset(x: 15)
                .frame(maxWidth: 75)
        }
        List(rawResults) { result in
            HStack {
                Text(result.name)
                    .listTextStyle()
                    .offset(x: 0)
                    .frame(maxWidth: 75)
                Text(result.date)
                    .listTextStyle()
                    .offset(x: 20)
                    .frame(maxWidth: 100)
                Text(result.result)
                    .listTextStyle()
                    .offset(x: 40)
                    .frame(maxWidth: 75)

            }
        }
        .frame(minHeight: 150, maxHeight: min(325, 150 + minRowHeight * CGFloat(rawResults.count) * 20))
        .padding(.all, 0)
        .headerProminence(.standard)
    }
    .overlay(
        Rectangle()
            .stroke(.cyan, lineWidth: 2))
    .padding(.bottom, 10)
}

var body: some View {
    ScrollView {
        VStack {
            Text("Real Time Results")
                .headerTextStyle()
            Button(action: beginTest) {
                Text("Start New Test")
                    .bodyTextStyle()
            }
            .hidden(testStatus[0].testInProgress && centralControl.connectedPeripheral != nil)
            .disabled(testStatus[0].testInProgress && centralControl.connectedPeripheral != nil)
            Button(action : {abortTestAttemptAlertEnable = true}) {
                Text("Abort Test")
                    .bodyTextStyle()
            }
```



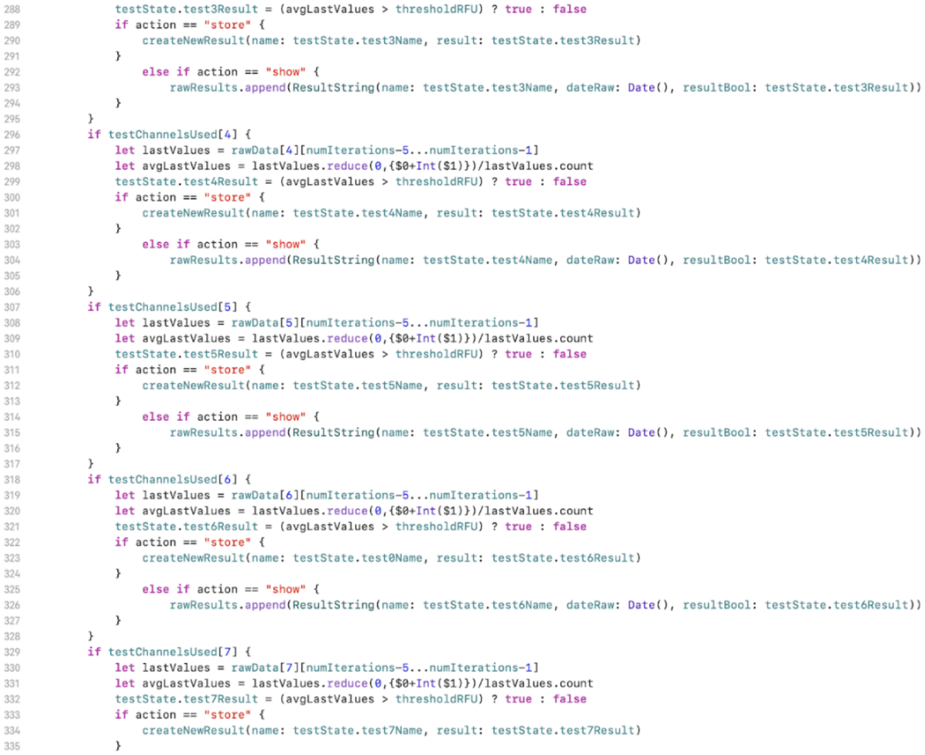
```
                .disabled(!testStatus[0].testInProgress || centralControl.connectedPeripheral == nil)
                .visible(testStatus[0].testInProgress && centralControl.connectedPeripheral != nil)
                if(testStatus[0].testInProgress && centralControl.connectedPeripheral != nil) {
                    Text(deviceHeatingUp ? "Device Heating Up" : "Device Reading Out")
                        .bodyTextStyle()
                }
                multiLineChart
                    .disabled(!testStatus[0].testInProgress && !resultsReady)
                ResultsDisplayView
                    .visible(resultsReady)
                Button(action: {
                    endTest()
                    cleanupTest()
                }, label: {
                    Text("Save Results and Clear Chart")
                        .bodyTextStyle()
                        .visible(resultsReady)
                })
            }
            .onReceive(NotificationCenter.default.publisher(for: Notification.TimerTestFired, object: nil)) { notification in
                guard testStatus[0].testInProgress == true && centralControl.connectedPeripheral != nil else {
                    return
                }
                checkData()
            }
            .onReceive(NotificationCenter.default.publisher(for: Notification.CharDataReady, object: nil)) { notification in
                guard testStatus[0].testInProgress == true && centralControl.connectedPeripheral != nil else {
                    return
                }
                updateData()
            }
            .onAppear {
                rawData = [TestState.rawDataToIntArray(rawData: testStatus[0].test0Data), TestState.rawDataToIntArray(rawData: testStatus[0].test1Data),
                    TestState.rawDataToIntArray(rawData: testStatus[0].test2Data), TestState.rawDataToIntArray(rawData: testStatus[0].test3Data),
                    TestState.rawDataToIntArray(rawData: testStatus[0].test4Data),TestState.rawDataToIntArray(rawData: testStatus[0].test5Data),
                    TestState.rawDataToIntArray(rawData: testStatus[0].test6Data), TestState.rawDataToIntArray(rawData: testStatus[0].test7Data)]
                if(testStatus[0].testInProgress == true && centralControl.connectedPeripheral != nil) {
                    continueTestInProgress()
                }
                loadData()
                showResults()
            }
            .alert(readPlotDataFailedErrStr, isPresented: $readPlotDataFailedAlertEnable, actions: { Button("Ok") {
                centralControl.communicationError = nil
            }})
            .alert(readPlotDataEmptyErrStr, isPresented: $readPlotDataEmptyAlertEnable, actions: { Button("Ok") {
                centralControl.communicationError = nil
            }})
```



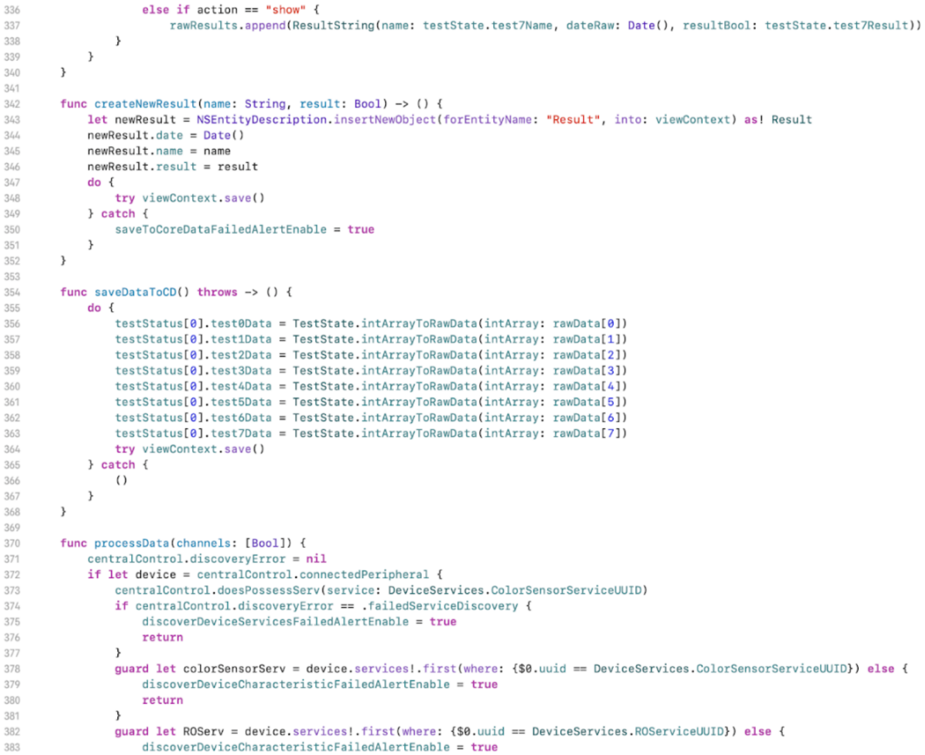
```
            .alert(writeStartTestFailedErrStr, isPresented: $writeStartTestFailedAlertEnable, actions: { Button("Ok") {
                centralControl.communicationError = nil
            }})
            .alert(saveToCoreDataFailedErrStr, isPresented: $saveToCoreDataFailedAlertEnable, actions: { Button("Ok") {
            }})
            .alert(discoverDeviceCharacteristicFailedErrStr, isPresented: $discoverDeviceServicesFailedAlertEnable, actions: { Button("Ok") {
                centralControl.discoveryError = nil
            }})
            .alert(discoverDeviceServicesFailedErrStr, isPresented: $discoverDeviceCharacteristicFailedAlertEnable, actions: { Button("Ok") {
                centralControl.discoveryError = nil
            }})
            .alert(deviceDisconnectedErrStr, isPresented: $deviceDisconnectedAlertEnable, actions: { Button("Ok") {
            }})
            .alert(abortTestAttemptErrStr, isPresented: $abortTestAttemptAlertEnable, actions: {
                VStack {
                    Button("Yes, abort test") {
                        endTest()
                        cleanupTest()
                    }
                    Button("No, do NOT abort test") {
                    }}})
            .alert(noChannelsEnabledErrStr , isPresented: $noChannelsEnabledAlertEnable, actions: { Button("Ok") {
            }})
        }
    }

    init (testState : ObservedObject<TestState>) {
        self._testState = testState
        self._data = StateObject(wrappedValue: TestPlotChartData())
        self.deviceServices = DeviceServices()
        self.timerManager = TimerManager()
    }

    func loadData() -> Void {
        if rawData[0].count < 3 {
            data.reinitialize()
            return
        }
        testChannelsUsed = TestState.rawDataToBoolArray(rawData: testStatus[0].testChannelsUsed)
        numChannelsUsed = testChannelsUsed.filter{$0 == true}.count
        var maskedRawData : [[UInt16]?] = Array(repeating: [], count: 8)
        for i in 0..<numChannels {
            if testChannelsUsed[i] {
                maskedRawData[i] = rawData[i]
            }
        }
        if testChannelsUsed == Array<Bool>(repeating: false, count: 8) {
            return
```



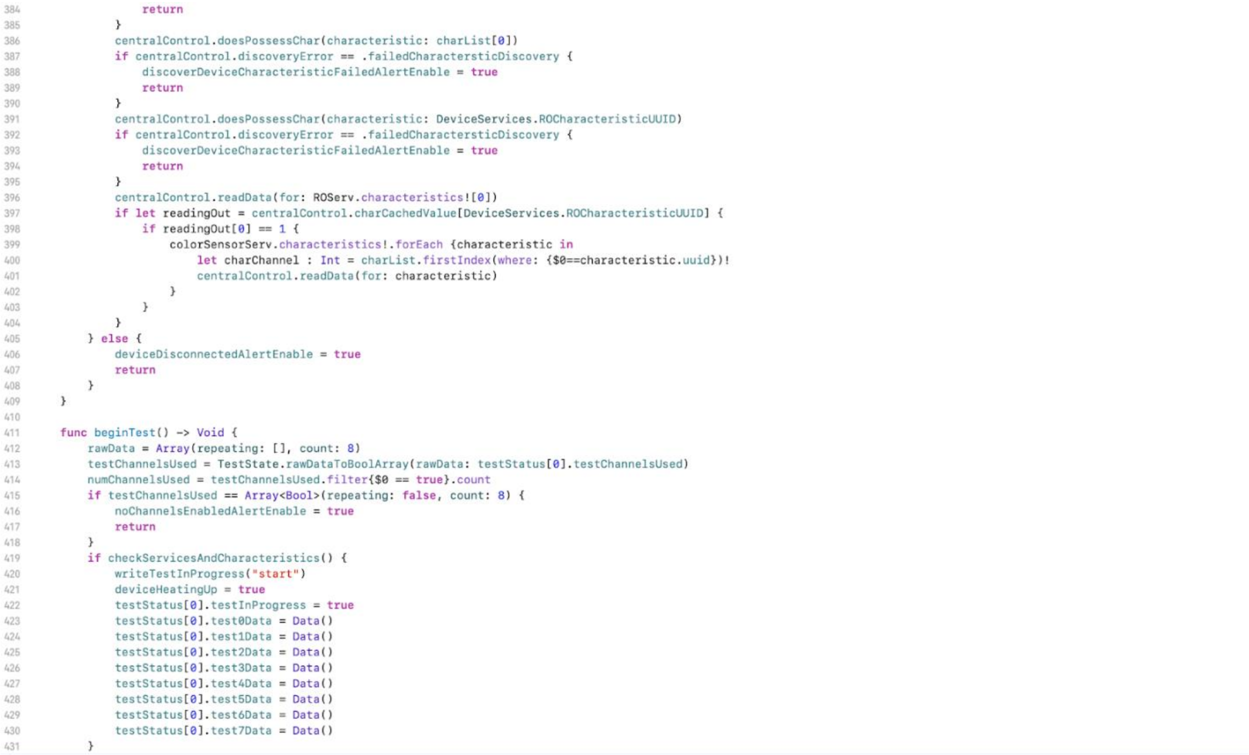
```
        }
        data.insertValuesOf(chartData: maskedRawData)
        dataLoaded = true
    }

    func withResults(do action: String) {
        rawResults = []
        guard rawData[0].count == numIterations && rawData[1].count == numIterations && rawData[2].count == numIterations && rawData[3].count == numIterations &&
            rawData[4].count == numIterations && rawData[5].count == numIterations &&  rawData[6].count == numIterations && rawData[7].count == numIterations else {
            return
        }
        resultsReady = true
        if testChannelsUsed[0] {
            let lastValues = rawData[0][numIterations-5...numIterations-1]
            let avgLastValues = lastValues.reduce(0,{$0+Int($1)})/lastValues.count
            testState.test0Result = (avgLastValues > thresholdRFU) ? true : false
            if action == "store" {
                createNewResult(name: testState.test0Name, result: testState.test0Result)
            }
                else if action == "show" {
                    rawResults.append(ResultString(name: testState.test0Name, dateRaw: Date(), resultBool: testState.test0Result))
            }
        }
        if testChannelsUsed[1] {
            let lastValues = rawData[1][numIterations-5...numIterations-1]
            let avgLastValues = lastValues.reduce(0,{$0+Int($1)})/lastValues.count
            testState.test1Result = (avgLastValues > thresholdRFU) ? true : false
            if action == "store" {
                createNewResult(name: testState.test1Name, result: testState.test1Result)
            }
                else if action == "show" {
                    rawResults.append(ResultString(name: testState.test1Name, dateRaw: Date(), resultBool: testState.test1Result))
            }
        }
        if testChannelsUsed[2] {
            let lastValues = rawData[2][numIterations-5...numIterations-1]
            let avgLastValues = lastValues.reduce(0,{$0+Int($1)})/lastValues.count
            testState.test2Result = (avgLastValues > thresholdRFU) ? true : false
            if action == "store" {
                createNewResult(name: testState.test2Name, result: testState.test2Result)
            }
                else if action == "show" {
                    rawResults.append(ResultString(name: testState.test2Name, dateRaw: Date(), resultBool: testState.test2Result))
            }
        }
        if testChannelsUsed[3] {
            let lastValues = rawData[3][numIterations-5...numIterations-1]
            let avgLastValues = lastValues.reduce(0,{$0+Int($1)})/lastValues.count
```



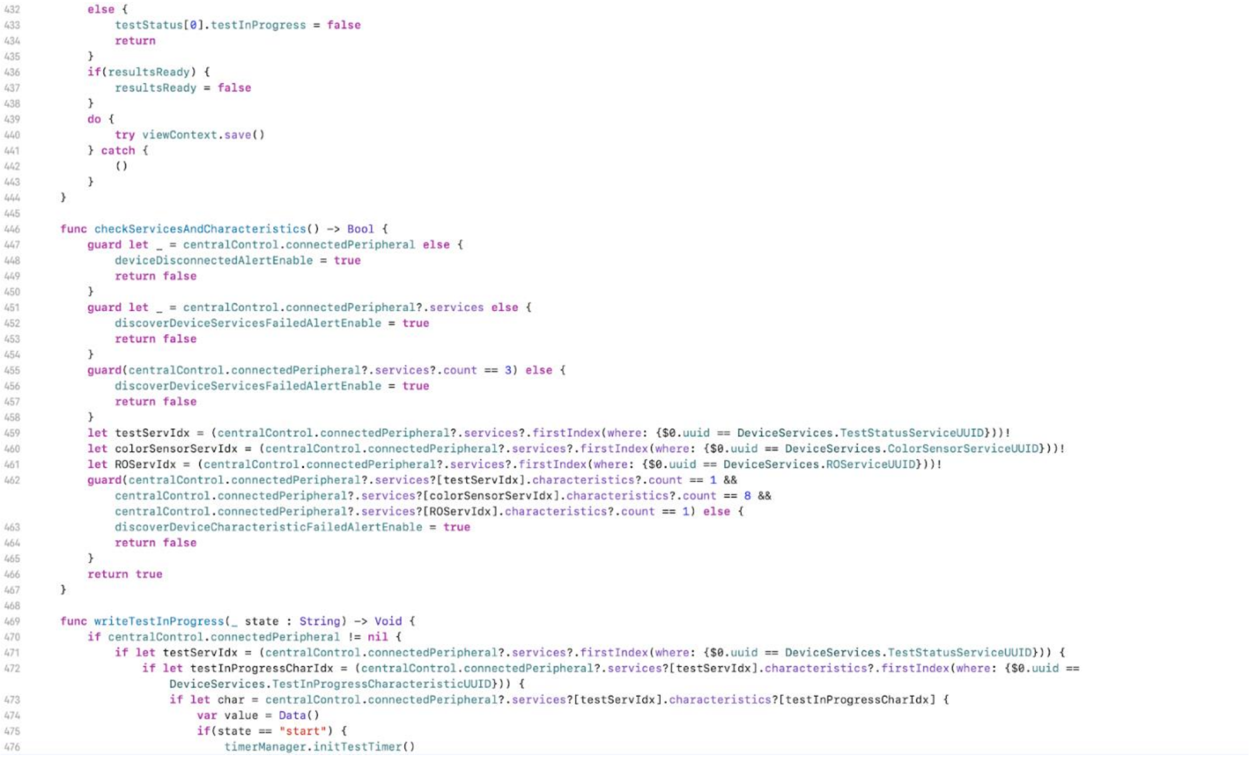
```
            testState.test3Result = (avgLastValues > thresholdRFU) ? true : false
            if action == "store" {
                createNewResult(name: testState.test3Name, result: testState.test3Result)
            }
                else if action == "show" {
                    rawResults.append(ResultString(name: testState.test3Name, dateRaw: Date(), resultBool: testState.test3Result))
            }
        }
        if testChannelsUsed[4] {
            let lastValues = rawData[4][numIterations-5...numIterations-1]
            let avgLastValues = lastValues.reduce(0,{$0+Int($1)})/lastValues.count
            testState.test4Result = (avgLastValues > thresholdRFU) ? true : false
            if action == "store" {
                createNewResult(name: testState.test4Name, result: testState.test4Result)
            }
                else if action == "show" {
                    rawResults.append(ResultString(name: testState.test4Name, dateRaw: Date(), resultBool: testState.test4Result))
            }
        }
        if testChannelsUsed[5] {
            let lastValues = rawData[5][numIterations-5...numIterations-1]
            let avgLastValues = lastValues.reduce(0,{$0+Int($1)})/lastValues.count
            testState.test5Result = (avgLastValues > thresholdRFU) ? true : false
            if action == "store" {
                createNewResult(name: testState.test5Name, result: testState.test5Result)
            }
                else if action == "show" {
                    rawResults.append(ResultString(name: testState.test5Name, dateRaw: Date(), resultBool: testState.test5Result))
            }
        }
        if testChannelsUsed[6] {
            let lastValues = rawData[6][numIterations-5...numIterations-1]
            let avgLastValues = lastValues.reduce(0,{$0+Int($1)})/lastValues.count
            testState.test6Result = (avgLastValues > thresholdRFU) ? true : false
            if action == "store" {
                createNewResult(name: testState.test0Name, result: testState.test6Result)
            }
                else if action == "show" {
                    rawResults.append(ResultString(name: testState.test6Name, dateRaw: Date(), resultBool: testState.test6Result))
            }
        }
        if testChannelsUsed[7] {
            let lastValues = rawData[7][numIterations-5...numIterations-1]
            let avgLastValues = lastValues.reduce(0,{$0+Int($1)})/lastValues.count
            testState.test7Result = (avgLastValues > thresholdRFU) ? true : false
            if action == "store" {
                createNewResult(name: testState.test7Name, result: testState.test7Result)
            }
```



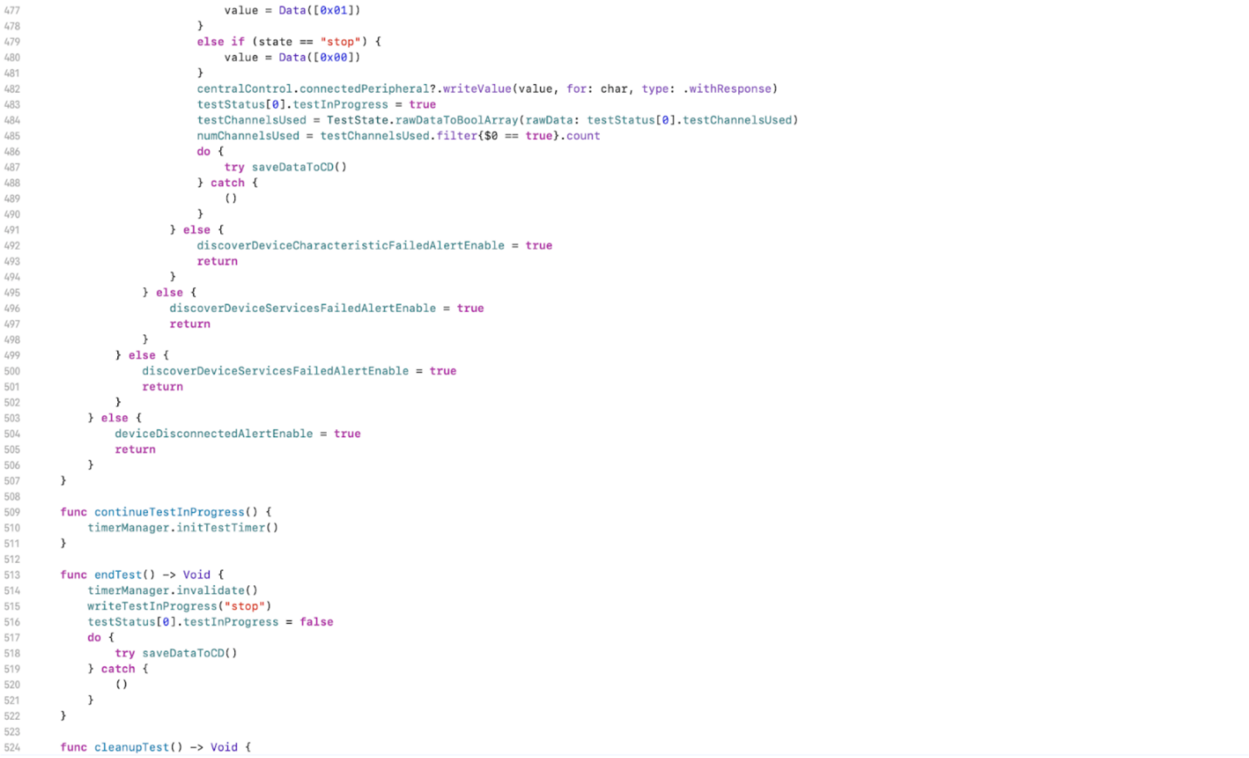
```
                else if action == "show" {
                    rawResults.append(ResultString(name: testState.test7Name, dateRaw: Date(), resultBool: testState.test7Result))
            }
        }
    }

    func createNewResult(name: String, result: Bool) -> () {
        let newResult = NSEntityDescription.insertNewObject(forEntityName: "Result", into: viewContext) as! Result
        newResult.date = Date()
        newResult.name = name
        newResult.result = result
        do {
            try viewContext.save()
        } catch {
            saveToCoreDataFailedAlertEnable = true
        }
    }

    func saveDataToCD() throws -> () {
        do {
            testStatus[0].test0Data = TestState.intArrayToRawData(intArray: rawData[0])
            testStatus[0].test1Data = TestState.intArrayToRawData(intArray: rawData[1])
            testStatus[0].test2Data = TestState.intArrayToRawData(intArray: rawData[2])
            testStatus[0].test3Data = TestState.intArrayToRawData(intArray: rawData[3])
            testStatus[0].test4Data = TestState.intArrayToRawData(intArray: rawData[4])
            testStatus[0].test5Data = TestState.intArrayToRawData(intArray: rawData[5])
            testStatus[0].test6Data = TestState.intArrayToRawData(intArray: rawData[6])
            testStatus[0].test7Data = TestState.intArrayToRawData(intArray: rawData[7])
            try viewContext.save()
        } catch {
            ()
        }
    }

    func processData(channels: [Bool]) {
        centralControl.discoveryError = nil
        if let device = centralControl.connectedPeripheral {
            centralControl.doesPossessServ(service: DeviceServices.ColorSensorServiceUUID)
            if centralControl.discoveryError == .failedServiceDiscovery {
                discoverDeviceServicesFailedAlertEnable = true
                return
            }
            guard let colorSensorServ = device.services!.first(where: {$0.uuid == DeviceServices.ColorSensorServiceUUID}) else {
                discoverDeviceCharacteristicFailedAlertEnable = true
                return
            }
            guard let ROServ = device.services!.first(where: {$0.uuid == DeviceServices.ROServiceUUID}) else {
                discoverDeviceCharacteristicFailedAlertEnable = true
```



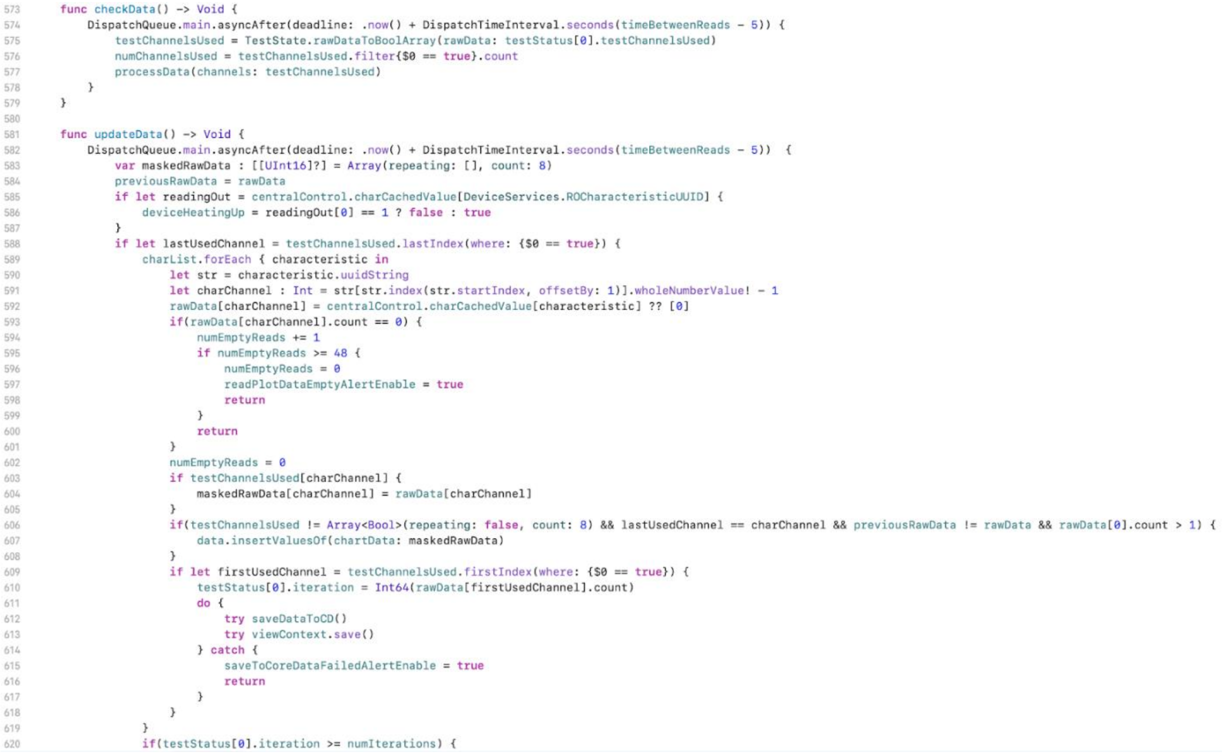
```
                return
            }
            centralControl.doesPossessChar(characteristic: charList[0])
            if centralControl.discoveryError == .failedCharactersticDiscovery {
                discoverDeviceCharacteristicFailedAlertEnable = true
                return
            }
            centralControl.doesPossessChar(characteristic: DeviceServices.ROCharacteristicUUID)
            if centralControl.discoveryError == .failedCharactersticDiscovery {
                discoverDeviceCharacteristicFailedAlertEnable = true
                return
            }
            centralControl.readData(for: ROServ.characteristics![0])
            if let readingOut = centralControl.charCachedValue[DeviceServices.ROCharacteristicUUID] {
                if readingOut[0] == 1 {
                    colorSensorServ.characteristics!.forEach {characteristic in
                        let charChannel : Int = charList.firstIndex(where: {$0==characteristic.uuid})!
                        centralControl.readData(for: characteristic)
                    }
                }
            }
        } else {
            deviceDisconnectedAlertEnable = true
            return
        }
    }

    func beginTest() -> Void {
        rawData = Array(repeating: [], count: 8)
        testChannelsUsed = TestState.rawDataToBoolArray(rawData: testStatus[0].testChannelsUsed)
        numChannelsUsed = testChannelsUsed.filter{$0 == true}.count
        if testChannelsUsed == Array<Bool>(repeating: false, count: 8) {
            noChannelsEnabledAlertEnable = true
            return
        }
        if checkServicesAndCharacteristics() {
            writeTestInProgress("start")
            deviceHeatingUp = true
            testStatus[0].testInProgress = true
            testStatus[0].test0Data = Data()
            testStatus[0].test1Data = Data()
            testStatus[0].test2Data = Data()
            testStatus[0].test3Data = Data()
            testStatus[0].test4Data = Data()
            testStatus[0].test5Data = Data()
            testStatus[0].test6Data = Data()
            testStatus[0].test7Data = Data()
        }
```



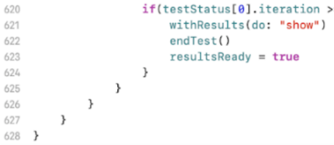
```
        else {
            testStatus[0].testInProgress = false
            return
        }
        if(resultsReady) {
            resultsReady = false
        }
        do {
            try viewContext.save()
        } catch {
            ()
        }
    }

    func checkServicesAndCharacteristics() -> Bool {
        guard let _ = centralControl.connectedPeripheral else {
            deviceDisconnectedAlertEnable = true
            return false
        }
        guard let _ = centralControl.connectedPeripheral?.services else {
            discoverDeviceServicesFailedAlertEnable = true
            return false
        }
        guard(centralControl.connectedPeripheral?.services?.count == 3) else {
            discoverDeviceServicesFailedAlertEnable = true
            return false
        }
        let testServIdx = (centralControl.connectedPeripheral?.services?.firstIndex(where: {$0.uuid == DeviceServices.TestStatusServiceUUID}))!
        let colorSensorServIdx = (centralControl.connectedPeripheral?.services?.firstIndex(where: {$0.uuid == DeviceServices.ColorSensorServiceUUID}))!
        let ROServIdx = (centralControl.connectedPeripheral?.services?.firstIndex(where: {$0.uuid == DeviceServices.ROServiceUUID}))!
        guard(centralControl.connectedPeripheral?.services?[testServIdx].characteristics?.count == 1 &&
            centralControl.connectedPeripheral?.services?[colorSensorServIdx].characteristics?.count == 8 &&
            centralControl.connectedPeripheral?.services?[ROServIdx].characteristics?.count == 1) else {
            discoverDeviceCharacteristicFailedAlertEnable = true
            return false
        }
        return true
    }

    func writeTestInProgress(_ state : String) -> Void {
        if centralControl.connectedPeripheral != nil {
            if let testServIdx = (centralControl.connectedPeripheral?.services?.firstIndex(where: {$0.uuid == DeviceServices.TestStatusServiceUUID})) {
                if let testInProgressCharIdx = (centralControl.connectedPeripheral?.services?[testServIdx].characteristics?.firstIndex(where: {$0.uuid ==
                    DeviceServices.TestInProgressCharacteristicUUID})) {
                    if let char = centralControl.connectedPeripheral?.services?[testServIdx].characteristics?[testInProgressCharIdx] {
                        var value = Data()
                        if(state == "start") {
                            timerManager.initTestTimer()
```



```
                            value = Data([0x01])
                        }
                        else if (state == "stop") {
                            value = Data([0x00])
                        }
                        centralControl.connectedPeripheral?.writeValue(value, for: char, type: .withResponse)
                        testStatus[0].testInProgress = true
                        testChannelsUsed = TestState.rawDataToBoolArray(rawData: testStatus[0].testChannelsUsed)
                        numChannelsUsed = testChannelsUsed.filter{$0 == true}.count
                        do {
                            try saveDataToCD()
                        } catch {
                            ()
                        }
                    } else {
                        discoverDeviceCharacteristicFailedAlertEnable = true
                        return
                    }
                } else {
                    discoverDeviceServicesFailedAlertEnable = true
                    return
                }
            } else {
                discoverDeviceServicesFailedAlertEnable = true
                return
            }
        } else {
            deviceDisconnectedAlertEnable = true
            return
        }
    }

    func continueTestInProgress() {
        timerManager.initTestTimer()
    }

    func endTest() -> Void {
        timerManager.invalidate()
        writeTestInProgress("stop")
        testStatus[0].testInProgress = false
        do {
            try saveDataToCD()
        } catch {
            ()
        }
    }

    func cleanupTest() -> Void {
```

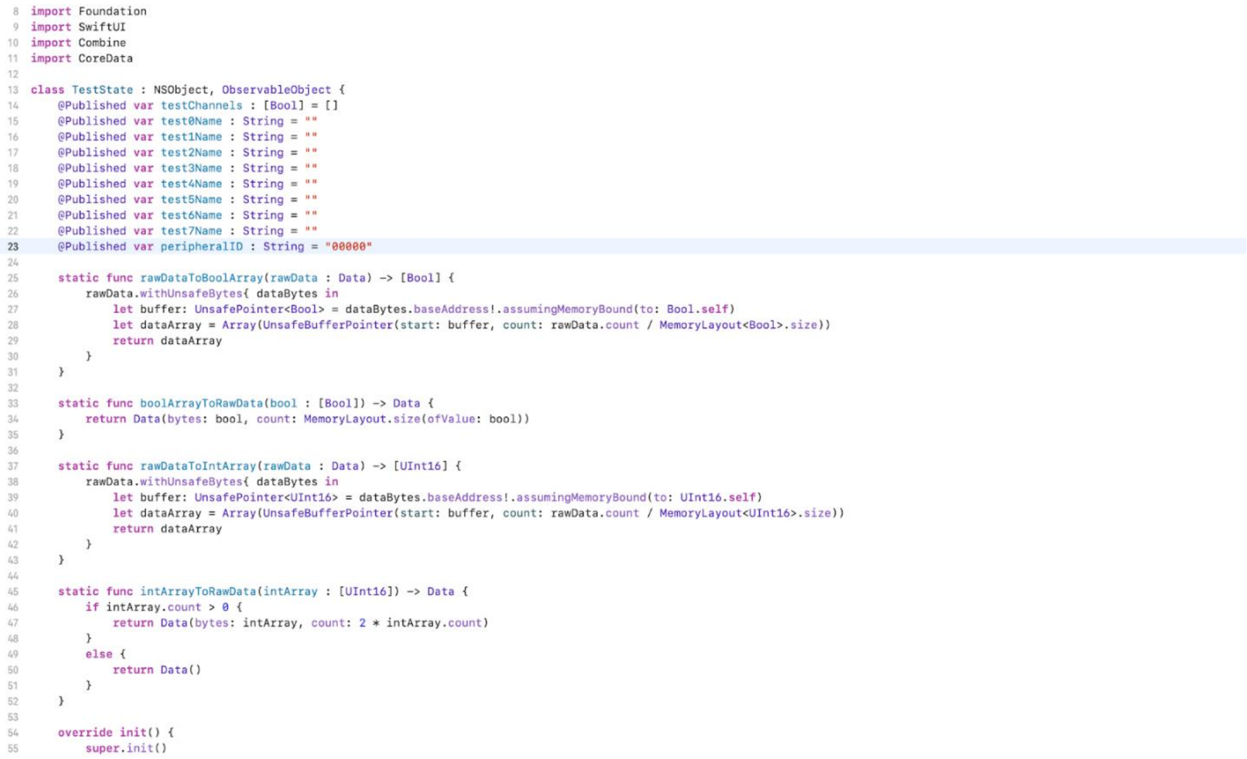


```swift
    func checkData() -> Void {
        DispatchQueue.main.asyncAfter(deadline: .now() + DispatchTimeInterval.seconds(timeBetweenReads - 5)) {
            testChannelsUsed = TestState.rawDataToBoolArray(rawData: testStatus[0].testChannelsUsed)
            numChannelsUsed = testChannelsUsed.filter{$0 == true}.count
            processData(channels: testChannelsUsed)
        }
    }

    func updateData() -> Void {
        DispatchQueue.main.asyncAfter(deadline: .now() + DispatchTimeInterval.seconds(timeBetweenReads - 5))  {
            var maskedRawData : [[UInt16]?] = Array(repeating: [], count: 8)
            previousRawData = rawData
            if let readingOut = centralControl.charCachedValue[DeviceServices.ROCharacteristicUUID] {
                deviceHeatingUp = readingOut[0] == 1 ? false : true
            }
            if let lastUsedChannel = testChannelsUsed.lastIndex(where: {$0 == true}) {
                charList.forEach { characteristic in
                    let str = characteristic.uuidString
                    let charChannel : Int = str[str.index(str.startIndex, offsetBy: 1)].wholeNumberValue! - 1
                    rawData[charChannel] = centralControl.charCachedValue[characteristic] ?? [0]
                    if(rawData[charChannel].count == 0) {
                        numEmptyReads += 1
                        if numEmptyReads >= 48 {
                            numEmptyReads = 0
                            readPlotDataEmptyAlertEnable = true
                            return
                        }
                        return
                    }
                    numEmptyReads = 0
                    if testChannelsUsed[charChannel] {
                        maskedRawData[charChannel] = rawData[charChannel]
                    }
                    if(testChannelsUsed != Array<Bool>(repeating: false, count: 8) && lastUsedChannel == charChannel && previousRawData != rawData && rawData[0].count > 1) {
                        data.insertValuesOf(chartData: maskedRawData)
                    }
                    if let firstUsedChannel = testChannelsUsed.firstIndex(where: {$0 == true}) {
                        testStatus[0].iteration = Int64(rawData[firstUsedChannel].count)
                        do {
                            try saveDataToCD()
                            try viewContext.save()
                        } catch {
                            saveToCoreDataFailedAlertEnable = true
                            return
                        }
                    }
                }
                if(testStatus[0].iteration >= numIterations) {
```

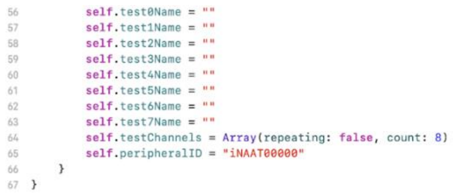


```swift
                if(testStatus[0].iteration >= numIterations) {
                    withResults(do: "show")
                    endTest()
                    resultsReady = true
                }
            }
        }
    }
}
```

TestState.swift

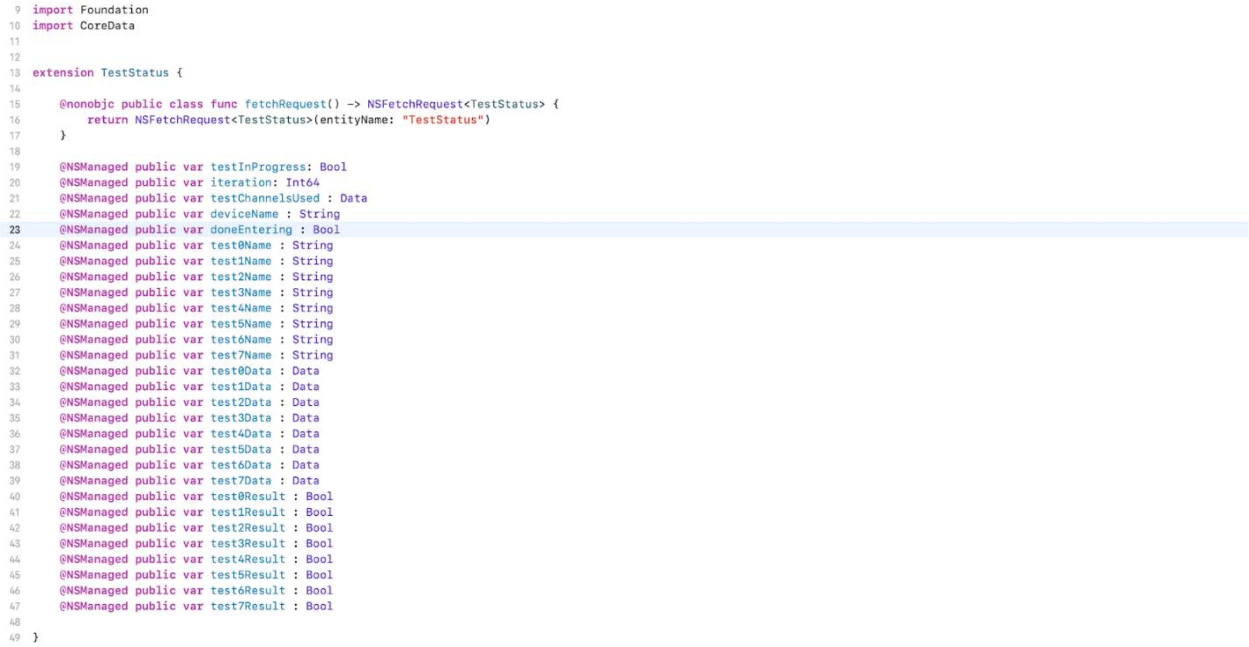


```swift
import Foundation
import SwiftUI
import Combine
import CoreData

class TestState : NSObject, ObservableObject {
    @Published var testChannels : [Bool] = []
    @Published var test0Name : String = ""
    @Published var test1Name : String = ""
    @Published var test2Name : String = ""
    @Published var test3Name : String = ""
    @Published var test4Name : String = ""
    @Published var test5Name : String = ""
    @Published var test6Name : String = ""
    @Published var test7Name : String = ""
    @Published var peripheralID : String = "00000"

    static func rawDataToBoolArray(rawData : Data) -> [Bool] {
        rawData.withUnsafeBytes{ dataBytes in
            let buffer: UnsafePointer<Bool> = dataBytes.baseAddress!.assumingMemoryBound(to: Bool.self)
            let dataArray = Array(UnsafeBufferPointer(start: buffer, count: rawData.count / MemoryLayout<Bool>.size))
            return dataArray
        }
    }

    static func boolArrayToRawData(bool : [Bool]) -> Data {
        return Data(bytes: bool, count: MemoryLayout.size(ofValue: bool))
    }

    static func rawDataToIntArray(rawData : Data) -> [UInt16] {
        rawData.withUnsafeBytes{ dataBytes in
            let buffer: UnsafePointer<UInt16> = dataBytes.baseAddress!.assumingMemoryBound(to: UInt16.self)
            let dataArray = Array(UnsafeBufferPointer(start: buffer, count: rawData.count / MemoryLayout<UInt16>.size))
            return dataArray
        }
    }

    static func intArrayToRawData(intArray : [UInt16]) -> Data {
        if intArray.count > 0 {
            return Data(bytes: intArray, count: 2 * intArray.count)
        }
        else {
            return Data()
        }
    }

    override init() {
        super.init()
```



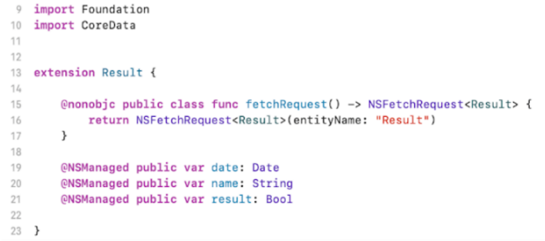
```swift
        self.test0Name = ""
        self.test1Name = ""
        self.test2Name = ""
        self.test3Name = ""
        self.test4Name = ""
        self.test5Name = ""
        self.test6Name = ""
        self.test7Name = ""
        self.testChannels = Array(repeating: false, count: 8)
        self.peripheralID = "iNAAT00000"
    }
}
```

TestStatus+CoreDataProperties.swift



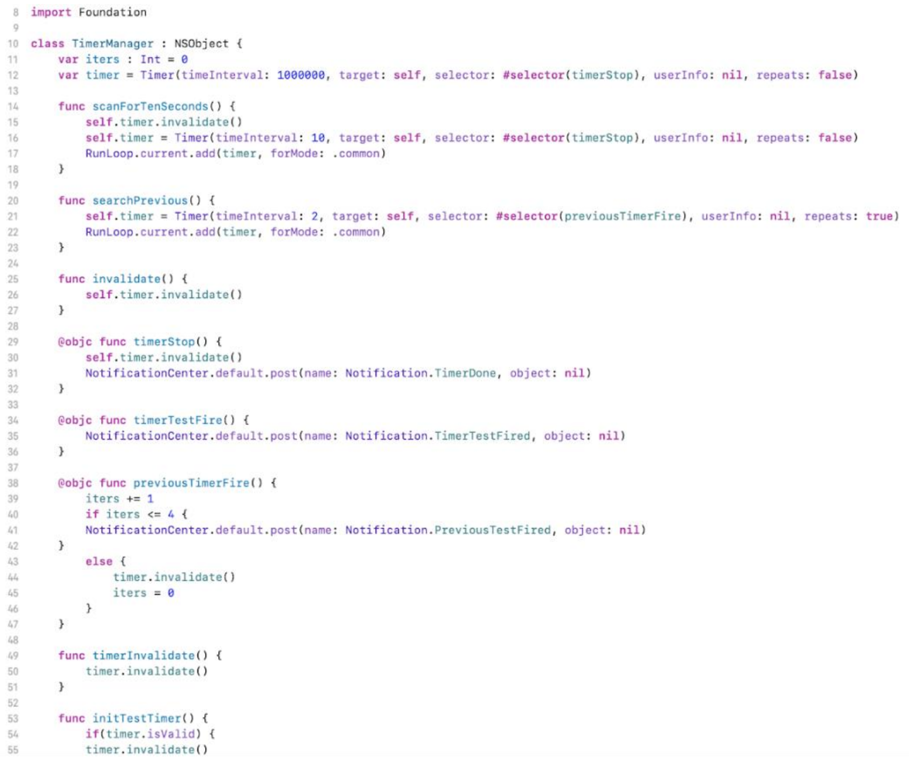
```swift
import Foundation
import CoreData


extension TestStatus {

    @nonobjc public class func fetchRequest() -> NSFetchRequest<TestStatus> {
        return NSFetchRequest<TestStatus>(entityName: "TestStatus")
    }

    @NSManaged public var testInProgress: Bool
    @NSManaged public var iteration: Int64
    @NSManaged public var testChannelsUsed : Data
    @NSManaged public var deviceName : String
    @NSManaged public var doneEntering : Bool
    @NSManaged public var test0Name : String
    @NSManaged public var test1Name : String
    @NSManaged public var test2Name : String
    @NSManaged public var test3Name : String
    @NSManaged public var test4Name : String
    @NSManaged public var test5Name : String
    @NSManaged public var test6Name : String
    @NSManaged public var test7Name : String
    @NSManaged public var test0Data : Data
    @NSManaged public var test1Data : Data
    @NSManaged public var test2Data : Data
    @NSManaged public var test3Data : Data
    @NSManaged public var test4Data : Data
    @NSManaged public var test5Data : Data
    @NSManaged public var test6Data : Data
    @NSManaged public var test7Data : Data
    @NSManaged public var test0Result : Bool
    @NSManaged public var test1Result : Bool
    @NSManaged public var test2Result : Bool
    @NSManaged public var test3Result : Bool
    @NSManaged public var test4Result : Bool
    @NSManaged public var test5Result : Bool
    @NSManaged public var test6Result : Bool
    @NSManaged public var test7Result : Bool

}
```

Result+CoreDataProperties.swift

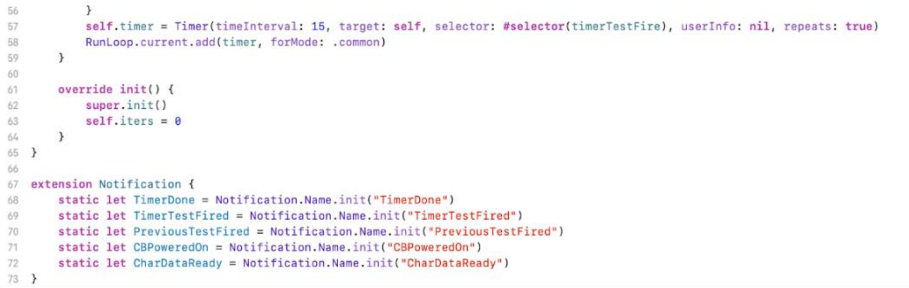


```swift
import Foundation
import CoreData


extension Result {

    @nonobjc public class func fetchRequest() -> NSFetchRequest<Result> {
        return NSFetchRequest<Result>(entityName: "Result")
    }

    @NSManaged public var date: Date
    @NSManaged public var name: String
    @NSManaged public var result: Bool

}
```

TimerManager.swift

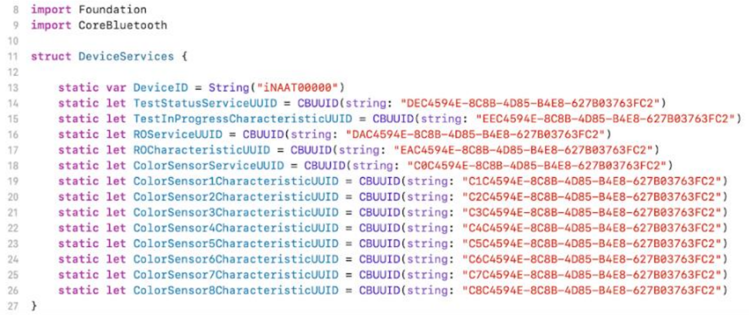


```swift
import Foundation

class TimerManager : NSObject {
    var iters : Int = 0
    var timer = Timer(timeInterval: 1000000, target: self, selector: #selector(timerStop), userInfo: nil, repeats: false)

    func scanForTenSeconds() {
        self.timer.invalidate()
        self.timer = Timer(timeInterval: 10, target: self, selector: #selector(timerStop), userInfo: nil, repeats: false)
        RunLoop.current.add(timer, forMode: .common)
    }

    func searchPrevious() {
        self.timer = Timer(timeInterval: 2, target: self, selector: #selector(previousTimerFire), userInfo: nil, repeats: true)
        RunLoop.current.add(timer, forMode: .common)
    }

    func invalidate() {
        self.timer.invalidate()
    }

    @objc func timerStop() {
        self.timer.invalidate()
        NotificationCenter.default.post(name: Notification.TimerDone, object: nil)
    }

    @objc func timerTestFire() {
        NotificationCenter.default.post(name: Notification.TimerTestFired, object: nil)
    }

    @objc func previousTimerFire() {
        iters += 1
        if iters <= 4 {
        NotificationCenter.default.post(name: Notification.PreviousTestFired, object: nil)
    }
        else {
            timer.invalidate()
            iters = 0
        }
    }

    func timerInvalidate() {
        timer.invalidate()
    }

    func initTestTimer() {
        if(timer.isValid) {
        timer.invalidate()
```



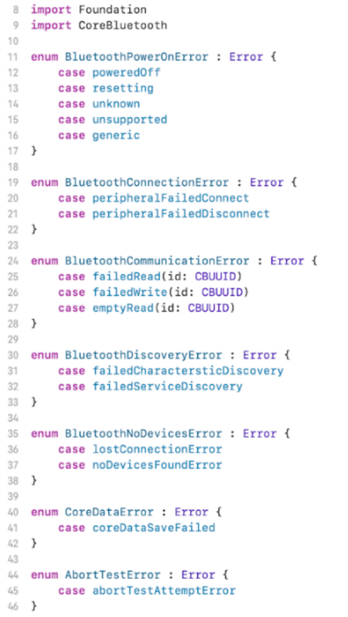
```swift
        }
        self.timer = Timer(timeInterval: 15, target: self, selector: #selector(timerTestFire), userInfo: nil, repeats: true)
        RunLoop.current.add(timer, forMode: .common)
    }

    override init() {
        super.init()
        self.iters = 0
    }
}

extension Notification {
    static let TimerDone = Notification.Name.init("TimerDone")
    static let TimerTestFired = Notification.Name.init("TimerTestFired")
    static let PreviousTestFired = Notification.Name.init("PreviousTestFired")
    static let CBPoweredOn = Notification.Name.init("CBPoweredOn")
    static let CharDataReady = Notification.Name.init("CharDataReady")
}
```

DeviceServices.swift



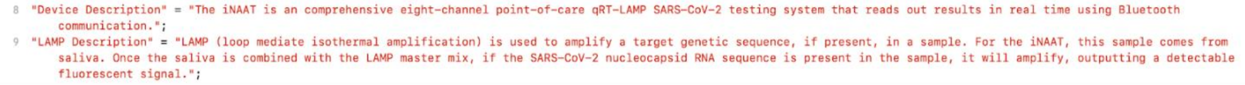
```swift
import Foundation
import CoreBluetooth

struct DeviceServices {

    static var DeviceID = String("iNAAT00000")
    static let TestStatusServiceUUID = CBUUID(string: "DEC4594E-8C8B-4D85-B4E8-627B03763FC2")
    static let TestInProgressCharacteristicUUID = CBUUID(string: "EEC4594E-8C8B-4D85-B4E8-627B03763FC2")
    static let ROServiceUUID = CBUUID(string: "DAC4594E-8C8B-4D85-B4E8-627B03763FC2")
    static let ROCharacteristicUUID = CBUUID(string: "EAC4594E-8C8B-4D85-B4E8-627B03763FC2")
    static let ColorSensorServiceUUID = CBUUID(string: "C0C4594E-8C8B-4D85-B4E8-627B03763FC2")
    static let ColorSensor1CharacteristicUUID = CBUUID(string: "C1C4594E-8C8B-4D85-B4E8-627B03763FC2")
    static let ColorSensor2CharacteristicUUID = CBUUID(string: "C2C4594E-8C8B-4D85-B4E8-627B03763FC2")
    static let ColorSensor3CharacteristicUUID = CBUUID(string: "C3C4594E-8C8B-4D85-B4E8-627B03763FC2")
    static let ColorSensor4CharacteristicUUID = CBUUID(string: "C4C4594E-8C8B-4D85-B4E8-627B03763FC2")
    static let ColorSensor5CharacteristicUUID = CBUUID(string: "C5C4594E-8C8B-4D85-B4E8-627B03763FC2")
    static let ColorSensor6CharacteristicUUID = CBUUID(string: "C6C4594E-8C8B-4D85-B4E8-627B03763FC2")
    static let ColorSensor7CharacteristicUUID = CBUUID(string: "C7C4594E-8C8B-4D85-B4E8-627B03763FC2")
    static let ColorSensor8CharacteristicUUID = CBUUID(string: "C8C4594E-8C8B-4D85-B4E8-627B03763FC2")
}
```

CustomErrors.swift



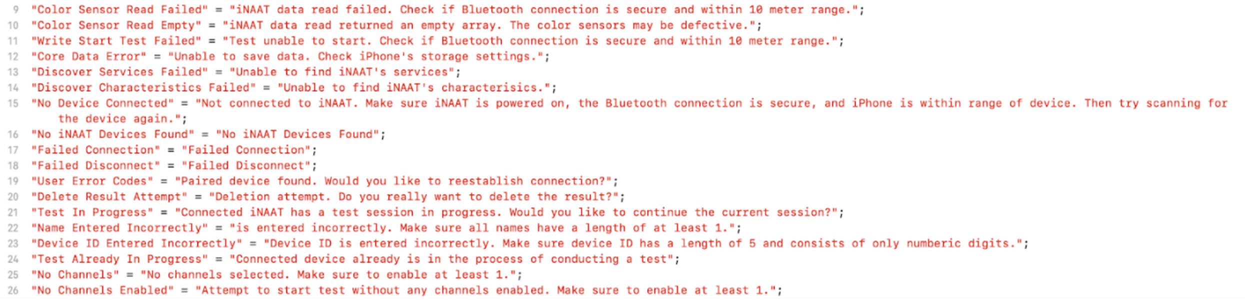
```swift
import Foundation
import CoreBluetooth

enum BluetoothPowerOnError : Error {
    case poweredOff
    case resetting
    case unknown
    case unsupported
    case generic
}

enum BluetoothConnectionError : Error {
    case peripheralFailedConnect
    case peripheralFailedDisconnect
}

enum BluetoothCommunicationError : Error {
    case failedRead(id: CBUUID)
    case failedWrite(id: CBUUID)
    case emptyRead(id: CBUUID)
}

enum BluetoothDiscoveryError : Error {
    case failedCharactersticDiscovery
    case failedServiceDiscovery
}

enum BluetoothNoDevicesError : Error {
    case lostConnectionError
    case noDevicesFoundError
}

enum CoreDataError : Error {
    case coreDataSaveFailed
}

enum AbortTestError : Error {
    case abortTestAttemptError
}
```

iNAAT App Description.strings



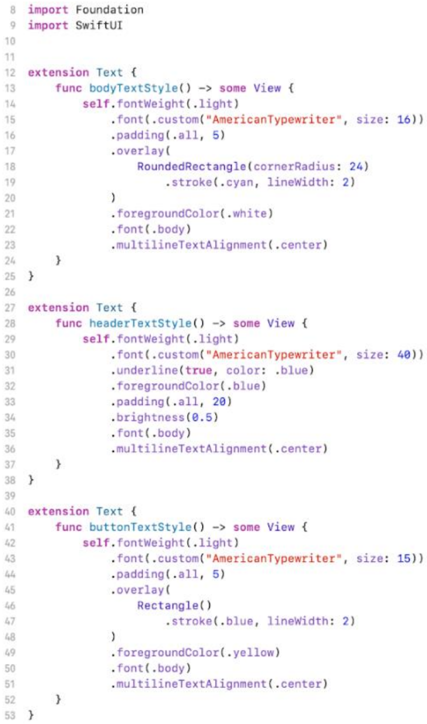
```
"Device Description" = "The iNAAT is an comprehensive eight-channel point-of-care qRT-LAMP SARS-CoV-2 testing system that reads out results in real time using Bluetooth
    communication.";
"LAMP Description" = "LAMP (loop mediate isothermal amplification) is used to amplify a target genetic sequence, if present, in a sample. For the iNAAT, this sample comes from
    saliva. Once the saliva is combined with the LAMP master mix, if the SARS-CoV-2 nucleocapsid RNA sequence is present in the sample, it will amplify, outputting a detectable
    fluorescent signal.";
```

User Error Codes.strings

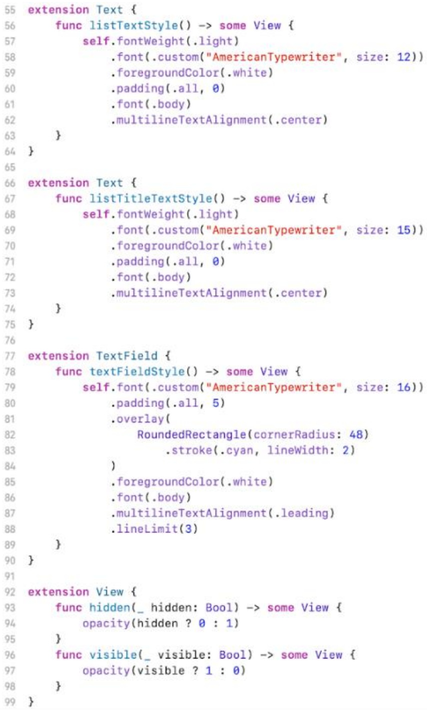


```
"Color Sensor Read Failed" = "iNAAT data read failed. Check if Bluetooth connection is secure and within 10 meter range.";
"Color Sensor Read Empty" = "iNAAT data read returned an empty array. The color sensors may be defective.";
"Write Start Test Failed" = "Test unable to start. Check if Bluetooth connection is secure and within 10 meter range.";
"Core Data Error" = "Unable to save data. Check iPhone's storage settings.";
"Discover Services Failed" = "Unable to find iNAAT's services";
"Discover Characteristics Failed" = "Unable to find iNAAT's characterisics.";
"No Device Connected" = "Not connected to iNAAT. Make sure iNAAT is powered on, the Bluetooth connection is secure, and iPhone is within range of device. Then try scanning for
    the device again.";
"No iNAAT Devices Found" = "No iNAAT Devices Found";
"Failed Connection" = "Failed Connection";
"Failed Disconnect" = "Failed Disconnect";
"User Error Codes" = "Paired device found. Would you like to reestablish connection?";
"Delete Result Attempt" = "Deletion attempt. Do you really want to delete the result?";
"Test In Progress" = "Connected iNAAT has a test session in progress. Would you like to continue the current session?";
"Name Entered Incorrectly" = "is entered incorrectly. Make sure all names have a length of at least 1.";
"Device ID Entered Incorrectly" = "Device ID is entered incorrectly. Make sure device ID has a length of 5 and consists of only numberic digits.";
"Test Already In Progress" = "Connected device already is in the process of conducting a test";
"No Channels" = "No channels selected. Make sure to enable at least 1.";
"No Channels Enabled" = "Attempt to start test without any channels enabled. Make sure to enable at least 1.";
```

AppTheme.swift



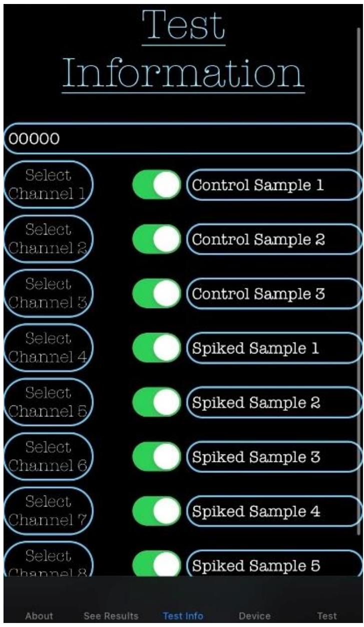
```swift
import Foundation
import SwiftUI


extension Text {
    func bodyTextStyle() -> some View {
        self.fontWeight(.light)
            .font(.custom("AmericanTypewriter", size: 16))
            .padding(.all, 5)
            .overlay(
                RoundedRectangle(cornerRadius: 24)
                    .stroke(.cyan, lineWidth: 2)
            )
            .foregroundColor(.white)
            .font(.body)
            .multilineTextAlignment(.center)
    }
}

extension Text {
    func headerTextStyle() -> some View {
        self.fontWeight(.light)
            .font(.custom("AmericanTypewriter", size: 40))
            .underline(true, color: .blue)
            .foregroundColor(.blue)
            .padding(.all, 20)
            .brightness(0.5)
            .font(.body)
            .multilineTextAlignment(.center)
    }
}

extension Text {
    func buttonTextStyle() -> some View {
        self.fontWeight(.light)
            .font(.custom("AmericanTypewriter", size: 15))
            .padding(.all, 5)
            .overlay(
                Rectangle()
                    .stroke(.blue, lineWidth: 2)
            )
            .foregroundColor(.yellow)
            .font(.body)
            .multilineTextAlignment(.center)
    }
}
```



```swift
extension Text {
    func listTextStyle() -> some View {
        self.fontWeight(.light)
            .font(.custom("AmericanTypewriter", size: 12))
            .foregroundColor(.white)
            .padding(.all, 0)
            .font(.body)
            .multilineTextAlignment(.center)
    }
}

extension Text {
    func listTitleTextStyle() -> some View {
        self.fontWeight(.light)
            .font(.custom("AmericanTypewriter", size: 15))
            .foregroundColor(.white)
            .padding(.all, 0)
            .font(.body)
            .multilineTextAlignment(.center)
    }
}

extension TextField {
    func textFieldStyle() -> some View {
        self.font(.custom("AmericanTypewriter", size: 16))
            .padding(.all, 5)
            .overlay(
                RoundedRectangle(cornerRadius: 48)
                    .stroke(.cyan, lineWidth: 2)
            )
            .foregroundColor(.white)
            .font(.body)
            .multilineTextAlignment(.leading)
            .lineLimit(3)
    }
}

extension View {
    func hidden(_ hidden: Bool) -> some View {
        opacity(hidden ? 0 : 1)
    }
    func visible(_ visible: Bool) -> some View {
        opacity(visible ? 1 : 0)
    }
}
```

Appendix D

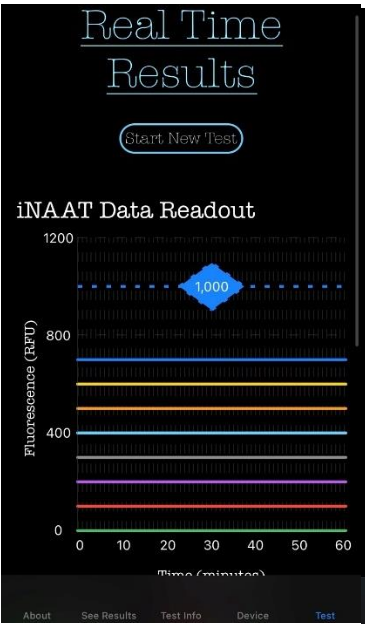
iOS App Screenshots



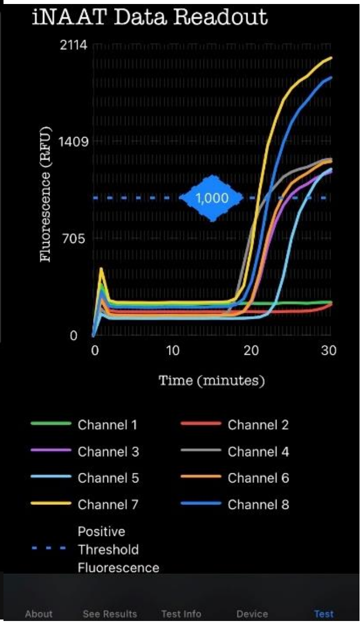
Test Information

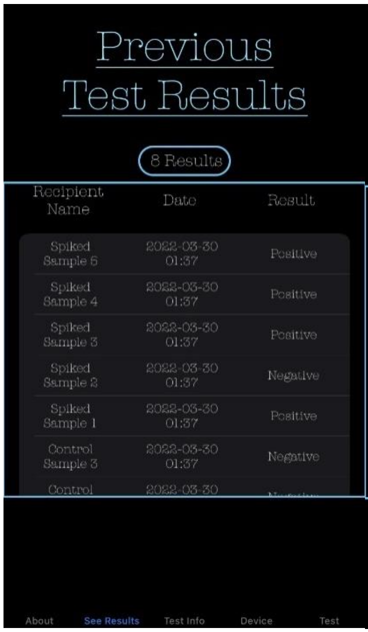
00000

Select Channel 1	●	Control Sample 1
Select Channel 2	●	Control Sample 2
Select Channel 3	●	Control Sample 3
Select Channel 4	●	Spiked Sample 1
Select Channel 5	●	Spiked Sample 2
Select Channel 6	●	Spiked Sample 3
Select Channel 7	●	Spiked Sample 4
Select Channel 8	●	Spiked Sample 5



Real Time Results

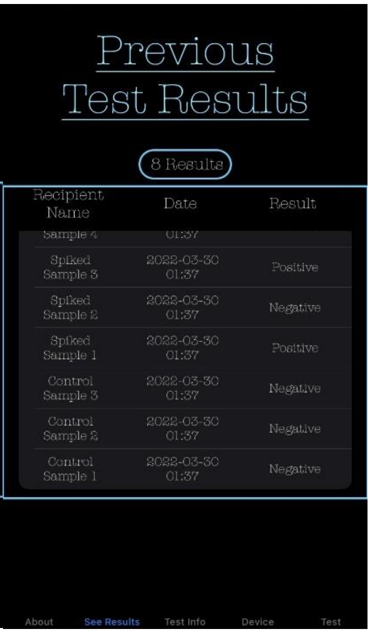




Previous Test Results

8 Results

Recipient Name	Date	Result
Spiked Sample 6	2022-03-30 01:37	Positive
Spiked Sample 4	2022-03-30 01:37	Positive
Spiked Sample 3	2022-03-30 01:37	Positive
Spiked Sample 2	2022-03-30 01:37	Negative
Spiked Sample 1	2022-03-30 01:37	Positive
Control Sample 3	2022-03-30 01:37	Negative
Control	2022-03-30	



Previous Test Results

8 Results

Recipient Name	Date	Result
Sample 4	01:37	
Spiked Sample 3	2022-03-30 01:37	Positive
Spiked Sample 2	2022-03-30 01:37	Negative
Spiked Sample 1	2022-03-30 01:37	Positive
Control Sample 3	2022-03-30 01:37	Negative
Control Sample 2	2022-03-30 01:37	Negative
Control Sample 1	2022-03-30 01:37	Negative

Appendix E

Contributors to iNAAT Design

The design of the iNAAT was a years-long process that predates my acceptance into GuanLabs. It was truly a team effort and required the contribution of many individuals and tools. This section details a list of individuals and code repositories that assisted in the design of the iNAAT. URLs to repositories will be provided here, but a full citation can be found in the bibliography.

First, Dr. Guan spearheaded the project. With expertise in the design of point-of-care tests, he saw the opportunity to apply the principles of POC to the ongoing COVID-19 pandemic.

Graduate students at GuanLabs, mainly Zifan Tang, performed a validation and comparison of the N and E gene primers used in the RT-LAMP assay.

Next, GuanLabs researcher Aneesh Kshirsager did the majority of the Eagle PCB design, all the mechanical design, and all of the component selection. Additionally, he wrote much the Raspberry Pi code that controls the heat control MOSFET, LEDs and the original TCS34725 color sensors.

Without BlueZ (https://github.com/bluez), the Bluetooth programming would have to be done with low-level socket programming. It was integral to the speedy validation real-time readout feature of the iNAAT.

SwiftUICharts (https://github.com/willdale/SwiftUICharts) provided a template to display multi-line chart data. The repository contains demos of each chart type and was easily adaptable to the needs of the iNAAT system.

Adafruit’s TCA9548A driver (https://github.com/adafruit/Adafruit_CircuitPython_TCA9548A) was used and modified to facilitate the multiplexing between the 8 test channels.

Lastly, Apple provided many frameworks (Core Bluetooth, Core Data, and SwiftUI) that allow rapid prototyping of smartphone apps.

My contributions to the iNAAT are in the software design. I demonstrated that the VEML3328 color sensors could be used as an alternative to the TCS34725 and added them to the PCB. Then, I organized Aneesh’s code into a coordinated crash-proof program. I also set up the Bluetooth LE GATT server on the Raspberry Pi. Lastly, I coded the iOS app in its entirety.

BIBLIOGRAPHY

[1] David M. Cutler, "The COVID-19 Pandemic and the $16 Trillion Virus," *JAMA,* p. 1562, 2020.

[2] WorldOmeter, "Coronavirus Death Toll," 4 April 2022. [Online]. Available: https://www.worldometers.info/coronavirus/coronavirus-death-toll/.

[3] CDC, "Overview of Testing for SARS-CoV-2, the virus that causes COVID-19," 11 February 2022. [Online]. Available: https://www.cdc.gov/coronavirus/2019-ncov/hcp/testing-overview.html.

[4] V. Bauerlein, "Covid-19 Testing Crunch Hampers Efforts to Curb Omicron Variant Surge," Wall Street Journal, 30 December 2021. [Online]. Available: https://www.wsj.com/articles/covid-19-testing-crunch-hampers-efforts-to-curb-omicron-variant-surge-11640860201?mod=article_inline.

[5] W. M. Freeman, "Quantitative RT-PCR: Pitfalls and Potenti," *BioTechniques,* 1999.

[6] A. Ferrini, "An Introduction to PCR," Technology Networks, 17 March 2021. [Online]. Available: https://www.technologynetworks.com/genomics/articles/an-introduction-to-pcr-345445.

[7] S. Ryding, "What is RT-LAMP Technology?," News Medical Life Sciences, 10 March 2021. [Online]. Available: https://www.news-medical.net/health/What-is-RT-LAMP-Technology.aspx#:~:text=Whereas%20a%20traditional%20PCR%20can,than%20normal%20RT%2DPCR%20assays..

[8] T. Notomi, "Loop-mediated isothermal amplification of DNA," *Nucleic Acids Research.*

[9] J. P. Broughton, "CRISPR–Cas12-based detection of SARS-CoV-2," *nature biotechnology,* 2020.

[10] ZifanTang, "Rapid detection of novel coronavirus SARS-CoV-2 by RT-LAMP coupled solid-state nanopores," *Biosensors and Bioelectronics,* vol. 197, 2022.

[11] Vishay Semiconductors, "RGBCIR Color Sensor With I2C Interface," [Online]. Available: https://www.vishay.com/docs/84968/veml3328.pdf.

[12] Caddock Electronics, "MP725 Surface Mount Power Film Resistors," [Online]. Available: http://www.caddock.com/Online_catalog/Mrktg_Lit/MP725.pdf.

[13] Raspberry Pi Foundation, "Raspberry Pi Zero W," [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-zero-w/.

[14] Arduino, "Arduino UNO," [Online]. Available: https://www.arduino.cc/en/main/arduinoBoardUno.

[15] B. Project, "BlueZ Official Linux Bluetooth Protocol Stack," [Online]. Available: http://www.bluez.org/.

[16] Edvotek, "Edvotek EdvoCycler Jr. PCR," [Online]. Available: https://www.schoolspecialty.com/edvotek-edvocycler-jr-2039705?utm_source=google&utm_medium=shopping&utm_campaign=13806759776&product_id=2039705&ad_group_id=126163195802&feed_item_id&target_id=pla-1367238450906&gclid=CjwKCAjwopWSBhB6EiwAjxmqDT0c4zBVZpzZVZC.

[17] SPWIndustrial, "96 wells Real-Time PCR System MSLPCR30 virus PCR test machine," [Online]. Available: https://spwindustrial.com/96-wells-real-time-pcr-system-mslpcr30-virus-pcr-test-machine/.

[18] MikroElektronika, "Click Boards Sensors Optical Color 10 Click Front," [Online]. Available: https://www.mikroe.com/color-10-click.

[19] DigiKey, "MP725-1.00-1%," [Online]. Available: https://www.digikey.com/en/products/detail/caddock-electronics-inc/MP725-1-00-1/2182082.

[20] Adafruit, "TCA9548A I2C Multiplexer," [Online]. Available: https://www.adafruit.com/product/2717?gclid=CjwKCAjwi6WSBhA-EiwA6Niok63qBimaLwCG-RXhvBR8BNqHkpVIZtD9b-6_K56DwFMVVykgN23P1RoCPJEQAvD_BwE.

[21] bluez, "bluez," [Online]. Available: https://github.com/bluez/bluez.

[22] Apple, "State and Data Flow," [Online]. Available: https://developer.apple.com/documentation/swiftui/state-and-data-flow.

[23] V. Bulavin, "SwiftUI View Lifecycle," 18 November 2020. [Online]. Available: https://www.vadimbulavin.com/swiftui-view-lifecycle/.

[24] Apple, "Core Bluetooth," [Online]. Available: https://developer.apple.com/documentation/corebluetooth.

[25] Apple, "Core Data Programming Guide," [Online]. Available: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CoreData/index.html.

[26] "New Rutgers Saliva Test for Coronavirus Gets FDA Approval," Rutgers Today, 13 April 2020. [Online]. Available: https://www.rutgers.edu/news/new-rutgers-saliva-test-coronavirus-gets-fda-approval.

[27] Life Technologies, "SYTO® Green-Fluorescent Nucleic Acid Stains," [Online]. Available: https://www.thermofisher.com/document-connect/document-connect.html?url=https%3A%2F%2Fassets.thermofisher.com%2FTFS-Assets%2FLSG%2Fmanuals%2Fmp07572.pdf.

ACADEMIC VITA

DEAN DEROSA

Summary

Senior Electrical Engineering student with research experience in biomedical device design and extracurricular project experience

Education

The Pennsylvania State University- Schreyer Honors College Graduation: May 2022
Bachelor of Science in Electrical Engineering, Minor in Computer Engineering

Professional Experience

Research Assistant- GuanLab - State College, PA 08/2020 to Present

- Helping design the hardware and software of 8-channel PCR device with Bluetooth-connected iOS app to display results in real time
- Writing an honors thesis on the device operation and design process for department approval

EDG Intern- MathWorks- Natick, MA 05/2021 to 08/2021

- Worked with Controls Quality Engineering team to write unit tests for new software components
- Wrote performance tests for algorithms in Control Systems toolbox

Deputy Outreach Committee Chair- IEEE PSU, State College, PA 10/2019 to 03/2020

- Traveled to local schools to demonstrate student-built devices to middle-school aged children
- Began preparation to organize robotics day, a children’s robotics competition hosted by IEEE

Project Experience

Advanced Vehicle Team, Year 1 08/2021 to Present

- Working towards designing an object detection system for use in the SAE AutoDrive II challenge
- Selected camera and radar models to use, developed camera synchronization system, and assisted in organizing ROS architecture

PCB Design for Use in Course 01/2022 to Present

- Designing a PCB in Eagle with a motor driver, accelerometer, IR sensors, and LEDs for use in an undergraduate embedded systems course.

Bluetooth Programming 01/2022 to Present

- Creating a GATT server on a Raspberry Pi using in C using BlueZ

Work History

Server - Tavola Restaurant & Bar- Springfield, PA 05/2019 to 08/2020
Food Runner- Allen Street Grill- State College, PA 11/2018 to 03/2020
Swim Lesson Instructor- Healthplex Sports Club- Springfield, PA 04/2018 to 01/2020
Head Swim Lesson Instructor- Swarthmore Swim Club – Swarthmore, PA 04/2015 to 08/2018