Design Requirements

## Design Overview

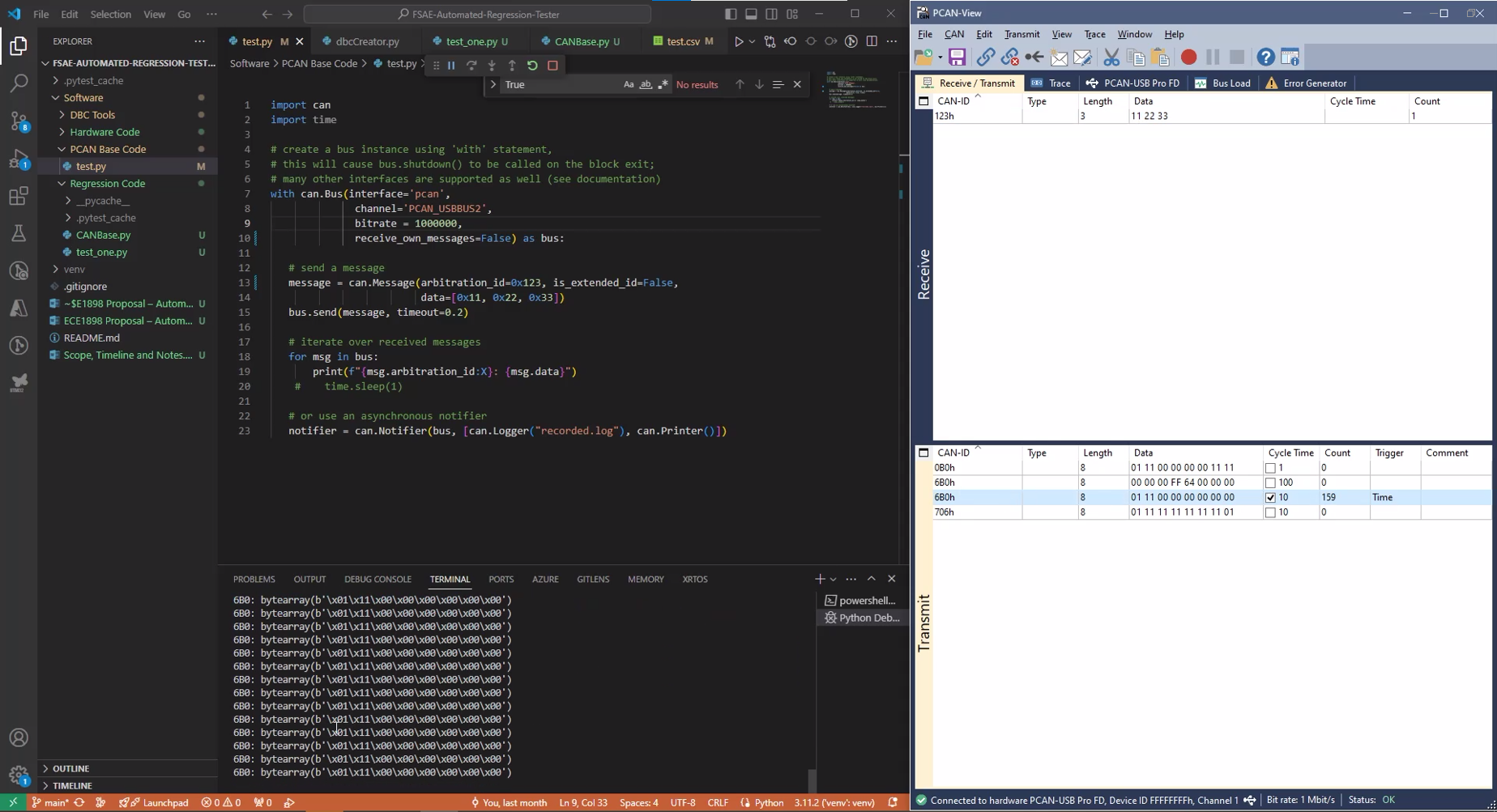
### CAN Communications

The CAN communication was don’t using a Peak System PCAN module. This module connects to the computer and is able to both send and receive CAN messages. Integration of the PCAN into the Automated Testing System was done using the python can library.

#### Design Revision 1

In order for this to be possible I started off with some basic python code to purely send CAN messages using the PCAN. In order to do this, I connected the two CAN channels on the PCAN together and read the message that I was sending on PCAN View, which is a GUI developed to manually send CAN messages and view the messages being received on the CAN bus. With this initial revision code I also decided to write code to be able to read the CAN messages with python and print them out to the terminal.

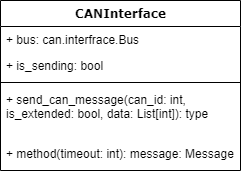
The 1st revision code created a CAN message object using the *can.Message()* function, in this case I used a random CAN ID and data frame just for testing purposes when making the message. I then sent the CAN message onto the bus using *can.Send().* For receiving CAN messages I used the *can.Notifier()* function which created a asynchronous thread that would wait for CAN messages to arrive and then print them out onto the terminal window.



Link to video

#### Design Revision 2

Once I had confirmed that I was able to send and receive CAN messages I went about writing a class for the CAN functionality so I could later integrate it easier into the larger Automated Regression Testing Software.



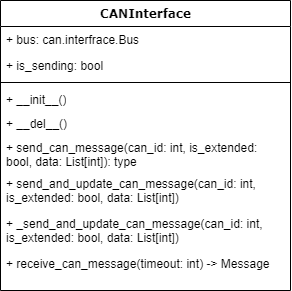
The above image is the initial CANInterface class diagram. It was very basic allowing us to create an object that holds the Bus interface that allows us to communicate with the PCAN hardware and two functions that could be called to send a CAN message and to receive the latest message on the CAN bus.

A computer screen shot of a program

Description automatically generated  
*Initial CANInterface class code*

#### Design Revision 3

While developing the basic automated testing code using PyTest I realized that that it would be much ore user friendly for the end user to have a function they can call to send specific CAN messages continuously at a predetermined rate. I then went about implementing a function in the CANInterface class that would open a thread to be able to send a CAN message in the background without the user having to intervene.



The above image is the new CANInterface class diagram. There are two new functions in it; send\_and\_update\_can\_message and \_ send\_and\_update\_can\_message. The second one is meant to be a private function that handles the actual sending on the can message and the timing between messages. The first one is meant to be the public function that handles starting the thread to allow the messages to be send in the background.

A screen shot of a computer program

Description automatically generated  
*CAN message send code to continuously send messages*

#### Design Revision 4

I then went about implementing further functionality in order to be able to send different CAN messages at a predetermined rate automatically. This was to improve user experience by allowing them to add the message to a list and then calling a function that would start a thread to send the messages. The functions use a thread lock in order to be able to write and read to a common list that holds the CAN message objects.

A screenshot of a computer program

Description automatically generated  
  
A screenshot of a message

Description automatically generated

The above images show the new CANInterface and MessageData class diagram. The MessageData class acts as a struct to hold the cantools Message object as well as data that corresponds to the Message object. This gives us an easy way to access the CAN Message object as well as the data for the message, thus making it more organized and easier to store both the Message information and the payload in in message\_list\_100Hz list. The message\_list\_100Hz is then read in the \_send\_and\_update\_can\_message\_100hz function that sends the CAN messages on the bus. We also have an add\_message\_100Hz function that allows the user to add a message to be sent repetitively on the bus.

A computer screen with text and images

Description automatically generated  
*CAN message send code to continuously send messages from message\_list\_100Hz*

*A computer screen with colorful text

Description automatically generated  
Code to add MessageData objects to the message\_list\_100Hz*

*A screenshot of a computer program

Description automatically generated  
Code for the MessageData class, it holds the Message object and the data for the object*

### Analog Fault Injection Hardware

#### Design Concept Revision 1

The initial design concept for the Analog Fault Injection Hardware included using an Arduino UNO to simulate sensors that would be connected to the Vehicle Controller. The hardware would take in a UART signal from the computer running the Automated Regression Testing Suite and use the PWM output pins on the ATMEGA328P to send an analog voltage to the sensor input pins on the vehicle controller. The diagram below shows a high-level overview of the system.

#### Initial Design Verification

#### Functional Diagram