AN134

16-125289

Serial Sniffer

Introduction

In telecommunication and data transmission, serial communication is the process of sending data one bit at a time, sequentially, over a communication channel or computer bus. In certain instances, one may desire to see the messages going to and from the serial master and the drive. The following is a tested method of sniffing the serial port to create a decoded communications log of serial binary and ASCII commands.

Hardware Requirements

Qty	Description
1	B&B 9PCDT RS-232 9-Pin Data Tap https://buy.advantech-bb.com/Serial/Port-Combiners-and-Splitters/model-BB-9PCDT.htm
3	DB9 Female to RJ11 Modular Adapter
2	USB to RS232|RJ11 Plug https://elestream.com/product/usb-to-serial-adapter-rj11-plug-for-copley-drives/
2	DB9 Male to DB9 Male Gender Changer
1	Telephone Cable (RJ11)

B&B Data Tap

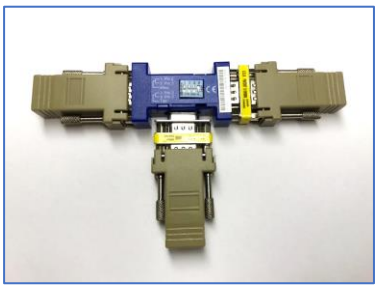
USB to RS232|RJ11 Plug

DB9 Male to Male Changer

DB9 Female to RJ11 Adapter

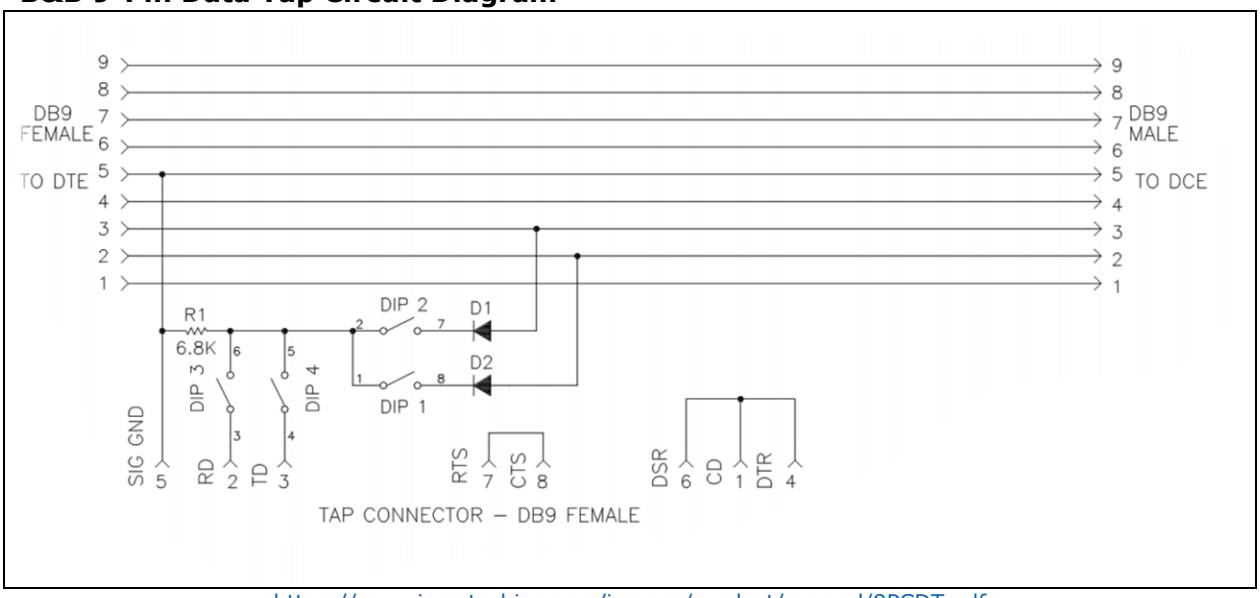
Telephone Cable (RJ11)

Final Product



The image above labeled "Final Product" shows all the hardware connected without the cabling attached. Alternatively, two of the DB9 Female to RJ11 Modular Adapters and the two DB9 Male to Male Gender Changers can be replaced by two DB9 Male to RJ11 Modular Adapters.

B&B 9-Pin Data Tap Circuit Diagram



https://www.icomtechinc.com/images/product/manual/9PCDT.pdf

In the above circuit diagram, there are 4 switches in total in the splitter (DIP 1, 2, 3, 4). To only capture the Receive (Rx) signals sent by the drive, turn ON switches DIP1 and DIP3. To only capture the Transmit (Tx) signals sent to the drive, turn ON switches DIP2 and DIP3. To sniff both Rx and Tx signals at the same time, turn on DIP1, 2, and 3. Because both signals are received on the same input, the Tx and Rx messages are grouped together on the same packet. After separating the messages, some timestamps must be artificially created since multiple messages are received at the same time on the same packet.

Software Requirements

Python version 3.5.2 or newer along with the python serial module "Pyserial" must be installed. In order to follow the steps provided in this document, Python must be added to the Windows Path to run as an executable and a tool used to install and manage python called "Pip" must be installed.

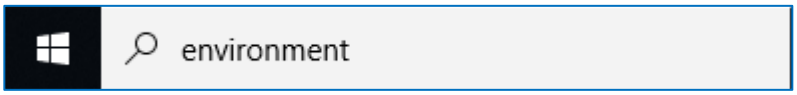
First download and install the latest version of Python from the web.

Link to Python 3.7.4: https://www.python.org/downloads/

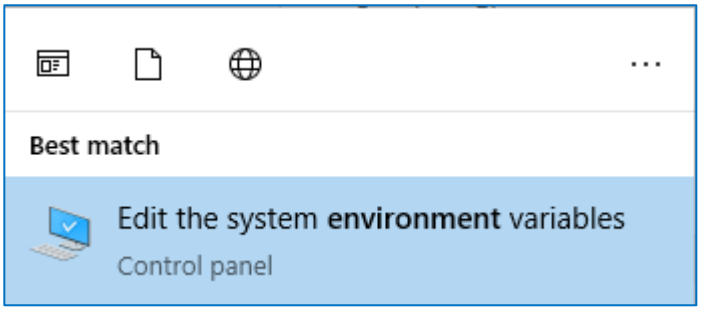
The location of the newly installed python37 folder was changed from its default location used by the installer "Windows(C:)\Users\<username>\Appdata\local\programs\Python\Python37" to a new, shorter location "Windows(C:) > python37". To move the location of the folder, navigate to its default location used by the installer and "cut and paste" the folder in the Windows(C:) folder.

How to add Python to Windows Path

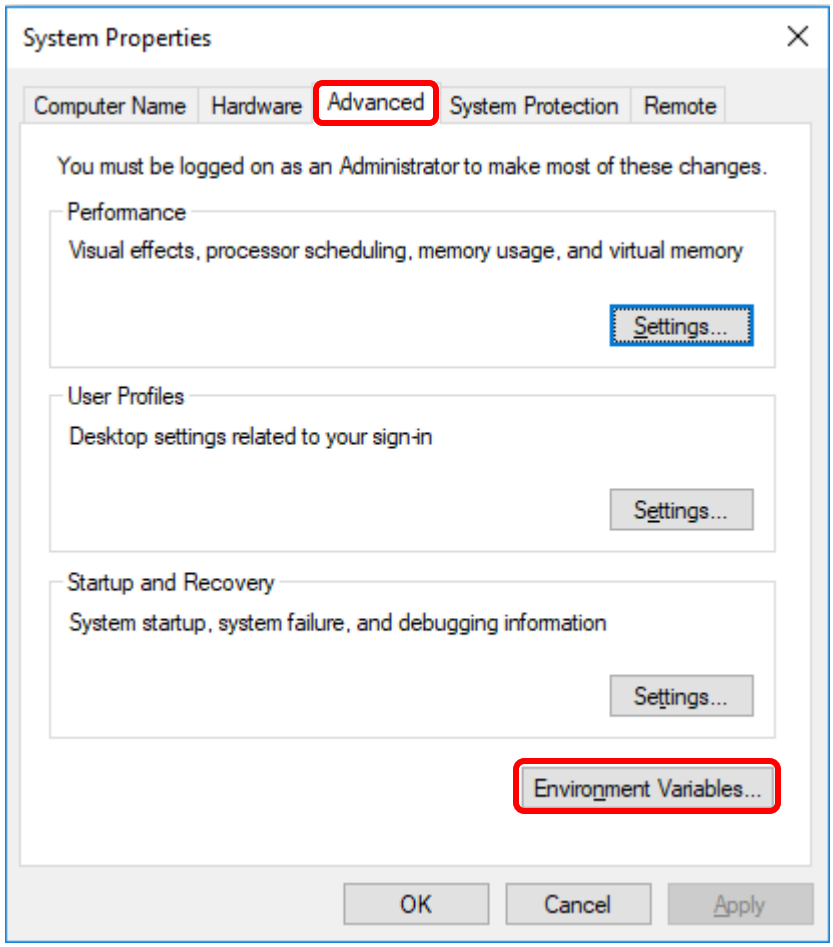
Type "environment" in the search bar on the lower left corner.



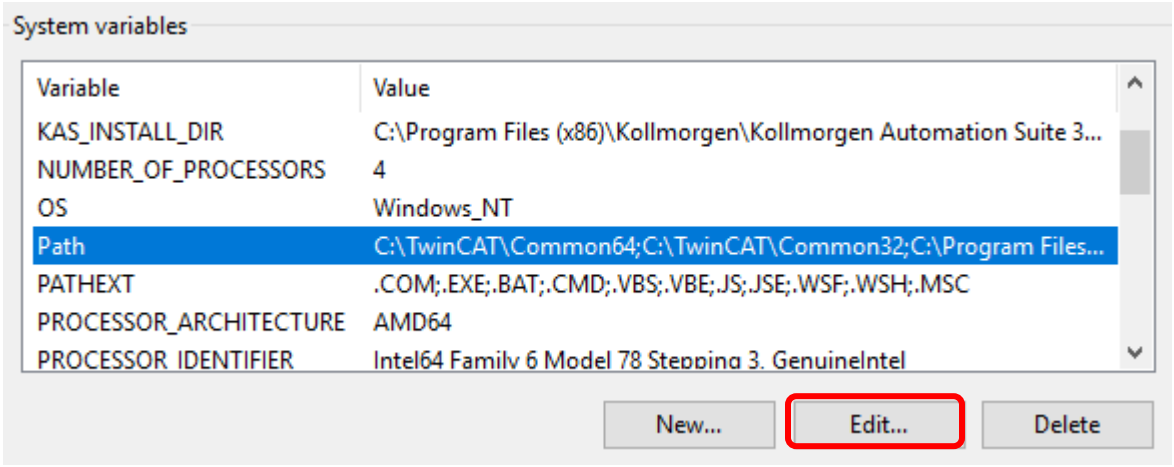
Select the "Edit the system environment variables" button that appears. The same button is in the Control Panel.



Next, select "Environment Variables".



Select the Path under System variables shown below and select Edit.



Select the "New" Button and type the file location of the Python37 folder shown below.

Click OK to return to the Environment Variables Menu. Click OK again to return to the System Properties Menu. Click Apply and then OK. Python has been successfully added to the Windows Path.

Confirm that Python has been added to your system path. Open the command prompt and type "python" and press the Enter Key. If the installed version of python is not displayed, try "python37" and press Enter.

```
C:\Users\aredamonti>python37
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Run the following command to exit Python 3.7: "exit()"

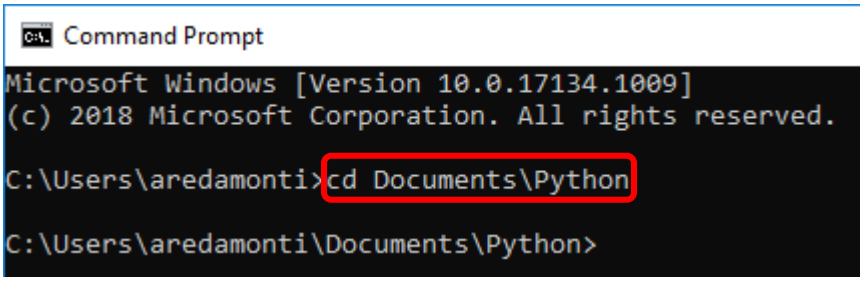
How to install Pip

Please refer to the following website as a reference: https://www.liquidweb.com/kb/install-pip-windows/

Download the file get-pip.py to a folder on your computer. Open the command prompt and navigate to the folder containing get-pip.py. To navigate to the folder, use the change directory command "cd" shown below.

The get-pip.py file was saved in "C:\Users\aredamonti\Documents\Python\get-pip.py".

Run the command: "cd Documents\Python"



The get-pip.py file can now be accessed via the command prompt. Run the command: "python37 get-pip.py"

Pip is now installed.

How to install the Pyserial Module

Run the following in the command prompt to install the python serial module:

"python37 -m pip install pyserial"

```
C:\Users\aredamonti>python37 -m pip install pyserial
Requirement already satisfied: pyserial in c:\python37\lib\site-packages (3.4)
```

The python serial module is now installed.

Create Serial_Sniffer.py

Serial_Sniffer.py is a script located at the end of this applications note. Copy and paste the script into a text editor and save it as a .py file (python file extension). Notepad++ is the recommended script editor application. Download Notepad++ using the following link: https://notepad-plus-plus.org/downloads/

Serial_Sniffer.py is used for parsing serial binary and ASCII messages. It is a python script (not executable) and can be run on Linux or Windows systems. It was saved in a folder named "Serial_Sniffer_Folder" located on the Desktop. Be sure that the file has been given permission to allow executing the file as a program. Right click on the file and select properties. For Windows Systems, permissions access is found in the Security Tab, and for Linux Systems there is a separate tab for permissions within the file properties.

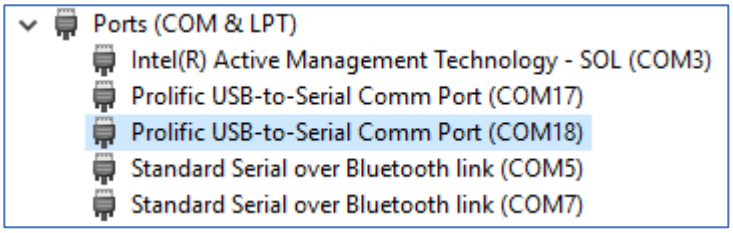
Setting Up

Open the Serial_Sniffer.py file and edit lines 7 and 13 to match name of the port associated with the Data Tap.

```
ser = serial.Serial('COM18', 115200, timeout = 2);
```

```
COM = "COM18"
```

For Windows Systems, the port names are "COM" followed by the port number. To find the port number, use the Device Manager.



For Linux Systems, the port name will be /dev/ttyUSB followed by the port number. See the format below.

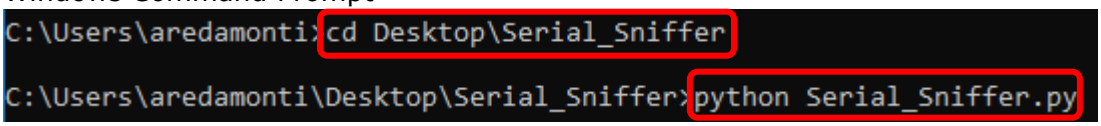
```
ser = serial.Serial('/dev/ttyUSB0', 115200, timeout = 2);
```

In line 7, enter in the correct baud rate and timeout value in seconds. The defaults for these are 115200 baud and 2 seconds. See the following link for more details:
https://pyserial.readthedocs.io/en/latest/pyserial_api.html

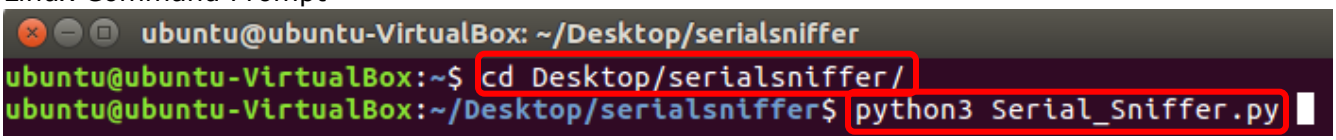
Open the command prompt. Navigate to the Serial_Sniffer_Folder on the Desktop. To navigate to the folder, run "cd Desktop\Serial_Sniffer_Folder".

Next, run the following: "python37 Serial_Sniffer.py" to begin logging data.

Windows Command Prompt



Linux Command Prompt



The raw, unformatted data will appear in the command prompt.

To stop recording data, perform a keyboard interrupt by holding "Ctrl + Shift + C" on the keyboard.

Close the command prompt and open the Serial_Sniffer_Folder on the Desktop. There should be two new text files created in the folder.

The unformatted data that appeared on the command prompt can be found in the text file named "data_raw".

The formatted data can be found in the text file named "data_new".

Below is an example of the raw data found in data_raw.txt

40 10:34:31.828 ['0x00', '0xfb', '0x01', '0x0c', '0x00', '0xac', '0x00', '0xa8', '0x02', '0x00', '0x00', '0x00', '0xf0', '0x00']

41 10:34:31.832 ['0x00', '0xf3', '0x01', '0x0c', '0x00', '0xa4', '0x00', '0x58']

42 10:34:31.834 ['0x02', '0x00', '0x00', '0x00', '0x00', '0x00']

43 10:34:32.008 ['0x67', '0x20', '0x72', '0x30', '0x0d', '0x76', '0x20', '0x31', '0x30', '0x30', '0x39', '0x0d']

Below is the same data formatted correctly found in data_new.txt file.

60 10:34:31.906 00 fb 01 0c 00 ac Get Variable 0x00ac

61 10:34:31.923 00 a8 02 00 00 00 f0 00

62 10:34:31.941 00 f3 01 0c 00 a4 Get Variable 0x00a4

63 10:34:31.959 00 58 02 00 00 00 00 00

64 10:34:31.977 g r0 ASCII Command

65 10:34:31.995 v 1009 ASCII Response

The example above was generated by Serial_Sniffer.py. The script will format all the data in either serial binary or ASCII command format.

Python Script Serial_Sniffer.py:

```
#!/usr/bin/python
import serial, time, sys, re
from time import gmtime, strftime
from datetime import datetime
from functools import reduce

ser = serial.Serial('COM20', 115200, timeout = 2); # Please change COM port to correct number
ser.flushInput();
ser.flushOutput();
fp_new = open("data_new.txt", "w")
fp_raw = open("data_raw.txt", "w")

COM = "COM20"

D_ALL = []            # All raw unformatted data stored here
T_ALL = []            # All timestamps for raw data stored here
D_TEMP = []           # An empty list used to temporarily store entries
D_NEW = []            # All the correctly formatted data
D_NEW_0 = []          # All the correctly formatted data with an artificially generated timestamp
time_num_list = []    #Formats all timestamps in T_ALL and converts each timestamp to a number
New_Time = []         # List of artificially generated timestamps

'''
When using the ASCII Command Line, type all letters lower case
'''

ASCII_MESSAGE = ["                    ASCII Command", "                    ASCII Response"]

ASCII_SPACE = ['0x20'] #ASCII Character for typing a space (pressing space bar)
ASCII_PERIOD = ['0x2e'] #ASCII Character for typing a period (.)
ASCII_0_Node_ID = ['0x30','0x31','0x32'] # CAN Node ID's are 0,1,2. You can add more if you
like. Just input the ASCII number in hexadecimal form here.
ASCII_1_AXIS_LETTER = ['0x61','0x62','0x63','0x64'] #Axis a, b, c, d (up to 4 axis per drive)
ASCII_2_COMMAND_CODE = ['0x67','0x69','0x63','0x72','0x73','0x74'] # different command
types: g-get, i-register, c-copy, r-reset, s-set, t-trajectory
ASCII_3_MEMORY_BANK = ['0x72','0x66'] # r-RAM, f-FLASH
ASCII_4_CARRIAGE_RETURN = ['0x0d']
ASCII_5_The_Letter_l = ['0x6c'] # The letter 'l' used in "ldenc" command
ASCII_6_The_Letter_d = ['0x64'] # The letter 'd' used in "ldenc" command
ASCII_7_The_Letter_e = ['0x65'] # The letter 'e' used in "enc" or "ldenc" encoder command
ASCII_8_The_Letter_n = ['0x6e'] # The letter 'n' used in "enc" or "ldenc" encoder command
ASCII_9_The_Letter_c = ['0x63'] # The letter 'c' used in "enc" or "ldenc" encoder command
ASCII_10_The_Letter_t = ['0x74'] # The letter 't' used in a trajectory command
```

```
ASCII_11_The_Trajectory_Command_Numbers = ['0x30','0x31','0x32','0x33','0x34'] #The
numbers (0,1,2,3,4) used after the trajectory command "t"

AXIS = ['0x00']   # input axis number to monitor
OP_CODE =
['0x00','0x01','0x03','0x04','0x05','0x06','0x07','0x08','0x09','0x0a','0x0b','0x0c','0x0d','0x0e','0x
0f','0x10','0x11','0x12','0x14','0x15','0x16','0x17','0x18','0x1b','0x1c','0x1d','0x1e','0x1f','0x21']
x = 0                # initializes a variable used throughout
while True:
  try:
    time_raw = datetime.now()
    time_now = time_raw.strftime('%H:%M:%S.%f')[:-3]
    bytesToRead = ser.inWaiting()
    if(bytesToRead > 0):
      data_raw = ser.read(bytesToRead)
      S = ["0x{:02x}".format(i) for i in data_raw] #S = ['0x%02x' % ord(i) for i in data_raw]
      D_ALL.append(S)
      T_ALL.append(time_now)
      x += 1
      print(x-1, T_ALL[x-1], D_ALL[x-1])

  except KeyboardInterrupt:

    with open("data_raw.txt", "w") as file: #Compile raw data with timestamp
      for i in range(0, len(D_ALL)):
        fp_raw.write("%d %s %s \n"%(i,T_ALL[i],D_ALL[i]))

    for i in range(0, len(D_ALL)-1): #Compile data into list; each element is one byte of data
      D_ALL[0] += D_ALL[1]
      del(D_ALL[1])
    D_ALL0 = D_ALL[0]

    # Provides helpful information to user after reading the op-code of the message
    def Look_Up_Op_Code(Op_Code):
      x = Op_Code
      if (x == '00'): return("           ")
      elif (x == '07'): return("         Get Amplifier Operating Mode ")
      elif (x == '0a'): return("         Get Flash CRC Value ")
      elif (x == '0b'): return("         Swap Operating Modes ")
      elif (x == '0c'): return("         Get Variable ")
      elif (x == '0d'): return("         Set Variable ")
      elif (x == '0e'): return("         Copy Variable ")
      elif (x == '0f'): return("         Trace Command ")
      elif (x == '10'): return("         Reset Command ")
      elif (x == '17'): return("         Trajectory Command ")
      elif (x == '12'): return("         Error Log Command ")
      elif (x == '14'): return("         CVM Command ")
      elif (x == '1b'): return("         Encoder Command ")
      elif (x == '1c'): return("         Get CAN Object Command ")
      elif (x == '1d'): return("         Set CAN Object Command ")
```

```
        elif (x == '21'): return("          Dynamic File Command Interface ")
        else: return("          Op-code Unknown ")

    # Provides helpful information to user for Get and Set op-codes
    def Parameter_Information(List):
        x = List
        if ((x[2] == '01') and (x[3] == '0c')): return("0x" + str(x[4]) + str(x[5]))
        elif ((x[2] == '02') and (x[3] == '0d')): return("0x" + str(x[4]) + str(x[5]) + " to 0x" +
str(x[6]) + str(x[7]))
        elif ((x[2] == '03') and (x[3] == '0d')): return("0x" + str(x[4]) + str(x[5]) + " to 0x" +
str(x[6]) + str(x[7]) + str(x[8]) + str(x[9]))
        else: return (" ")

    #Is_ASCII_Command will determine if the data is formatted in such a way that the data holds
the start of an ASCII Command. Returns True or False.
    def Is_ASCII_Command(List):
        x = List
        if ((x[0] in ASCII_0_Node_ID) and (x[1] in ASCII_PERIOD) and (x[2] in
ASCII_1_AXIS_LETTER)): return(True) #Example of ASCII Command using this format: 0.a g r0
        elif ((x[0] in ASCII_0_Node_ID) and (x[1] in ASCII_SPACE) and (x[2] in
ASCII_2_COMMAND_CODE)): return(True) # Example: 3 s f0x30 1200
        elif ((x[0] in ASCII_PERIOD) and (x[1] in ASCII_1_AXIS_LETTER) and (x[2] in
ASCII_SPACE)): return(True) # Example: .b s f0x30 1200
        elif ((x[0] in ASCII_2_COMMAND_CODE) and (x[1] in ASCII_SPACE) and (x[2] in
ASCII_3_MEMORY_BANK)): return(True) # Example: g r0
        elif ((x[0] in ASCII_5_The_Letter_l) and (x[1] in ASCII_6_The_Letter_d) and (x[2] in
ASCII_7_The_Letter_e)): return(True) # Example: ldenc clear
        elif ((x[0] in ASCII_7_The_Letter_e) and (x[1] in ASCII_8_The_Letter_n) and (x[2] in
ASCII_9_The_Letter_c)): return(True) # Example: enc clear
        elif ((x[0] in ASCII_10_The_Letter_t) and (x[1] in ASCII_SPACE) and (x[2] in
ASCII_11_The_Trajectory_Command_Numbers)): return(True) #Example: t 1
        else: return(False)

    # Correct_ASCII_Index will return an index value. The index value is where the main function
will begin searching for the second carriage return after the first message. The length of the first
message depends on the format of the message.
    def Correct_ASCII_Index(List):
        x = List
        if ((x[0] in ASCII_0_Node_ID) and (x[1] in ASCII_PERIOD) and (x[2] in
ASCII_1_AXIS_LETTER)): return(8) # Example of ASCII Command using this format: 0.a g r0
        elif ((x[0] in ASCII_0_Node_ID) and (x[1] in ASCII_SPACE) and (x[2] in
ASCII_2_COMMAND_CODE)): return(6) # Example: 3 s f0x30 1200
        elif ((x[0] in ASCII_PERIOD) and (x[1] in ASCII_1_AXIS_LETTER) and (x[2] in
ASCII_SPACE)): return(7) #Example: .b s f0x30 1200
        elif ((x[0] in ASCII_2_COMMAND_CODE) and (x[1] in ASCII_SPACE) and (x[2] in
ASCII_3_MEMORY_BANK)): return(4) #Example: g r0
        elif ((x[0] in ASCII_5_The_Letter_l) and (x[1] in ASCII_6_The_Letter_d) and (x[2] in
ASCII_7_The_Letter_e)): return(11) #Example: ldenc clear
        elif ((x[0] in ASCII_7_The_Letter_e) and (x[1] in ASCII_8_The_Letter_n) and (x[2] in
ASCII_9_The_Letter_c)): return(9) #Example: enc clear
```

```
        elif ((x[0] in ASCII_10_The_Letter_t) and (x[1] in ASCII_SPACE) and (x[2] in
ASCII_11_The_Trajectory_Command_Numbers)): return(3) #Example: t 1
        else: return(False)

    def time_to_num(x): #Displays timestamp data
        t = x
        t = re.findall(r"[\d']+", t)
        (h, m, s, us) = t
        result = float(h)*3600 + float(m)*60 + float(s) + (float(us)*1e-3)
        return(result)

    def num_to_date(x): #Returns string in hh:mm:ss.uuu format
        return "%02d:%02d:%02d.%03d"%reduce(lambda
ll,b:divmod(ll[0],b)+ll[1:],[(x*1000,),1000,60,60])

    def parse_data(): #Correctly format data
        global D_ALL0, D_NEW
        bytes = int(D_ALL0[2],0)
        bytes = bytes*2
        list_edit = D_ALL0[0:bytes+4]
        list_edit = [int(x,16) for x in list_edit]
        checksum = reduce(lambda x,y:x^y, list_edit)
        checksum = hex(checksum)
        list_edit = ['0x%02x' % x for x in list_edit]
        if((checksum == '0x5a') and (list_edit[0] in AXIS) and (list_edit[3] in OP_CODE)):
            list_edit = [int(x,16) for x in list_edit]
            list_edit = ['%02x' % x for x in list_edit]
            Op_Code_Description = Look_Up_Op_Code(list_edit[3])
            I6format = " ".join(list_edit)
            D_NEW += [(str(I6format))+ str(Op_Code_Description) +
Parameter_Information(list_edit)]
            del D_ALL0[0:(len(list_edit))]

        elif Is_ASCII_Command(D_ALL0):
            carriage_return_counter = 0
            carriage_return_finder = Correct_ASCII_Index(D_ALL0)
            '''
            carriage_return_finder sets the first entry (or byte) for the while loop to check if it is the
first carriage return.
            The while loop will start checking each element after the ASCII Command and compare it
to the carriage return.
            Example: The ASCII Command is g r0 where the first 3 bytes will be the command code
"g" followed by a space " " followed by a memory bank "r".
            We would set the carriage_return_finder = 4 in this case because we know elements 0-3
"g r0" are not the carriage return.
            '''
            while carriage_return_counter < 2:
                if D_ALL0[carriage_return_finder] in ASCII_4_CARRIAGE_RETURN:
                    carriage_return_counter += 1
                    if carriage_return_counter == 1:
```

```
                firstcarriagereturn = carriage_return_finder + 1
                list_edit_0 = D_ALL0[0:firstcarriagereturn]
                list_edit_0 = [int(x,16) for x in list_edit_0]
                list_edit_0 = ['%02x' % x for x in list_edit_0]
                I6format = " ".join(list_edit_0)
                I6format = bytearray.fromhex(I6format).decode()
                D_NEW += [(str(I6format)) + ASCII_MESSAGE[0]]
              elif carriage_return_counter == 2:
                secondcarriagereturn = carriage_return_finder + 1
                list_edit_0 = D_ALL0[firstcarriagereturn:secondcarriagereturn]
                list_edit_0 = [int(x,16) for x in list_edit_0]
                list_edit_0 = ['%02x' % x for x in list_edit_0]
                I6format = " ".join(list_edit_0)
                I6format = bytearray.fromhex(I6format).decode()
                D_NEW += [(str(I6format)) + ASCII_MESSAGE[1]]
            carriage_return_finder += 1
          del D_ALL0[0:secondcarriagereturn]
        else:
          del D_ALL0[0]

    # Parse all the data until the length of the data is less than 4.
    # The universal length of bytes of any serial binary header is 4.
    # If there are only 3 bytes left in the data, there couldn't possibly be a message there.
    while(len(D_ALL0)>=4):
      parse_data()

    for i in range(len(T_ALL)):
      time_num_list.append(time_to_num(T_ALL[i]))

    new_time = [time_num_list[0], time_num_list[-1]]
    diff = new_time[1] - new_time[0]
    q = diff/(len(D_NEW)-1)
    del(new_time[1])
    # The list "new_time" contains the lowest, earliest timestamp value (known as the first
timestamp)

    # The first timestamp will be added by a constant q for as many times as we need to match
the length of D_NEW
    while(len(new_time)<len(D_NEW)): #Creates new timestamps
      u = new_time[-1] + q
      new_time.append(u)

    for i in range(len(new_time)):
      New_Time.append(num_to_date(new_time[i]))

    for i in range(len(D_NEW)):
      b = []
      b += str(New_Time[i])
      b += str(' ')
      b += str(D_NEW[i])
```

```
    b = [""].join(b)]
    D_NEW_0 += b

with open("data_new.txt", "w") as file:
    for i in range(0, len(D_NEW_0)):
        fp_new.write("%d %s \n"%(i,D_NEW_0[i]))

sys.exit(0)
```

Revision History

Date	Version	Revision
12/3/2019	Rev 00	Initial release