# Save Lab Space and Cost: Controlling Serial and Ethernet Test Instruments with a Raspberry Pi 3 Model B

APPLICATION NOTE

KEITHLEY
A Tektronix Company

Tektronix®

Part of the cost of establishing and operating a test lab is the cost of the computer resources necessary to automate routines and procedures. Whether it is a single or multiple station setup, quality test equipment represents a better investment than a computer. Although it could be argued that the cost of a computer is minimal compared to that of the test equipment, the few hundred dollars spent on a computer could be better applied to adding features or options to the test equipment. Therefore, leveraging a low-cost alternative with a sizable online user community and numerous examples is worth considering.

Raspberry Pi is a credit card-sized, single-board computer that features an ARM-compatible integrated CPU and an on-chip GPU. The Pi's small size, low power requirements, and adaptability make it very popular for a variety of applications, including robotics, data logging, and other electronics projects. The recommended distribution for the Raspberry Pi is known as Raspbian, a Debian-based Linux operating system. The newest edition of Raspbian—Stretch—comes equipped with various software development tools and supports many programming languages, including Python. The Python programming language is known for its readability and ease of use. This document details the best methods for remote instrumentation and data acquisition using the Raspberry Pi 3 Model B.

This document describes a general setup of the Raspberry Pi and the installation of recommended software tools for test automation. Example code for the Keithley DAQ6510 Data Acquisition and Logging Multimeter System demonstrates the implementation of PyVISA and how an operator can communicate with lab test tools. An example illustrates pure sockets-based communication (for Ethernet/LAN-enabled instrumentation) that minimizes the number of software plugins necessary to get operational.

## Raspberry Pi Setup and Configuration

Using the Raspberry Pi requires an HDMI-compatible monitor or another monitor type with an HDMI adapter cable, a micro USB power supply that can supply 2 A at 5 V, and a standard keyboard and mouse. To turn the Pi on, simply connect it to power and it will boot automatically.

### Installing Raspbian

The Raspberry Pi Foundation recommends using a class 4 8 GB micro SD card, ideally with NOOBS (New Out of Box Software) preinstalled. NOOBS is the operating system installation manager for the Raspberry Pi.

If not using a pre-configured SD card for the Raspberry Pi, the first step is to acquire a blank micro SD card with a minimum of 8 GB of storage space. For SD cards 8 GB or larger, the card should be reformatted as FAT. If the chosen SD card has more than 32 GB of storage space (for example, 64 GB or more), it should be reformatted to FAT32.

Once the card is formatted, install NOOBS onto it. The NOOBS zip file can be downloaded from the official Raspberry Pi website. Once the file has been downloaded, copy the contents to the root of the blank formatted SD card. Insert the micro SD into the slot on the bottom of the Raspberry Pi.

### Initial Boot

The first time the Pi is powered on, select Raspbian (full desktop version) under the Install tab. The full version of NOOBS has Raspbian included, so it can be directly installed from the SD card without an Internet connection. If the operating system image on the card is outdated and a new version has been released, the option to download the latest version becomes available once the Raspberry Pi is connected to the internet.

To establish an Internet connection, simply connect the Pi to the network via the LAN port or use the Raspberry Pi 3 Model B's onboard WiFi capabilities to connect to an available wireless network through the WiFi tab in the NOOBS installer window. An Internet connection is necessary to download and install updates and new Python packages from the command terminal.

Once Raspbian is installed, the desktop will appear, which offers access to the applications menu in the top-left corner. The applications menu has all the included programming IDEs, the Raspberry Pi configuration options, and shutdown options.

The Raspbian installation can be updated to install only the most recent version of the current Raspbian image on the Pi. It will not upgrade the operating system to a different Raspbian image.

To apply updates, open the command terminal and input:

```
sudo apt-get update
sudo apt-get dist-upgrade
```

Then, reboot the Raspberry Pi.

For more information about setup, see the official Raspberry Pi Foundation documentation page.

## VISA

The Raspberry Pi can communicate with instruments through VISA via Python using the PyVISA library. PyVISA is a frontend to VISA, providing a Python API that can connect to multiple backends. The default backend is the traditional National Instruments NI-VISA library. However, NI-VISA is not compatible with Raspbian on the Raspberry Pi. Instead, the PyVISA-py backend is used. PyVISA-py is a purely Python implementation of the VISA library that supports the most common attributes and methods.

### Python2

To install PyVISA on the raspberry pi, open the Raspberry Pi command prompt and input:

```
pip install –U pyvisa
```

To install PyVISA-py on the raspberry pi, open the Raspberry Pi command prompt and input:

```
pip install pyvisa-py
```

### Python3

To install PyVISA on the raspberry pi, open the Raspberry Pi command prompt and input:

```
pip3 install –U pyvisa
```

To install PyVISA-py on the raspberry pi, open the Raspberry Pi command prompt and input:

```
pip3 install pyvisa-py
```

The PyVISA-py backend can be selected when instantiating the VISA Resource Manager by passing the '@py' argument. The '@py' argument allows PyVISA to bypass the default National Instruments backend.

The following is a basic example that details how to open, close, and verify a PyVISA TCP/IP resource connection in Python3 by querying for the instrument's ID string and printing it to the console window. Simply connect the Pi and the instrument with an Ethernet cable via the LAN ports.

### TCPIP Example

```
import visa                                            #Use pyvisa library

rm = visa.ResourceManager('@py')                       #Use the pyvisa-py backend
address = "TCPIP0::169.254.153.137::inst0::INSTR"      #Keithley DAQ6510 IP address TCP/IP string
inst = rm.open_resource(address)

inst.write("*RST")                                     #Sends the reset command to the instrument
print(inst.query("*IDN?"))                             #Prints the instrument ID string to the
                                                       command line


inst.close()                                           #Close the connection
rm.close()
```

The advantage to using PyVISA is that it can facilitate various interfacing methods for communication other than Ethernet, including RS-232 via PySerial. Due to the nature of the PyVISA-py backend, additional Python libraries must be installed to use resource types other than TCP/IP.

## Python2

To install PySerial on the raspberry pi, open the Raspberry Pi command prompt and input:

```
python -m pip install pyserial
```

To check what resource types are available and whether the libraries they depend on have been installed, open the command terminal and input:

```
python -m visa info
```

## Python3

To install PySerial on the raspberry pi, open the Raspberry Pi command prompt and input:

```
python3 -m pip install pyserial
```

To check what resource types are available and whether the libraries they depend on have been installed, open the command terminal and input:

```
python3 -m visa info
```

To establish serial communication between the Raspberry Pi and an instrument, simply use a USB-to-serial converter cable with the USB end plugged into the Pi and the serial pins connected to the instrument's RS-232 port. For best results, set the instrument's baud rate to 9600.

## Python2

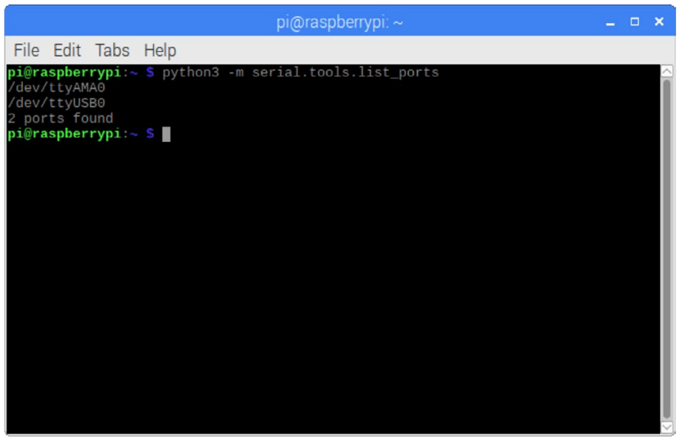
To identify available serial ports or adapters that are connected to the Pi, open the Raspberry Pi command prompt and input:

```
python -m serial.tools.list_ports
```

## Python3

To identify available serial ports or adapters that are connected to the Pi, open the Raspberry Pi command prompt and input:

```
python3 -m serial.tools.list_ports
```



```
pi@raspberrypi:~ $ python3 -m serial.tools.list_ports
/dev/ttyAMA0
/dev/ttyUSB0
2 ports found
pi@raspberrypi:~ $
```

The following example details how to open, close, and execute a long-term temperature scan with a PyVISA Serial RS-232 resource connection in Python3. The instrument's ID string and the scan data is printed to the console before being sent to a .csv file.

## RS-232 Example

```
import visa                                        #Use pyvisa library
import time

rm = visa.ResourceManager('@py')                   #Use the pyvisa-py backend
address = "ASRL/dev/ttyUSB0::INSTR"                #Raspberry Pi USB Serial Port ASRL String
inst = rm.open_resource(address)
inst.write("*RST")
print(inst.query("*IDN?"))

inst.write("FORM:DATA ASCII")                      #Format scan data into ascii

inst.write("FUNC 'TEMP', (@101, 102)")             #Measure Temperature
inst.write("TEMP:TRAN FRTD, (@101, 102)")          #Set transducer to four wire RTD
inst.write("TEMP:RTD:FOUR PT3916, (@101, 102)")    #Set the RTD type to PT3916
inst.write("TEMP:OCOM ON, (@101, 102)")            #Turn on offset compensation
inst.write("TEMP:ODET ON, (@101, 102)")            #Turn on open lead detector

inst.write("ROUT:SCAN:COUN:SCAN 25")               #Set scan count to 25
inst.write("ROUT:SCAN:INT 300")                    #Execute a scan every 5 minutes
inst.write("ROUT:SCAN:CRE (@101, 102)")            #Scan channels 101 and 102
inst.write("INIT")                                 #Initiate the scan

# monitor for scan completion…

time.sleep(5.0)
trigState = inst.query(':TRIG:STAT?')              # query the state of the scan activity which is either
                                                   # "running" when the measurements are being made
                                                   # or "waiting" when the interval (delay) between scans
                                                   # is active

while("RUNNING" in trigState) | ("WAITING" in trigState):
  print("Running....")
  time.sleep(5.0)
  trigState = inst.query(':TRIG:STAT?')

data = inst.query('TRAC:DATA? 1, 50, "defbuffer1", CHAN, READ, UNIT')
print(data)                                        #Print readings to console

inst.close()                                       #Close the connection

#send data to csv file
with open("/home/pi/4Wire_RTD_Temperature.csv", "a") as log:
  log.write(data)

rm.close()
```

For more information, see the official PyVISA, PyVISA-py, and PySerial documentation.

## Ethernet Sockets

Communication through Sockets programming on the Raspberry Pi is possible via the socket module in the Python standard library. Socket programming allows two devices on the same network to communicate. For remote instrumentation, the instrument acts as a server and listens, while the Pi is programmed as a client that reaches out to form a connection.

Sockets programming is the most straightforward method to communicate with instruments with the Raspberry Pi because it does not require the user to install additional Python packages and is facilitated by a simple Ethernet network connection.

The following example details how to open, close, and execute an AC parameters scan with a socket connection in Python3. The instrument's ID string and the scan data is printed to the console, before being sent to a .csv file.

### Example

```
import socket
import time

TCP_IP = "169.254.153.137"                         #Instrument IP Address
TCP_PORT = 5025

def instsend(s, command):
  command += "\n"
  s.send(command.encode())
  return

def instrecv(s):
  return s.recv(1024).decode()

def instquery(s, command):
  instsend(s, command)
  return instrecv(s)

s = socket.socket()
s.connect((TCP_IP, TCP_PORT))                      #Pi will be client to server instrument
instsend(s, "*RST")
print(instquery(s, "*IDN?"))

instsend(s, "FORM:DATA ASCII")                     #Format scan data into ascii

instsend(s, "FUNC 'VOLT:AC', (@101)")              #Measure AC Volts on channel 1
instsend(s, "FUNC 'FREQ', (@102)")                 #Measure Frequency on channel 2
instsend(s, "FUNC 'PER', (@103)")                  #Measure Period on channel 3
instsend(s, "FUNC 'CURR:AC', (@121)")              #Measure AC Current on channel 21

instsend(s, "ROUT:SCAN:COUN:SCAN 10")              #Set scan count to 10
instsend(s, "ROUT:SCAN:CRE (@101:103, 121)")       #Scan channels 101-103 and 121
instsend(s, "INIT")                                #Initiate the scan

time.sleep(0.5)
trigState = instquery(s, ':TRIG:STAT?')            # query the state of the scan activity which is either
                                                   # "running" when the measurements are being made
                                                   # or "waiting" when the interval (delay) between scans
                                                   # is active
while("RUNNING" in trigState) | ("WAITING" in trigState):
  print("Running....")
  time.sleep(5.0)
  trigState = instquery(s, ':TRIG:STAT?')

data = instquery(s, 'TRAC:DATA? 1, 40, "defbuffer1", CHAN, READ, UNIT')
print(data)                                        #Print readings to console
s.close()                                          #Close the connection

with open("/home/pi/AC_Parameters.csv", "a") as log:  #send data to csv file
 log.write(data)
```

## Contact Information:

| | |
|---|---|
| **Australia*** | 1 800 709 465 |
| **Austria** | 00800 2255 4835 |
| **Balkans, Israel, South Africa and other ISE Countries** | +41 52 675 3777 |
| **Belgium*** | 00800 2255 4835 |
| **Brazil** | +55 (11) 3759 7627 |
| **Canada** | 1 800 833 9200 |
| **Central East Europe / Baltics** | +41 52 675 3777 |
| **Central Europe / Greece** | +41 52 675 3777 |
| **Denmark** | +45 80 88 1401 |
| **Finland** | +41 52 675 3777 |
| **France*** | 00800 2255 4835 |
| **Germany*** | 00800 2255 4835 |
| **Hong Kong** | 400 820 5835 |
| **India** | 000 800 650 1835 |
| **Indonesia** | 007 803 601 5249 |
| **Italy** | 00800 2255 4835 |
| **Japan** | 81 (3) 6714 3086 |
| **Luxembourg** | +41 52 675 3777 |
| **Malaysia** | 1 800 22 55835 |
| **Mexico, Central/South America and Caribbean** | 52 (55) 56 04 50 90 |
| **Middle East, Asia, and North Africa** | +41 52 675 3777 |
| **The Netherlands*** | 00800 2255 4835 |
| **New Zealand** | 0800 800 238 |
| **Norway** | 800 16098 |
| **People's Republic of China** | 400 820 5835 |
| **Philippines** | 1 800 1601 0077 |
| **Poland** | +41 52 675 3777 |
| **Portugal** | 80 08 12370 |
| **Republic of Korea** | +82 2 565 1455 |
| **Russia / CIS** | +7 (495) 6647564 |
| **Singapore** | 800 6011 473 |
| **South Africa** | +41 52 675 3777 |
| **Spain*** | 00800 2255 4835 |
| **Sweden*** | 00800 2255 4835 |
| **Switzerland*** | 00800 2255 4835 |
| **Taiwan** | 886 (2) 2656 6688 |
| **Thailand** | 1 800 011 931 |
| **United Kingdom / Ireland*** | 00800 2255 4835 |
| **USA** | 1 800 833 9200 |
| **Vietnam** | 12060128 |

*** European toll-free number. If not accessible, call:** +41 52 675 3777

Rev. 02.2018

KEITHLEY
A Tektronix Company

Find more valuable resources at TEK.COM

090618 SBG 1KW-61463-0