System Power Supply Programming

# Using Python and Sockets

Python programming is growing in popularity as it is easy to learn and available free of charge from www.python.org. Python can be used with sockets to send SCPI commands to a system power supply connected to the LAN. This article will describe four functions used to send SCPI commands and receive data from a system power supply. To get started programing you will need to download a free version of Python and the free environment Microsoft Visual Studio Code. If you are not familiar with Python, it supports a legacy 2.x version plus the newest version 3.x. The general rule of thumb is to use the latest version (3.7) unless you have a large installed base of 2.x code.

**Advantages of Python for test engineering**

- Free, open source
- Easy to learn
- Extensive libraries
- Integrates well with other languages

## Program Structure

The code is structured to emphasize programming the system power supply. Four functions were created to make it easy to explain the power supplies commands (SCPI) without having to understand Python or sockets. For example, `outPut('*RST')` sends the SCPI command `*RST` to the power supply. Future white papers will be more focused on the power supply SCPI commands. This white paper will provide a little background on Python, sockets and the four functions used to send the SCPI commands.

| Four functions used to communicate with Power Supply |
|---|
| Open Socket: `openSocket(IPaddr,port)` – Create a connection to IP address & port |
| Close Socket: `closeSockets()` – Closes the socket connection |
| Output: `outPut(cmd1)` – Sends SCPI commands as a string |
| Enter: `enTer()` – receives instrument data as a string |

**Figure 1. Four functions were created to send the SCPI commands and receive the measurement data from the power supply.**

## Python

If you are new to Python, you can find many useful tutorials and code snippets to explain every Python concept. One of the core benefits of Python is that it is easy to learn. There are a couple of things to keep in mind. Python is whitespace sensitive and uses indentation for flow control. Using the development environment Visual Studio Code helps with creating the proper indenting. Figure 2. provides an example of using white space (indenting) for flow control. Typically, variables are not declared or provided a data type and can be re-used. Although, a global declaration is available to share variables and their values between all functions.

```
def outPut(cmd1):              # Send SCPI command via sockets
    cmd1 = cmd1 + '\n'
    skt.send(cmd1.encode('ASCII'))
```

**Figure 2. Python uses indenting for flow control.**

## Sockets

Socket connections provide a way to communicate with an instrument connected via LAN. System power supplies often use the LAN interface as they are frequently located remotely in a test system. If you are interested in further exploring sockets, our power supplies use internet domain, streaming sockets. Software must use the same type of socket along with an IP address and port to connect. Sockets have built-in error trapping which is helpful for debugging. In addition, the code example, Figure 3 is using the socket-timeout feature to trap any errors that occur in communicating with the instrument.

## Open Socket Function

An IP address and port number are passed to the open socket routine. The Python code is shown in Figure 3. A global connection allows the other functions to be able to access it. Error and exceptions are handled in Python with a try-except statement. The first try-except routine defines the socket type, internet/streaming and sets the timeout. Our second try-except routine specifies the address for our socket connection. If the instrument is offline or has been assigned a new address an "`Error connecting to socket on instrument: timed out`" will occur. The error string is assigned to e using the as command. The %s acts as a placeholder for a string inside a print command. The placeholder is replaced by what follows % which in our case is the variable `e`.

```
def openSocket(IPaddr,port):      # Create a connection via sockets
    global skt

    try:
        skt = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        skt.settimeout(8)   # 8 second timeout
    except socket.error as e:
        print('Error creating socket: %s' % e)
        sys.exit(1)

    try:
        skt.connect((IPaddr, port))
    except socket.gaierror as e:
        print('Address-related error connecting to instrument: %s' % e)
        sys.exit(1)
    except socket.error as e:
        print('Error connecting to socket on instrument: %s' % e)
        sys.exit(1)
```

**Figure 3. Python function to open a socket to an instrument.**

## Output and Enter Functions

The output and enter functions are straightforward. The power supply receives the SCPI commands as strings followed by a “newline.” Python uses `“/n”` to send the code for a newline. The output function passes a SCPI command along with the new line to the instrument.

```
def outPut(cmd1):                  # Send SCPI command via sockets
    cmd1 = cmd1 + ‘\n’
    skt.send(cmd1.encode(‘ASCII’))

def enTer():                        # Receive instrument data via sockets
    dataStr=skt.recv(1024).decode(‘ASCII’)
    return dataStr.strip()
```

**Figure 4. Output and Enter functions pass ASCII data to and from the instrument.**

The enter function receives ASCII data from the instrument. Spaces and escape codes are stripped from the string. `Print(enTer())` will display the string received from the instrument. The routine can return numerical data or messages.

## Close Socket Function

The `closeSockets()` function is a one-line function `skt.close()`. It was created to keep symmetry with the `openSocket()` routine. Closing the sockets releases the resources associated with the connection.

## Summary

Python along with sockets provides a simple way to send SCPI commands and receive data from a system power supply. Using the four functions `openSocket()`, `outPut()`, `enTer()`, and `closeSockets()` makes it easy to view the SCPI commands being sent to the instrument. We hope that you use the attached sample program and modify it for your application. The sample program sets the power supplies voltage then measures the voltage and current. Keep in mind that our call centers do not have the staffing to support sockets or Python but can help you with questions about our power supplies.

```
# An example program to configure channel 1 of a modular DC power
# supply. The program follows the flow: open resources, reset,
# configure, main body, release resources. The hardware used is an N6700C
# and an N6762A

import socket, sys, time

def outPut(cmd1):               # Send SCPI command via sockets
    cmd1 = cmd1 + '\n'
    skt.send(cmd1.encode('ASCII'))

def enTer():                       # Receive instrument data via sockets
    dataStr=skt.recv(1024).decode('ASCII')
    return dataStr.strip()

def openSocket(IPaddr,port):       # Create a connection via sockets
    global skt

    try:
        skt = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        skt.settimeout(8)         # 8 second timeout
    except socket.error as e:
        print('Error creating socket: %s' % e)
        sys.exit(1)
    try:
        skt.connect((IPaddr, port))
    except socket.gaierror as e:
        print('Address-related error connecting to instrument: %s' % e)
        sys.exit(1)
    except socket.error as e:
        print('Error connecting to socket on instrument: %s' % e)
        sys.exit(1)

def closeSockets():                 # Close socket connection
      skt.close()
#
# main function
if __name__ == '__main__':
#
#   Assign resources
    openSocket('10.112.158.34',5025)
```

**Sample program**

```
#
#    Reset & configure mainframe / instrument
     outPut('*RST')
     outPut('DISP:VIEW METER1')       # Single channel meter
     print('mainframe configured')
#
#    Configure output channel 1
     outPut('OUTP OFF, (@1)')         # Output off
     outPut('VOLT 0, (@1)')           # Set voltage to 0
     outPut('VOLT:PROT:LEV 6, (@1)') # Set voltage limit 6 V
     outPut('CURR 1, (@1)')           # Set CC limit 1 A
     print('channel 1 configured')
#
#    Main body - Output voltages from 0 to 5 V with 1 V increments,
#    measure the corresponding voltage and current.
     outPut('OUTP ON, (@1)')          # Turn output on
     for i in range(0, 6, 1):         # Step voltage from 0 to 5 V
         outPut('VOLT ' + str(i) + ', (@1)')  # Set output voltage
         time.sleep(5)                # Allow output to settle 5 s
         outPut('MEAS:VOLT? (@1)')    # Measure voltage
         respnse = enTer()            # Retrieve data
         print(respnse + ' V    ', end='') # Print voltage
         outPut('MEAS:CURR? (@1)')    # Measure current
         respnse = enTer()            # Retrieve data
         print(respnse, 'A')          # Print current
     outPut('OUTP OFF, (@1)')         # Turn output off
     print('loop complete')
#
#    Close socket - release resources
     closeSockets()
#
#    End
```

**Sample program continued**