Getting Started with Python and 2600B Products

For instrument control from Windows, is best to use NI-VISA as the communication layer. From Python, there is a pyVISA wrapper to the NI-VISA libraries.

If you have never used pyVISA, one of my colleagues has a video on getting up and running:

https://www.youtube.com/watch?v=W5Brxiwnp5g

Here is some Python code to send 2600B TSP product over LAN:

Substitute your resource name.
 	
Typical formats:
 	
USB0::0x05E6::0x3706::04406302::INSTR
 	
GPIB0::26::INSTR
 	
Note the simple write and read for interaction with the opened resou rce.
	import visa
	
	import time
	
	instrument_resource_string = "TCPIP0::192.168.1.55::inst0::INSTR"
	
	resource_mgr = visa.ResourceManager()
	
	my_instr = resource_mgr.open_resource(instrument_resource_string)
	
	my_instr.timeout = 5000 #timeout in msec
	
	my_instr.write("*IDN?")
	
	print(my_instr.read())

For TSP instruments, you can load functions into the runtime memory of the instrument.	#define a function in TSP Runtime Memory
	
	my_instr.write("loadscript myScriptName")
	
	my_instr.write("function myBeepFunction(duration, freq)")
	
	my_instr.write("beeper.beep(duration, freq)")
	
	my_instr.write("end") #function definition
	
	my_instr.write("endscript")
	
	#run the script to load into runtime memory
	
	my_instr.write("myScriptName.run()")

Once they are loaded, you can call your functions and pass parameter values	#call our function
	
	my_instr.write("myBeepFunction(1, 1200)")
	
	time.sleep(0.1);
	
	my_instr.write("myBeepFunction(1, 800)")

When all done, close the connection before exit from Python.	#put instrument back to local and close connection
	
	my_instr.clear()
	
	my_instr.close()
	
	resource_mgr.close()

KEY Take Aways:

Leverage VISA for instrument communication

Make use of function encapsulation to create your API into the instrument state machine. Load a named script with one or more functions into the runtime memory of the TSP product. Loading functions to runtime memory will increase your throughput by vastly reducing the bus traffic between the PC and the instrument.

At instrument power cycle, the script is lost so you need to load the functions for each power cycle. Alternately, you can elect to save the scripts into NVMemory space and even to auto load at instrument boot if desired.

Next Steps:

Use our TSP Toolkit to protype your TSP code/functions. The goal is to create a single TSP file that contains the function definitions for your named script.

TSP Toolkit base version is an open-source scripting tool available as a Visual Studio Code (VS Code) extension. It can be used to develop TSP test scripts alongside your other favorite VS Code extensions, such as those for Python, C# and many more.

[Test Automation | Tektronix]

Simple Examples for 2600B or 2600A SourceMeter Product

Example 1: TSP Commands to Apply a Voltage and Measure a Current

```
reset()
errorqueue.clear()

local v_level = 1
local limitI = 0.1

smua.source.func = smua.OUTPUT_DCVOLTS
smua.source.limiti = limitI
smua.source.rangev = 20
smua.source.levelv = 0   -- this is the resting bias level as soon as output is turned on
smua.measure.nplc = 1
smua.measure.delay = smua.DELAY_AUTO
smua.measure.delayfactor = 1.0

display.screen = display.SMUA  -- if a dual channel model, this sets display to SMUA
display.smua.measure.func = display.MEASURE_DCAMPS  -- front panel display the current measurement

smua.measure.rangei = 100e-9              -- set a starting range which also turns off auto range
smua.measure.autorangei = smua.AUTORANGE_ON  -- restore auto range
smua.measure.lowrangei = 100e-9            -- lowest range to use by autorange algorithm

smua.source.output = smua.OUTPUT_ON
smua.source.levelv = v_level
print(smua.measure.i())
smua.source.output = smua.OUTPUT_OFF

smua.source.levelv = 0
```

Comments:
Make this into two functions: one to apply configurations and one to perform and return measurement(s)

Example 2: TSP Commands for Simple Sweep on 2600A or 2600B

NOTE: models without a letter suffix, e.g. 2636 vs. 2636A or 2636B, do not support smuX.trigger.xxx commands

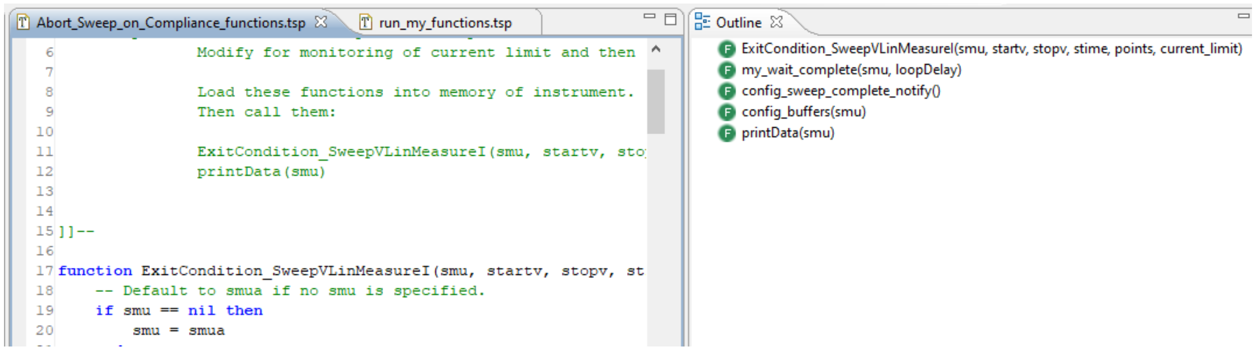
```
reset()
errorqueue.clear()
smu1 = node[1].smua   -- set an alias for the SMU channel to use
local startV = 0
local stopV = 10
local numPoints = 21
local limitI = 0.1    -- max current to allow
local nplc = 1       -- integration time for the measurement expressed as number of power line cycles

  -- Configure the SMU
  smu1.source.func                  = smu1.OUTPUT_DCVOLTS
  smu1.source.limiti                = limitI
  smu1.source.rangev = 20
  smu1.source.levelv = 0   -- this is the resting bias level as soon as output is turned on
  smu1.measure.nplc                 = nplc
  smu1.measure.delay                = smu1.DELAY_AUTO
  smu1.measure.delayfactor = 1.0

  smu1.measure.rangei = 100e-6              -- set a starting range which also turns off auto range
  smu1.measure.autorangei = smu1.AUTORANGE_ON  -- restore auto range
  smu1.measure.lowrangei = 100e-9            -- set the lowest range for auto range algorithm to use

  -- Prepare the Reading Buffers
  smu1.nvbuffer1.clear()
  smu1.nvbuffer1.collecttimestamps    = 1
  smu1.nvbuffer2.clear()
  smu1.nvbuffer2.collecttimestamps    = 1

  -- Configure SMU Trigger Model for Sweep

  -- when stimulus are set to zero, the trigger model passes on by without waiting for an event
  -- set these stimulus equal to something (digial trigger event) to coordinate the trigger model with other actions.
  smu1.trigger.arm.stimulus = 0
  smu1.trigger.source.stimulus = 0
  smu1.trigger.measure.stimulus = 0
  smu1.trigger.endpulse.stimulus = 0

  smu1.trigger.source.linearv(startV, stopV, numPoints)
  smu1.trigger.source.limiti         = limitI
  smu1.trigger.measure.action        = smu1.ENABLE
  smu1.trigger.measure.iv(smu1.nvbuffer1, smu1.nvbuffer2)

  smu1.trigger.endpulse.action       = smu1.SOURCE_HOLD

  -- By setting the endsweep action to SOURCE_IDLE, the output will return

  -- to the bias level at the end of the sweep.
  smu1.trigger.endsweep.action       = smu1.SOURCE_IDLE
  smu1.trigger.count                 = numPoints
  smu1.trigger.source.action         = smu1.ENABLE

  -- Ready to begin the test
  smu1.source.output                 = smu1.OUTPUT_ON
  -- Start the trigger model execution
  smu1.trigger.initiate()
  -- Wait until the sweep has completed
   waitcomplete()
```

```
smu1.source.output                = smu1.OUTPUT_OFF

-- Print the data back to the Console
print("Time\tVoltage\tCurrent")
for x=1,smu1.nvbuffer1.n do
   -- Voltage readings are in nvbuffer2.  Current readings are in nvbuffer1.
   print(smu1.nvbuffer1.timestamps[x], smu1.nvbuffer2[x], smu1.nvbuffer1[x])
 end
```

Advanced Example for 2600B or 2600A SourceMeter Product: sweep with SRQ on sweep done

Scenario: if doing breakdown voltage sweeps, desire ability to exit the sweep early if the current limit is encountered. Otherwise carry out the full sweep. Assert an SRQ when test is done.

Step 1: Here is using Test Script Builder software tool. It defines five functions to configure and carry out a voltage sweep and measures V and I. If the current limit is encountered, the sweep will abort early.

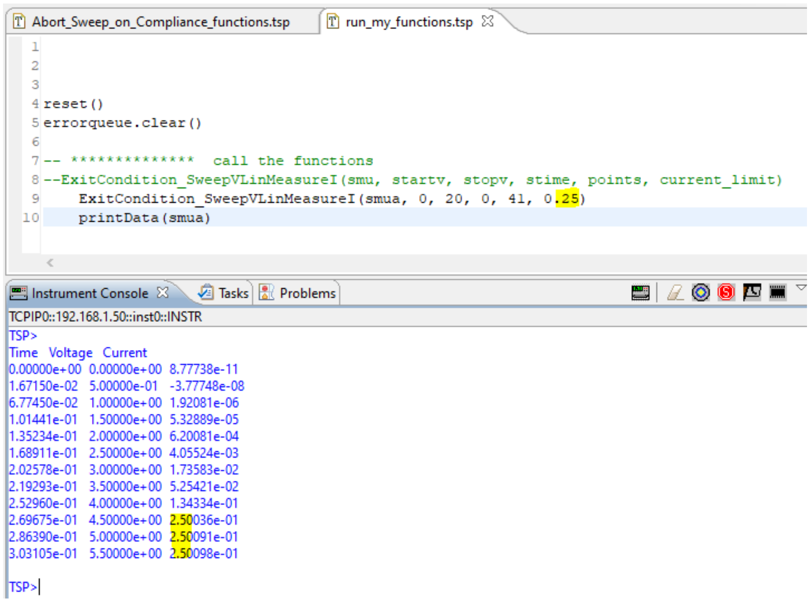


```
              Modify for monitoring of current limit and then

              Load these functions into memory of instrument.
              Then call them:

              ExitCondition_SweepVLinMeasureI(smu, startv, sto
              printData(smu)


]]--

function ExitCondition_SweepVLinMeasureI(smu, startv, stopv, st
    -- Default to smua if no smu is specified.
    if smu == nil then
        smu = smua
```

```
ExitCondition_SweepVLinMeasureI(smu, startv, stopv, stime, points, current_limit)
my_wait_complete(smu, loopDelay)
config_sweep_complete_notify()
config_buffers(smu)
printData(smu)
```

In a second file (run_my_functions.tsp), I can call the functions; this is my pseudo code for the later Python program and is used only for unit testing the function definitions.

Outcomes: The 250mA compliance condition was detected, and we exited before the full 41 points had been obtained. To have it exit more quickly, reduce the loopDelay value (50msec in this trial) for more frequent polling.



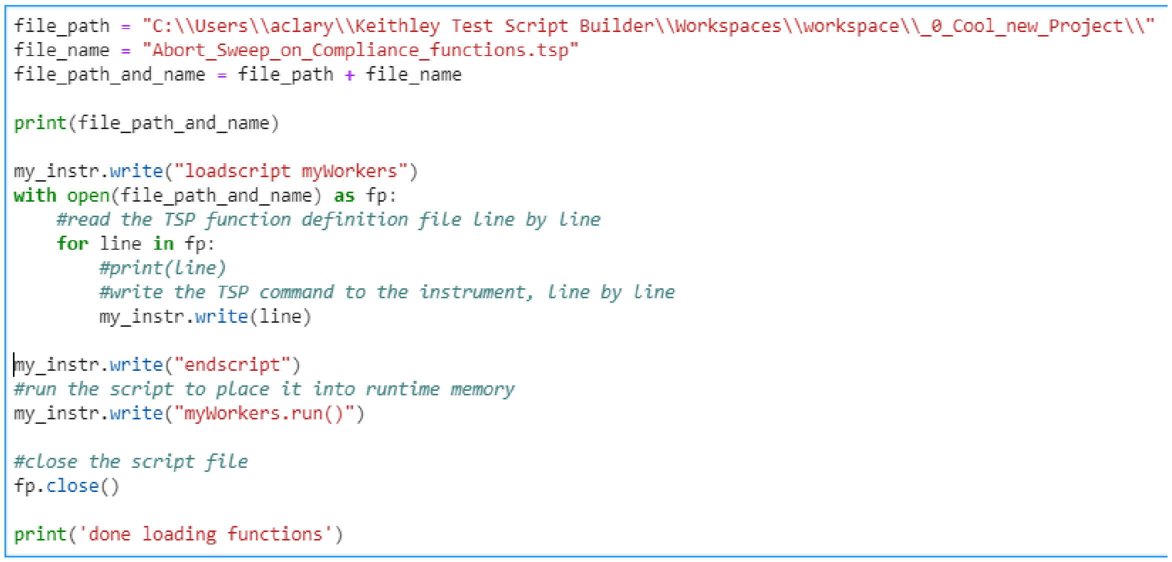
```



reset()
errorqueue.clear()

-- **************  call the functions
--ExitCondition_SweepVLinMeasureI(smu, startv, stopv, stime, points, current_limit)
    ExitCondition_SweepVLinMeasureI(smua, 0, 20, 0, 41, 0.25)
    printData(smua)
```

```
TCPIP0::192.168.1.50::inst0::INSTR
TSP>
Time  Voltage  Current
0.00000e+00  0.00000e+00  8.77738e-11
1.67150e-02  5.00000e-01  -3.77748e-08
6.77450e-02  1.00000e+00  1.92081e-06
1.01441e-01  1.50000e+00  5.32889e-05
1.35234e-01  2.00000e+00  6.20081e-04
1.68911e-01  2.50000e+00  4.05524e-03
2.02578e-01  3.00000e+00  1.73583e-02
2.19293e-01  3.50000e+00  5.25421e-02
2.52960e-01  4.00000e+00  1.34334e-01
2.69675e-01  4.50000e+00  2.50036e-01
2.86390e-01  5.00000e+00  2.50091e-01
3.03105e-01  5.50000e+00  2.50098e-01

TSP>
```

NOTE: close connection from Test Script Builder before attempting from Python

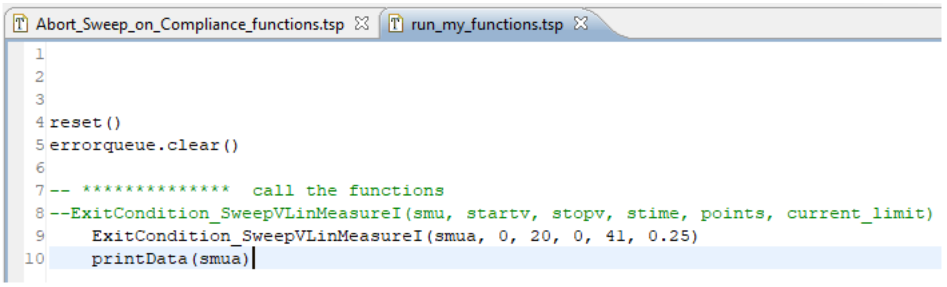
Step 2: From Python, use pyvisa and get a session.

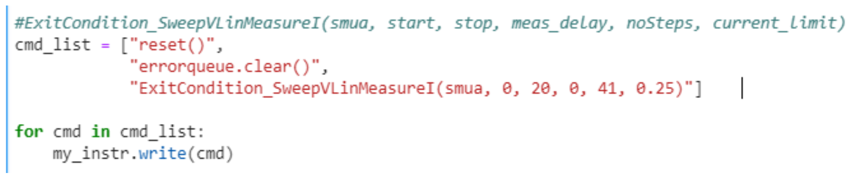
Load the tsp functions from step 1 by reading the function definition file and writing to the instrument, line by line.



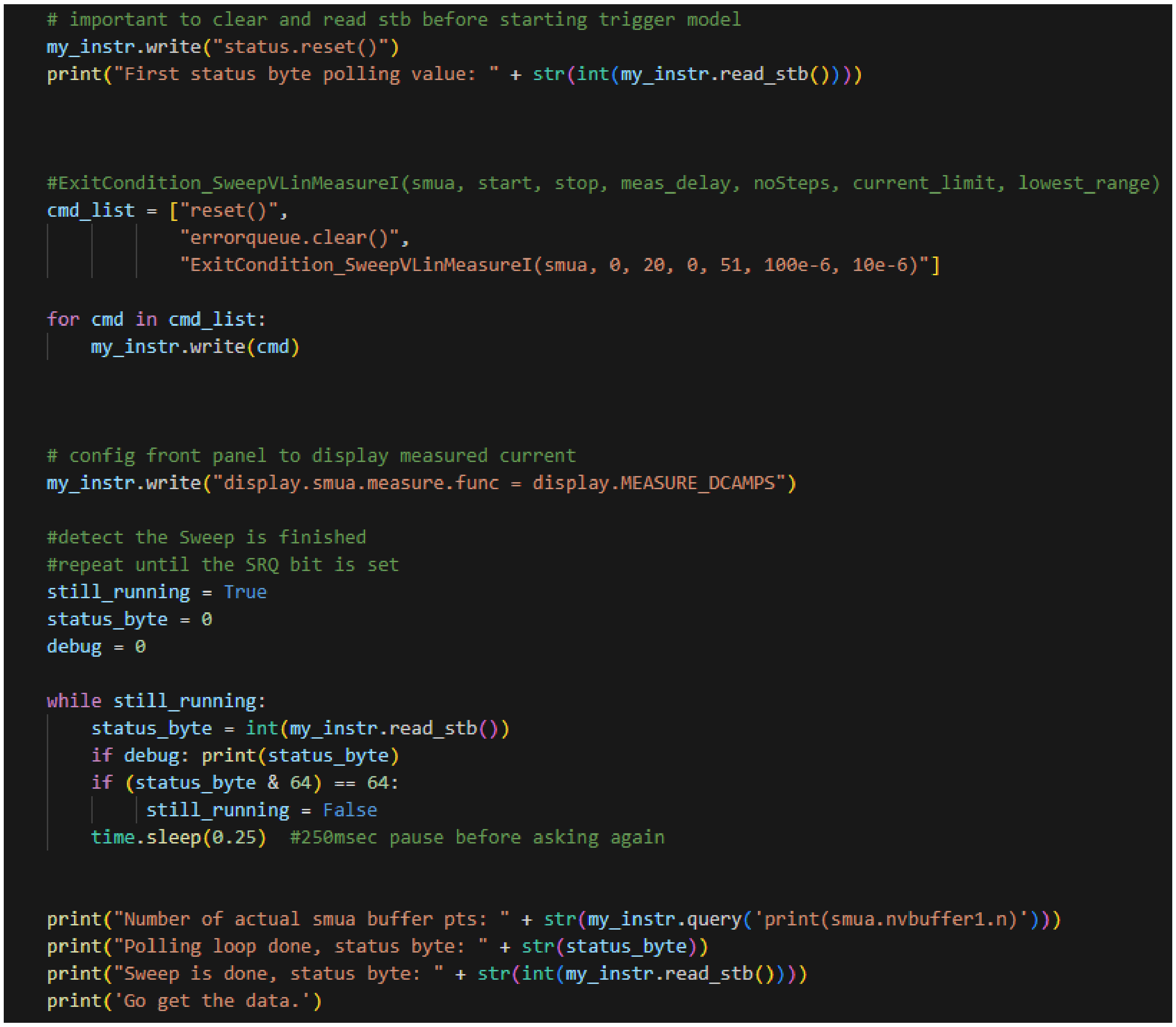
```python
file_path = "C:\\Users\\aclary\\Keithley Test Script Builder\\Workspaces\\workspace\\_0_Cool_new_Project\\"
file_name = "Abort_Sweep_on_Compliance_functions.tsp"
file_path_and_name = file_path + file_name

print(file_path_and_name)

my_instr.write("loadscript myWorkers")
with open(file_path_and_name) as fp:
    #read the TSP function definition file line by line
    for line in fp:
        #print(line)
        #write the TSP command to the instrument, line by line
        my_instr.write(line)

my_instr.write("endscript")
#run the script to place it into runtime memory
my_instr.write("myWorkers.run()")

#close the script file
fp.close()

print('done loading functions')
```

From Python, call your functions

In Test Script Builder or TSP Toolkit:
 	
This was our pseudo code to test the functions
	
	Abort_Sweep_on_Compliance_functions.tsp ⊠ run_my_functions.tsp ⊠
	`reset()`
	`errorqueue.clear()`
	
	`-- **************  call the functions`
	`--ExitCondition_SweepVLinMeasureI(smu, startv, stopv, stime, points, current_limit)`
	`    ExitCondition_SweepVLinMeasureI(smua, 0, 20, 0, 41, 0.25)`
	`    printData(smua)`

In Python	
	`#ExitCondition_SweepVLinMeasureI(smua, start, stop, meas_delay, noSteps, current_limit)`
	`cmd_list = ["reset()",`
	`            "errorqueue.clear()",`
	`            "ExitCondition_SweepVLinMeasureI(smua, 0, 20, 0, 41, 0.25)"]`
	
	`for cmd in cmd_list:`
	`    my_instr.write(cmd)`

Execute the function and detect when it is done:

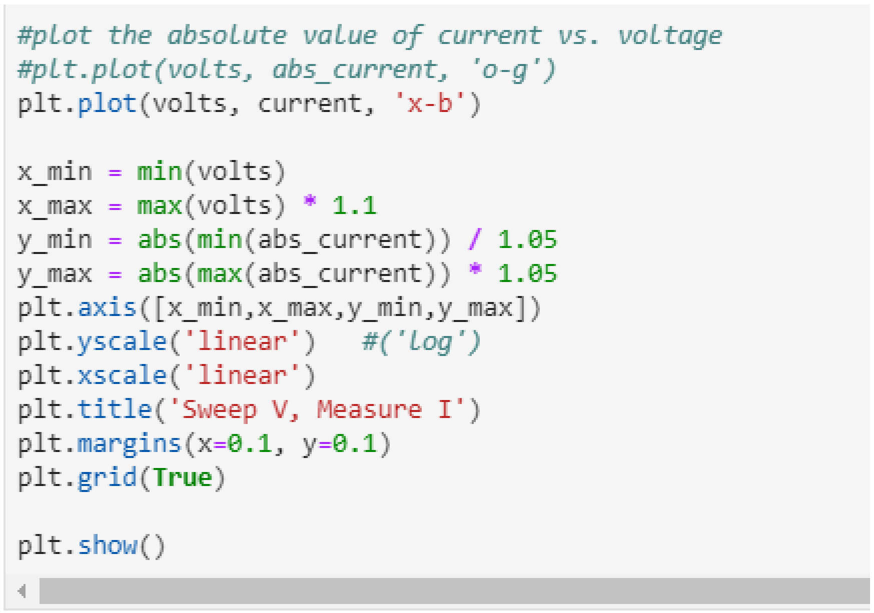


```python
# important to clear and read stb before starting trigger model
my_instr.write("status.reset()")
print("First status byte polling value: " + str(int(my_instr.read_stb())))



#ExitCondition_SweepVLinMeasureI(smua, start, stop, meas_delay, noSteps, current_limit, lowest_range)
cmd_list = ["reset()",
            "errorqueue.clear()",
            "ExitCondition_SweepVLinMeasureI(smua, 0, 20, 0, 51, 100e-6, 10e-6)"]

for cmd in cmd_list:
    my_instr.write(cmd)



# config front panel to display measured current
my_instr.write("display.smua.measure.func = display.MEASURE_DCAMPS")

#detect the Sweep is finished
#repeat until the SRQ bit is set
still_running = True
status_byte = 0
debug = 0

while still_running:
    status_byte = int(my_instr.read_stb())
    if debug: print(status_byte)
    if (status_byte & 64) == 64:
         still_running = False
    time.sleep(0.25)  #250msec pause before asking again


print("Number of actual smua buffer pts: " + str(my_instr.query('print(smua.nvbuffer1.n)')))
print("Polling loop done, status byte: " + str(status_byte))
print("Sweep is done, status byte: " + str(int(my_instr.read_stb())))
print('Go get the data.')
```

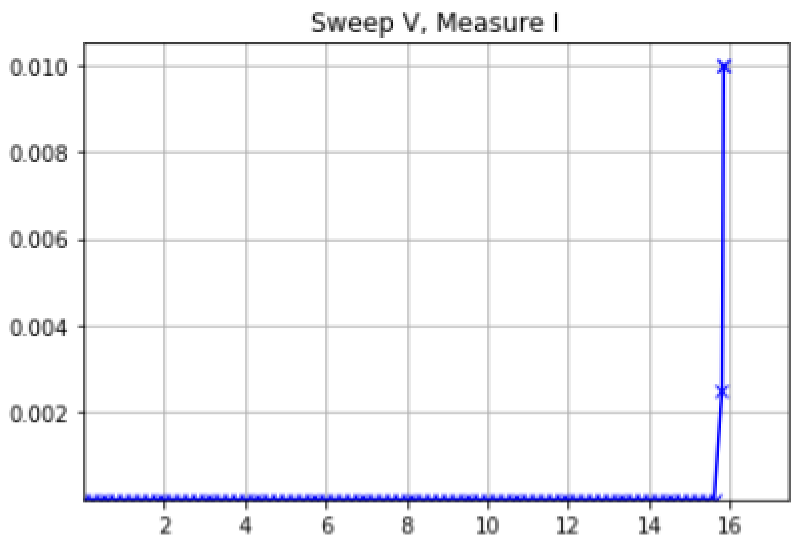
Read the data back

```python
import numpy as np
import matplotlib.pyplot as plt

#ask for voltage and current; buffer2=voltage
my_instr.write("printbuffer(1, smua.nvbuffer1.n, smua.nvbuffer2.readings, smua.nvbuffer1.readings)")
#one long comma delimited string
raw_data = my_instr.read()

# an array of strings alternating voltage, current, voltage, current
raw_data_array = raw_data.split(",")

volts = []
current = []
abs_current = []

# step through the array of strings, step size 2
# place the V and I into their own array of floats
for i in range(0, len(raw_data_array),2):
    volts.append(float(raw_data_array[i]))
    current.append(float(raw_data_array[i+1]))
    #absolute value of currents for log scale
    abs_current.append(abs(float(raw_data_array[i+1])))
```

Graph the Data:



```
#plot the absolute value of current vs. voltage
#plt.plot(volts, abs_current, 'o-g')
plt.plot(volts, current, 'x-b')

x_min = min(volts)
x_max = max(volts) * 1.1
y_min = abs(min(abs_current)) / 1.05
y_max = abs(max(abs_current)) * 1.05
plt.axis([x_min,x_max,y_min,y_max])
plt.yscale('linear')   #('log')
plt.xscale('linear')
plt.title('Sweep V, Measure I')
plt.margins(x=0.1, y=0.1)
plt.grid(True)

plt.show()
```



Full TSP code listing for function definition at end of this document.

Full TSP Code Listing for 2600B Sweep Function

```
--[[

     Tested with:  2602B, firmware 3.3.5

     Purpose:  Start with KISweep based SweepVLinMeasureI factory function.
               Modify for monitoring of current limit and then self-abort.

               Load these functions into memory of instrument.
               Then call them:

               ExitCondition_SweepVLinMeasureI(smu, startv, stopv, stime,
points, current_limit)



]]--

function ExitCondition_SweepVLinMeasureI(smu, startv, stopv, stime, points,
current_limit)
    -- Default to smua if no smu is specified.
    if smu == nil then
        smu = smua
    end

    -- Save settings in temporary variables so they can be restored at the
end.
    local l_s_levelv = smu.source.levelv
    local l_s_rangev = smu.source.rangev
    local l_s_autorangev = smu.source.autorangev
    local l_s_func = smu.source.func
    local l_m_autozero = smu.measure.autozero
    local l_d_screen = display.screen

    -- Clear the front panel display then prompt for input parameters if
missing.
    display.clear()
    if startv == nil then
        startv = display.prompt(l_volts_fmt, " Volts", "Enter START
voltage.", -1, -l_max_volts, l_max_volts)
        if startv == nil then
            -- Abort if Exit key pressed
            AbortScript(l_d_screen)
        end
    end
    if stopv == nil then
        stopv = display.prompt(l_volts_fmt, " Volts", "Enter STOP voltage.",
1, -l_max_volts, l_max_volts)
        if stopv == nil then
            -- Abort if Exit key pressed
            AbortScript(l_d_screen)
        end
    end
    if stime == nil then
        stime = display.prompt("+0.000E+00", " Seconds", "Enter SETTLING
time.", 0, 0, 10)
```

```
        if stime == nil then
            -- Abort if Exit key pressed
            AbortScript(l_d_screen)
        end
    end
    if points == nil then
        points = display.prompt("0000", " Points", "Enter number of sweep 
POINTS.", 10, 1, 1000)
        if points == nil then
            -- Abort if Exit key pressed
            AbortScript(l_d_screen)
        end
    end

    -- Update display with test info.
    display.settext("SweepV exit on I_LIMIT")  -- Line 1 (20 characters max)

    -- Configure source and measure settings.
    smu.source.output = smu.OUTPUT_OFF
    smu.source.func = smu.OUTPUT_DCVOLTS
    smu.source.levelv = 0
    smu.source.limiti = current_limit
    smu.source.rangev = math.max(math.abs(startv), math.abs(stopv))


    smu.measure.autozero = smu.AUTOZERO_ONCE

    smu.measure.lowrangei = 100e-9   -- consider adding this if pA is not 
important



    -- Reset trigger model
    smu.trigger.arm.stimulus = 0
    smu.trigger.source.stimulus = 0
    smu.trigger.measure.stimulus = 0
    smu.trigger.endpulse.stimulus = 0
    smu.trigger.arm.count = 1
    -- Configure the source action
    smu.trigger.source.linearv(startv, stopv, points)
    smu.trigger.source.limiti = current_limit
    smu.trigger.source.action = smu.ENABLE
    smu.trigger.endpulse.action = smu.SOURCE_HOLD
    -- Configure the measure action
    smu.trigger.measure.iv(smu.nvbuffer1,smu.nvbuffer2)  -- measure both
    smu.trigger.measure.action = smu.ENABLE
    -- Configure the delay
    if (stime > 0) then
        trigger.timer[1].reset()
        trigger.timer[1].delay = stime
        smu.trigger.measure.stimulus = trigger.timer[1].EVENT_ID
        trigger.timer[1].stimulus = smu.trigger.SOURCE_COMPLETE_EVENT_ID
    end
    -- Configure the sweep count
    smu.trigger.count = points
```

```
 -- ********* if config does not change, only the code from here down needs
called
 -- ********* consider chopping this function into some smaller pieces.

    -- Run the sweep and then turn the output off.

    -- clear any old data
    config_buffers(smu)

    -- get our status subsystem ready
    config_sweep_complete_notify()

    smu.source.output = smu.OUTPUT_ON
    smu.trigger.initiate()

    --waitcomplete()   this blocks

    -- replace with a custom version that looks for current limit or sweep
done
    --my_wait_complete(smu, loopDelay)
    my_wait_complete(smu, 0.05)

    smu.source.output = smu.OUTPUT_OFF



    -- Update the front panel display and restore modified settings.
    display.setcursor(2,1)
    display.settext("Test complete.")     -- Line 2 (32 characters max)
    smu.source.levelv = 0
    smu.source.rangev = l_s_rangev
    smu.source.autorangev = l_s_autorangev
    smu.source.func = l_s_func
    smu.source.levelv = l_s_levelv
    smu.measure.autozero = l_m_autozero
    delay(2)
    display.clear()
    display.screen = l_d_screen
end   -- function


function my_wait_complete(smu, loopDelay)

      repeat
            delay(loopDelay)

            if (bit.test(status.measurement.current_limit.condition, 2)) ==
true then
                  smu.abort()
                  -- add cmds to take smu to known state
                  -- set source level, etc.
                  smu.source.levelv = 0
                  smu.measure.i()  -- new measurement to clear current_limit
bit
            end  -- if
            if debug == 1 then print("Abort due to compliance detect") end
      until bit.test(status.condition, 8) == true  -- until sweeping bit
falling edge

end  -- function

function config_sweep_complete_notify()

       status.reset()

       -- config assert SRQ when SWEEPING bit changes high to low
       status.operation.sweeping.enable = status.operation.sweeping.SMUA
     status.operation.enable = status.operation.SWEEPING
     -- transistion registers
     status.operation.sweeping.ptr = 0
     status.operation.sweeping.ntr = status.operation.sweeping.SMUA
     -- when negative transition register changes state
     -- from 0 to 1, the sweep is done.

     status.node_enable = status.OSB
     status.request_enable = status.OSB


     -- also config to assert SRQ if SMUA current limit occurs
     -- Enable current limit bit in current limit register.
            status.measurement.current_limit.enable =
status.measurement.current_limit.SMUA
            -- Enable status measure current limit bit.
            status.measurement.enable = status.measurement.ILMT

            status.measurement.current_limit.ptr =
status.measurement.current_limit.SMUA
            status.measurement.current_limit.ntr = 0

            -- Set system summary; enable MSB......and retain the SWEEPING
OSB SRQ too
            status.node_enable = status.MSB + status.OSB
            -- Enable status SRQ MSB.
            status.request_enable = status.MSB + status.OSB



end  -- function

function config_buffers(smu)

            -- Setup a buffer to store the result(s) in and start testing.
          smu.nvbuffer1.clear()
          smu.nvbuffer1.appendmode = 1
          smu.nvbuffer1.collecttimestamps = 1
          smu.nvbuffer1.collectsourcevalues = 1

          -- do the voltage buffer too just in case
          smu.nvbuffer2.clear()
          smu.nvbuffer2.appendmode = 1
          smu.nvbuffer2.collecttimestamps = 1
          smu.nvbuffer2.collectsourcevalues = 1
```

```
end  -- function

function printData(smu)

     -- Print the data back to the Console in tabular format
      --print("Time\tVoltage\tCurrent")
      for x=1,smu.nvbuffer1.n do
            -- Voltage readings are in nvbuffer2.  Current readings are in
nvbuffer1.
            print(smu.nvbuffer1.timestamps[x], smu.nvbuffer1.sourcevalues[x],
smu.nvbuffer1[x])
      end

end  -- function
```