# IV Sweep GUI Documentation

**1. Installation and Important Notes**

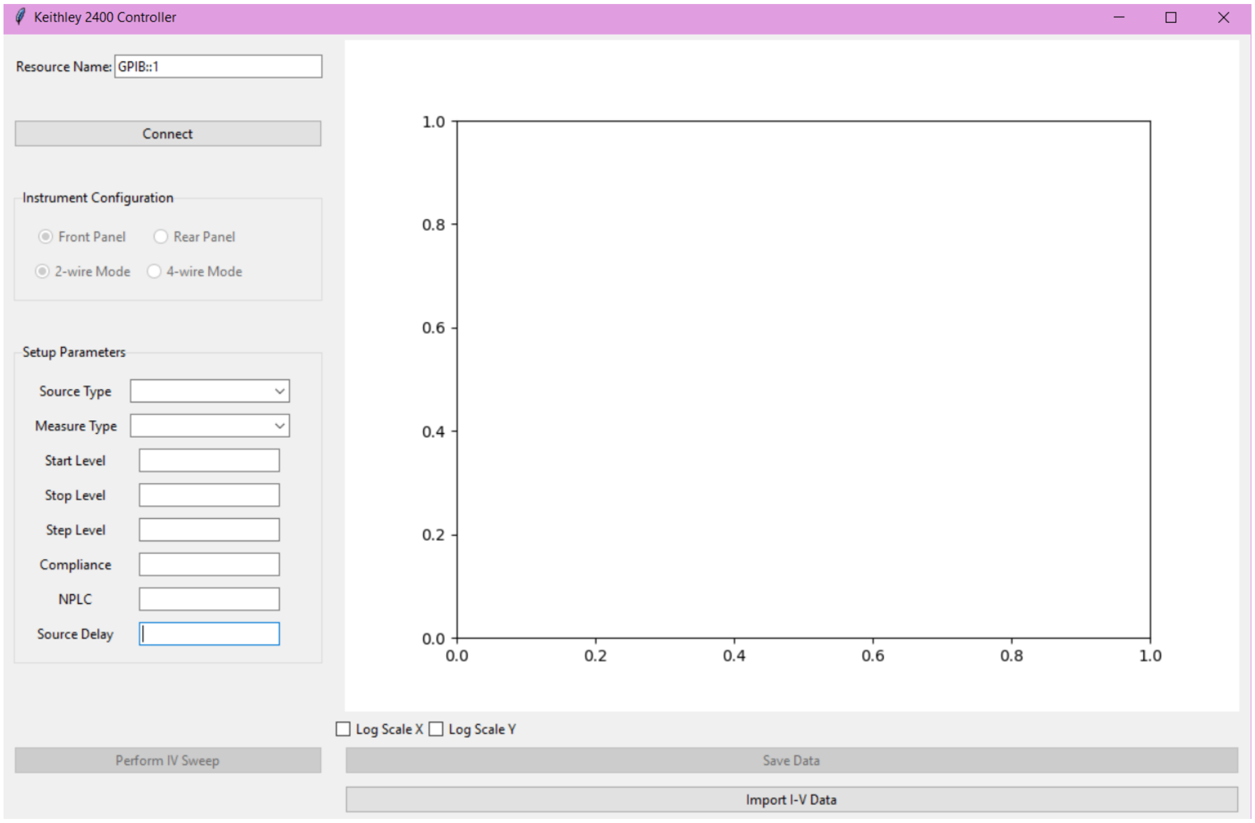
**2. Inputs**

**3. Measuring and Visualizing Data**

**4. Saving and Importing Data**

**Appendix A: IV Sweep GUI Python code**

**Appendix B: SMU Port ID**



Keithley 2400 Controller

Resource Name: GPIB::1

Instrument Configuration

● Front Panel ○ Rear Panel

● 2-wire Mode ○ 4-wire Mode

## 1. Installation and Important Notes

The current distribution of this program was designed to work on a Windows 7 platform, mainly on the PC connected to the Keithley 2400 Sourcemeter Unit (SMU) in JHE 322. It will still work on newer versions of Windows, however, will not run on Mac OS or Linux. Support for Mac OS can be created in the future.

Simply download the **IV Sweep GUI.exe** into the desired location and click to run (it may take a few moments to boot up). It is executable on a USB or external hard drive. Ensure the laptop/PC being used is connected to the SMU. All input definitions are explained in the next section.

[1] For the GUI to connect and send data to the SMU, the correct port ID must be stated in the *Resource Name* input field, as will be described below. If the SMU is connected to a different PC or via a different cable connection, this name must be correctly changed to the corresponding port. That requires running a separate python code to identify the ports and installation of the PyMeasure scientific package, explained in Appendix B. A function to detect and display connections within the GUI will be added in future versions, if this becomes necessary.

# 2. Inputs

1. **Resource Name:**

*Text Field*

The default is set to "GPIB::1", configured for use with the current PC and connection in JHE 322. Only change if the SMU is connected via a different port[1].

## 2.1 Instrument Configuration

1. **Front Panel/Rear Panel:**

*Select Field: Front Panel/Rear Panel*

Select where the wires/probes are connected on the SMU (front panel or rear panel).

2. **2-Wire Mode/4-Wire Mode:**

*Select Field: 2-Wire Mode/4-Wire Mode*

Select how many wire/probes are connected to the SMU and being used for the measurement.

## 2.2 Setup Parameters

1. **Source Type:**

*Drop Down Menu: VOLT/CURR*

Select the desired property (voltage or current) to be produced by the sourcemeter.

2. **Measure Type:**

*Drop Down Menu: VOLT/CURR*

Select the desired property (voltage or current) to be measured by the sourcemeter.

3. **Start Level:**

*Number (Floating Point):*

Input the desired voltage [V] or current [A] sweep start value.

4. **Stop Level:**

*Number (Floating Point):*

Input the desired voltage [V] or current [A] sweep stop value.

5. **Step Level**

*Number Field (Integer):*

Input the desired number of measurements to be taken.

6. **Compliance:**

*Number (Floating Point):*

Input the desired voltage [V] or current [A] compliance value.

The compliance value sets the current limit when sourcing voltage and sets the voltage limit when sourcing current. The sourcemeter output will not exceed this compliance value.

7. **NPLC (Number of power line cycles):**

*Number (Floating Point):*

Input the desired NPLC value.

NPLC controls the integration time for measurements, which affects the measurement speed and noise reduction. A higher NPLC value results in more accurate measurements but a longer measurement time. Around 1 is typically sufficient for IV measurements. Some recommended values are given below:

SLOW ~ 10

MEDIUM ~ 1

FAST ~ 0.1

8. **Source Delay:**

*Number (Floating Point):*

Input the desired source delay value.

The source delay is the time between sourcing and measuring. It allows the circuit to settle before performing another measurement. Around 0 is typically fine for IV measurements, however it can be changed accordingly.

## 3. Measuring and Visualizing Data

After inputting the port ID into the *Resource Name* field, click on “Connect” to connect to the SMU. There will be a pop-up stating whether the connection was successful. If no SMU was detected, double check the port ID or any loose connections.

Select the correct instrument configuration options and input all required values. Click on “Perform IV Sweep” to begin the measurement. After the measurements are taken, the plot on the right will automatically update with the data points.

The x-axis on the plot is automatically set to be the property selected as the *Source Type* and the y-axis to be the *Measure Type* property. Two toggle fields, *Log Scale X* and *Log Scale Y* can be selected to log that respective axis data.

## 4. Saving and Importing Data

Data is currently only saved as a .txt. Click on the button “Save Data” after the measurement is completed and save the file to the desired location. The resulting file has two columns, the first representing the x-axis data (Source Type) and the second representing the y-axis data (Measure Type).

Lastly, .txt or .csv files can be imported to display on the plot. First, change the *Source Type* and *Measure Type* input fields to correspond to the x- and y- axis properties first, and then click on “Import I-V Data”. Select the desired file and the output should display on the plot.

# Appendix A: IV Sweep GUI Python code

```
# BACKEND
import pyvisa
class Keithley2400Controller:
    def __init__(self, resource_name='GPIB::1', timeout=25000):
        self.resource_name = resource_name
        self.instrument = None
        self.timeout = timeout
        self.current_compliance = 0.01  # 10 mA default

    # Connect to Keithley 2400
    def connect(self):
        rm = pyvisa.ResourceManager()
        self.instrument = rm.open_resource(self.resource_name)
        self.instrument.timeout = self.timeout  # Set timeout
        self.instrument.write("*RST")  # Reset the instrument
        self.instrument.write("*CLS")  # Clear the status

    def identify(self):
        return self.instrument.query("*IDN?")

    def select_panel(self, panel='FRONT'):
        if panel.upper() == 'FRONT':
            self.instrument.write(":ROUT:TERM FRON")
        elif panel.upper() == 'REAR':
            self.instrument.write(":ROUT:TERM REAR")
        else:
            raise ValueError("Invalid panel option. Choose 'FRONT' or 'REAR'.")

    def set_measurement_mode(self, mode):
        if mode == 2:
            self.instrument.write(":SYST:RSEN OFF")  # 2-wire mode
        elif mode == 4:
            self.instrument.write(":SYST:RSEN ON")  # 4-wire mode
        else:
            raise ValueError("Invalid measurement mode. Choose 2 or 4.")

    def iv_sweep(self, source_type, measure_type, start_level, stop_level, step_level,
                 measure_compliance, nplc=1, source_delay=0.1, ovp=20):
        # Disable concurrent functions
        self.instrument.write(":SENS:FUNC:CONC OFF")

        # Calculate number of points for the sweep
        num_points = int(abs((stop_level - start_level)) / abs(step_level)) + 1

        # Set Over Voltage Protection (always present)
        self.instrument.write(f":SOUR:VOLT:PROT {ovp}")

        # Set source function and enable auto-range
        self.instrument.write(f":SOUR:FUNC {source_type.upper()}")
        if source_type.upper() == 'CURR':
            self.instrument.write(":SOUR:CURR:RANG:AUTO ON")
        else:  # 'VOLT'
            self.instrument.write(":SOUR:VOLT:RANG:AUTO ON")
```

```
# Set sense function, enable auto-range, and set compliance
self.instrument.write(f":SENS:FUNC '{measure_type.upper()}:DC'")
if measure_type.upper() == 'CURR':
    self.instrument.write(f":SENS:CURR:PROT {measure_compliance}")
    self.instrument.write(":SENS:CURR:RANG:AUTO ON")
else:  # 'VOLT'
    self.instrument.write(f":SENS:VOLT:PROT {measure_compliance}")
    self.instrument.write(":SENS:VOLT:RANG:AUTO ON")

# Configure source for sweep
self.instrument.write(f":SOUR:{source_type.upper()}:START {start_level}")
self.instrument.write(f":SOUR:{source_type.upper()}:STOP {stop_level}")
self.instrument.write(f":SOUR:{source_type.upper()}:STEP {step_level}")
self.instrument.write(f":SOUR:{source_type.upper()}:MODE SWE")
self.instrument.write(":SOUR:SWE:RANG AUTO")
self.instrument.write(":SOUR:SWE:SPAC LIN")

# Set NPLC, trigger count and source delay
self.set_nplc(nplc, measure_type)
self.instrument.write(f":TRIG:COUN {num_points}")
self.instrument.write(f":SOUR:DEL {source_delay}")

# Enable output and initiate measurement
self.instrument.write(":OUTP ON")
raw_data = self.instrument.query_ascii_values(":READ?")
```

```
        # Disable output after measurement
        self.instrument.write(":OUTP OFF")

        voltage = [raw_data[i] for i in range(0, len(raw_data), 5)]
        current = [raw_data[i + 1] for i in range(0, len(raw_data), 5)]

        return voltage, current

    def set_source_current_range(self, range_value):
        self.instrument.write(f":SOUR:CURR:RANG {range_value}")

    def set_source_voltage_range(self, range_value):
        self.instrument.write(f":SOUR:VOLT:RANG {range_value}")

    def set_measure_current_range(self, range_value):
        self.instrument.write(f":SENS:CURR:RANG {range_value}")

    def set_measure_voltage_range(self, range_value):
        self.instrument.write(f":SENS:VOLT:RANG {range_value}")

    def set_current_compliance(self, compliance):
        self.current_compliance = compliance
        self.instrument.write(f":SENS:CURR:PROT {compliance}")

    def set_nplc(self, nplc, measurement_type='CURR'):
        if measurement_type.upper() == 'CURR':
            self.instrument.write(f":SENS:CURR:NPLC {nplc}")
        elif measurement_type.upper() == 'VOLT':
            self.instrument.write(f":SENS:VOLT:NPLC {nplc}")
        else:
            raise ValueError("Invalid measurement type. Choose 'CURR' or 'VOLT'.")
# GUI
import tkinter as tk
from tkinter import ttk, messagebox, filedialog
import threading
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
import numpy as np
import csv


class KeithleyGUI:
    def __init__(self, master):
        self.master = master
        master.title('Keithley 2400 Controller')

        self.is_connected = False  # Track connection status
        self.data_collected = False  # Track if data has been collected

        # Initialize the instrument controller
        self.instrument = None  # Initialize without creating an object yet

        # Resource name field
        self.frame_resource = ttk.Frame(master)
        self.frame_resource.grid(row=0, column=0, padx=10, pady=5, sticky='ew')

        ttk.Label(self.frame_resource, text="Resource Name:").pack(side=tk.LEFT)

        self.resource_name_entry = ttk.Entry(self.frame_resource, width=30)

        self.resource_name_entry.pack(side=tk.LEFT, fill=tk.X, expand=True)

        self.resource_name_entry.insert(0, 'GPIB::1')



        # Connection Frame

        self.frame_connection = ttk.Frame(master)

        self.frame_connection.grid(row=1, column=0, padx=10, pady=5, sticky='ew')

        self.connect_button = ttk.Button(self.frame_connection, text="Connect",
command=self.connect_instrument)

        self.connect_button.pack(fill=tk.X)



        # Configuration Frame (Panel and Measurement Mode)

        self.frame_config = ttk.LabelFrame(master, text="Instrument Configuration", padding=(10,
10))

        self.frame_config.grid(row=2, column=0, padx=10, pady=5, sticky='ew')

        self.panel_var = tk.BooleanVar(value=True)

        self.front_panel_radio = ttk.Radiobutton(self.frame_config, text='Front Panel',
variable=self.panel_var, value=True, state='disabled', command=self.change_panel)

        self.front_panel_radio.grid(row=0, column=0, padx=5, pady=5)

        self.rear_panel_radio = ttk.Radiobutton(self.frame_config, text='Rear Panel',
variable=self.panel_var, value=False, state='disabled', command=self.change_panel)

        self.rear_panel_radio.grid(row=0, column=1, padx=5, pady=5)



        self.mode_var = tk.BooleanVar(value=False)

        self.two_wire_radio = ttk.Radiobutton(self.frame_config, text='2-wire Mode',
variable=self.mode_var, value=False, state='disabled', command=self.change_mode)

        self.two_wire_radio.grid(row=1, column=0, padx=5, pady=5)

        self.four_wire_radio = ttk.Radiobutton(self.frame_config, text='4-wire Mode',
variable=self.mode_var, value=True, state='disabled', command=self.change_mode)
```

```
        self.four_wire_radio.grid(row=1, column=1, padx=5, pady=5)


        # Setup Frame
        self.frame_setup = ttk.LabelFrame(master, text="Setup Parameters", padding=(10, 10))
        self.frame_setup.grid(row=3, column=0, padx=10, pady=5, sticky='ew')


        # Setup fields
        self.setup_fields = {
            'Source Type': ttk.Combobox(self.frame_setup, values=['VOLT', 'CURR'],
state="readonly"),
            'Measure Type': ttk.Combobox(self.frame_setup, values=['VOLT', 'CURR'],
state="readonly"),
            'Start Level': ttk.Entry(self.frame_setup),
            'Stop Level': ttk.Entry(self.frame_setup),
            'Step Level': ttk.Entry(self.frame_setup),
            'Compliance': ttk.Entry(self.frame_setup),
            'NPLC': ttk.Entry(self.frame_setup),
            'Source Delay': ttk.Entry(self.frame_setup),
        }
        for i, (label, widget) in enumerate(self.setup_fields.items()):
            ttk.Label(self.frame_setup, text=label).grid(row=i, column=0, padx=5, pady=5)
            widget.grid(row=i, column=1, padx=5, pady=5)


        # Plot Area
        self.figure = Figure(figsize=(8, 6), dpi=100)
        self.plot = self.figure.add_subplot(111)
        self.canvas = FigureCanvasTkAgg(self.figure, master)
        self.canvas.get_tk_widget().grid(row=0, column=1, rowspan=5, padx=10, pady=5)
        # Log Scale Toggles

        self.log_scale_x = tk.BooleanVar()

        self.log_scale_y = tk.BooleanVar()

        self.log_frame = ttk.Frame(master)

        self.log_frame.grid(row=5, column=1, sticky='ew')

        ttk.Checkbutton(self.log_frame, text='Log Scale X', variable=self.log_scale_x,
command=self.update_plot).pack(side=tk.LEFT)

        ttk.Checkbutton(self.log_frame, text='Log Scale Y', variable=self.log_scale_y,
command=self.update_plot).pack(side=tk.LEFT)



        # IV Sweep Button

        self.sweep_button = ttk.Button(master, text="Perform IV Sweep",
command=self.perform_iv_sweep)

        self.sweep_button.grid(row=6, column=0, padx=10, pady=5, sticky='ew')



        # Save Data Button

        self.save_button = ttk.Button(master, text="Save Data", command=self.save_data)

        self.save_button.grid(row=6, column=1, padx=10, pady=5, sticky='ew')



        # Import Data Button (New functionality)

        self.import_button = ttk.Button(master, text="Import I-V Data", command=self.import_data)

        self.import_button.grid(row=7, column=1, padx=10, pady=5, sticky='ew')



        self.update_button_states()



    def update_button_states(self):

        if self.is_connected:

            self.connect_button.config(state='disabled')
            self.sweep_button.config(state='normal')
            self.save_button.config(state='normal' if self.data_collected else 'disabled')
            self.front_panel_radio.config(state='normal')
            self.rear_panel_radio.config(state='normal')
            self.two_wire_radio.config(state='normal')
            self.four_wire_radio.config(state='normal')
        else:
            self.connect_button.config(state='normal')
            self.sweep_button.config(state='disabled')
            self.save_button.config(state='disabled')
            self.front_panel_radio.config(state='disabled')
            self.rear_panel_radio.config(state='disabled')
            self.two_wire_radio.config(state='disabled')
            self.four_wire_radio.config(state='disabled')

    def connect_instrument(self):
        resource_name = self.resource_name_entry.get()
        try:
            self.instrument = Keithley2400Controller(resource_name)
            self.instrument.connect()
            self.is_connected = True
            messagebox.showinfo("Connection", "Successfully connected to the instrument.")
        except Exception as e:
            messagebox.showerror("Connection Failed", str(e))
            self.is_connected = False
        finally:
            self.update_button_states()
```

```
def change_panel(self):
    if not self.is_connected:
        messagebox.showerror("Error", "Instrument is not connected.")
        return
    panel = 'REAR' if not self.panel_var.get() else 'FRONT'
    self.instrument.select_panel(panel)

def change_mode(self):
    if not self.is_connected:
        messagebox.showerror("Error", "Instrument is not connected.")
        return
    mode = 4 if self.mode_var.get() else 2
    self.instrument.set_measurement_mode(mode)

def perform_iv_sweep(self):
    if not self.is_connected:
        messagebox.showerror("Error", "Instrument is not connected.")
        return

    # Gather the setup parameters
    source_type = self.setup_fields['Source Type'].get()
    measure_type = self.setup_fields['Measure Type'].get()
    start_level = float(self.setup_fields['Start Level'].get())
    stop_level = float(self.setup_fields['Stop Level'].get())
    step_level = float(self.setup_fields['Step Level'].get())
    compliance = float(self.setup_fields['Compliance'].get())
```

```
        nplc = float(self.setup_fields['NPLC'].get())
        source_delay = float(self.setup_fields['Source Delay'].get())

        # Start the IV sweep in a separate thread
        threading.Thread(target=self.async_iv_sweep, args=(
            source_type, measure_type, start_level, stop_level, step_level,
            compliance, nplc, source_delay
        )).start()

    def async_iv_sweep(self, source_type, measure_type, start_level, stop_level, step_level, compliance, nplc, source_delay):
        try:
            # Execute the IV sweep without range parameters
            self.voltage, self.current = self.instrument.iv_sweep(
                source_type, measure_type, start_level, stop_level, step_level,
                compliance, nplc, source_delay
            )
            # Update the plot
            self.master.after(0, self.update_plot)
            self.data_collected = True  # Data has been collected
        except Exception as e:
            messagebox.showerror("Error", str(e))
        finally:
            self.master.after(0, self.update_button_states)  # Update GUI elements from the main thread
    def update_plot(self):
        # Determine what is being sourced and measured
        source_type = self.setup_fields['Source Type'].get()
```

```
measure_type = self.setup_fields['Measure Type'].get()
if source_type == 'VOLT' and measure_type == 'CURR':
    x_data, y_data = self.voltage, self.current
    x_label, y_label = 'Voltage (V)', 'Current (A)'
else:
    x_data, y_data = self.current, self.voltage
    x_label, y_label = 'Current (A)', 'Voltage (V)'

self.plot.clear()
# Apply log scale if selected and adjust data to absolute values for log scale
if self.log_scale_x.get():
    x_data = np.abs(x_data)
    self.plot.set_xscale('log')
else:
    self.plot.set_xscale('linear')

if self.log_scale_y.get():
    y_data = np.abs(y_data)
    self.plot.set_yscale('log')
else:
    self.plot.set_yscale('linear')

self.plot.plot(x_data, y_data, marker='o', linestyle='-')
self.plot.set_xlabel(x_label)
self.plot.set_ylabel(y_label)
self.plot.set_title('IV Sweep Results')
```

```
        # Set formatter for automatic scientific notation
        # self.plot.xaxis.set_major_formatter(ticker.ScalarFormatter(useMathText=True))
        # self.plot.ticklabel_format(style='sci', axis='x', scilimits=(0,0), useOffset=False)
        plt.setp(self.plot.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")
        # Update the plot
        self.canvas.draw()

    def save_data(self):
        if not self.data_collected:
            messagebox.showerror("Error", "No data available to save.")
            return
        filepath = filedialog.asksaveasfilename(defaultextension=".txt", filetypes=[("Text Files",
"*.txt")])
        if filepath:
            with open(filepath, 'w', newline='') as file:
                writer = csv.writer(file, delimiter='\t')
                writer.writerow(["Voltage", "Current"])
                for v, c in zip(self.voltage, self.current):
                    writer.writerow([v, c])
            messagebox.showinfo("Save File", "Data saved successfully.")

    def toggle_autorange(self):
        state = 'disabled' if self.autorange.get() else 'normal'
        self.setup_fields['Source Range'].config(state=state)
        self.setup_fields['Measure Range'].config(state=state)

    def import_data(self):
        filepath = filedialog.askopenfilename(filetypes=[("CSV files", "*.csv"), ("Text files", "*.txt")])
        if not filepath:
            return

        voltage, current = [], []
        try:
            with open(filepath, 'r') as file:
                reader = csv.reader(file, delimiter='\t' if filepath.endswith('.txt') else ',')
                next(reader)  # Skip header row
                for row in reader:
                    voltage.append(float(row[0]))
                    current.append(float(row[1]))
        except Exception as e:
            messagebox.showerror("Error", f"Failed to read file: {e}")
            return

        self.voltage, self.current = np.array(voltage), np.array(current)
        self.data_collected = True
        self.update_plot()

if __name__ == "__main__":
    root = tk.Tk()
    app = KeithleyGUI(root)
    root.mainloop()
```

# Appendix B: SMU Port ID

```
from pymeasure.instruments.resources import list_resources

list_resources()
```

The snippet above is the code required to identify all connections to the PC. It requires the package PyMeasure. The output will list all connections like below:

```
0 : ASRL2::INSTR : Not known

1 : ASRL10::INSTR : Not known

2 : GPIB0::1::INSTR : KEITHLEY INSTRUMENTS INC.,MODEL 2400,1248014,C30   Mar 17 2006 09:29:29/A02
/K/J

Out[1]:

('ASRL2::INSTR', 'ASRL10::INSTR', 'GPIB0::1::INSTR')
```

Input the correct port into the *Resource Name* field.