# *Python Files for Explaining the Interface*

## Main Interface File: interface.py

The main interface file is interface.py. This file serves as the central interface and uses the DMMCommands class to communicate with the DMM (Digital Multimeter). Various measurements can be performed, stopped, and saved.

In the file A34110\_commands.py, the DMMCommands class is defined, which maps various commands and functions of the Agilent A34110 DMM. With its help, measurements can be configured, calibrations performed, and data retrieved.

## Explanation of interface.py

### 1. Importing Required Libraries

* import pyvisa: To use the PyVISA library.
* import time: To use time-related functions.
* import thread: To stop measurements while they are running.
* from A34110\_commands import DMMCommands: To use measurement and other functions.

### 2. Initializing Resource Manager

The PyVISA ResourceManager is initialized to manage communication with connected instruments:

rm = pyvisa.ResourceManager()

### 3. Listing Available Resources

All available resources (connected instruments) are listed:

resources = rm.list\_resources()

print("Available resources:", resources)

### 4. Assigning a Resource

A resource is assigned to a variable for later use with the instrument:

resource = resources[0]

### 5. Establishing Communication

An attempt is made to establish communication with the resource. If communication fails, an error is displayed. Additionally, a DMMCommands object is initialized for the instrument:

try:

instrument = rm.open\_resource(resource)

dmm = DMMCommands(instrument)

except Exception as e:

print(f"Could not communicate with {resource}: {e}")

### 6. Performing Basic Device Operations

The script then queries the identity of the instrument, clears its status, resets it, and performs a zero calibration:

print(f"Instrument identity: {dmm.get\_identity()}")

time.sleep(1)

dmm.clear\_status()

print("Status cleared")

time.sleep(1)

dmm.reset()

print("Instrument reset")

time.sleep(1)

dmm.zero\_calibration()

print("Instrument calibrated")

time.sleep(1)

### 7. Defining start\_measurement(command) Function

A function start\_measurement(command) is defined to start a measurement in a separate thread based on the user’s input command (e.g., "DC" for DC Voltage). The script prompts the user to enter a command to start a measurement (e.g., DC, AC, CURR\_DC, etc.) or perform other operations such as CALIBRATE, STOP, or EXIT.

### 8. Threaded Measurement Execution

Measurement operations are executed in separate threads to enable continuous measurements without blocking the main program loop. If a measurement is already running, the user is asked to stop it before starting a new one.

### 9. Calibration Process

If the user enters the command "CALIBRATE," the program asks for the type of calibration to be performed (e.g., DC, AC, RES, etc.). User input is converted to uppercase for consistency and passed to the calibration method of the DMM object.

### 10. Stopping Measurements

If the user enters the command "STOP," the current measurement thread is stopped, and the user is asked whether to save the collected data. If the user chooses to save, the data is written to a file using dmm.save\_measurements(filename).

### 11. Exiting the Program

If the user enters the command "EXIT," the program displays an exit message. If a measurement is still running, it is stopped, and the corresponding thread is terminated. The program then exits the while loop and terminates.

## Explanation of A34110\_commands.py

### Overview

This file defines the DMMCommands class, which consolidates all the functions and commands that can be executed with the Agilent A34110 DMM. Two main commands are used for communication with the measurement device:

#### Write Command

The write command is used to send a command or instruction to the device without expecting a direct response or data return. For instance, it can be used to configure the instrument or send a command to start a process.

#### Query Command

The query command sends a command to the device and immediately requests a response. It combines a write operation with a read operation. For example, it is used when measurements, status information, or other data need to be retrieved from the instrument.

### 1. Importing Required Libraries

* import time: To use time-related functions.
* import threading: To manage parallel processes, such as performing measurements while other processes are running.

### 2. Defining the DMMCommands Class

The DMMCommands class consolidates all operations for the Agilent A34110 DMM. It is initialized with an instrument object that establishes a connection to the DMM via PyVISA. A threading event \_stop\_event is created to control stopping measurements. The \_measurement\_data list stores collected measurement data, and the measurement\_type attribute stores the type of measurement being performed (e.g., "DC Voltage").

### 3. Basic Methods

* get\_identity(): Queries the device's identity (e.g., model and manufacturer).
* reset(): Resets the device to its factory settings.
* clear\_status(): Clears the device’s status registers.

### 4. Configuration Methods

Methods like configure\_dc\_voltage() and configure\_resistance() configure the device for specific measurement types.

### 5. Zero Calibration

A zero-calibration function performs a zero calibration of the instrument:

def zero\_calibration(self):

self.instrument.write('CAL:ZERO:ALL')

### 6. Measurement Methods

Functions such as measure\_dc\_voltage() and measure\_ac\_voltage() retrieve the current measurement value based on the configured measurement type.

### 7. Fetch Methods

* fetch\_data: Retrieves the last measured data.
* fetch\_status: Retrieves the current operational status of the device.

### 8. Measurement Control Methods

* initiate(): Starts a measurement or sequence.
* abort(): Stops a running measurement.
* stop\_measurement(): Sets an internal event (\_stop\_event) to end a loop or process in the software.

### 9. Calibration Process

The calibrate function is used to calibrate the DMM for a specific measurement type (e.g., DC, AC, RES).

### 10. Handling Measurements

Methods like handle\_dc\_voltage\_measurement() and handle\_resistance\_measurement() manage the entire process of starting a measurement, including setting the measurement\_type, calibrating the instrument, and starting the measurement loop.

### 11. Performing Measurements

The \_perform\_measurement() function measures data at a specified frequency and prints the data until the "stop" command is set.

### 12. Saving Measurements

The save\_measurements() function saves the collected measurement data to a file. The data is stored in a text format with a header indicating the measurement type and timestamps.

def save\_measurements(self, filename):

measurement\_type = self.measurement\_type if self.measurement\_type else "Measurement"

with open(filename, "w") as file:

file.write(f"Time,{measurement\_type}\n") # Header line with automatically determined measurement type

for timestamp, measurement in self.\_measurement\_data:

file.write(f"{timestamp} - {measurement.strip()}\n") # Write the timestamp and measurement

print(f"Measurements saved to {filename}.")