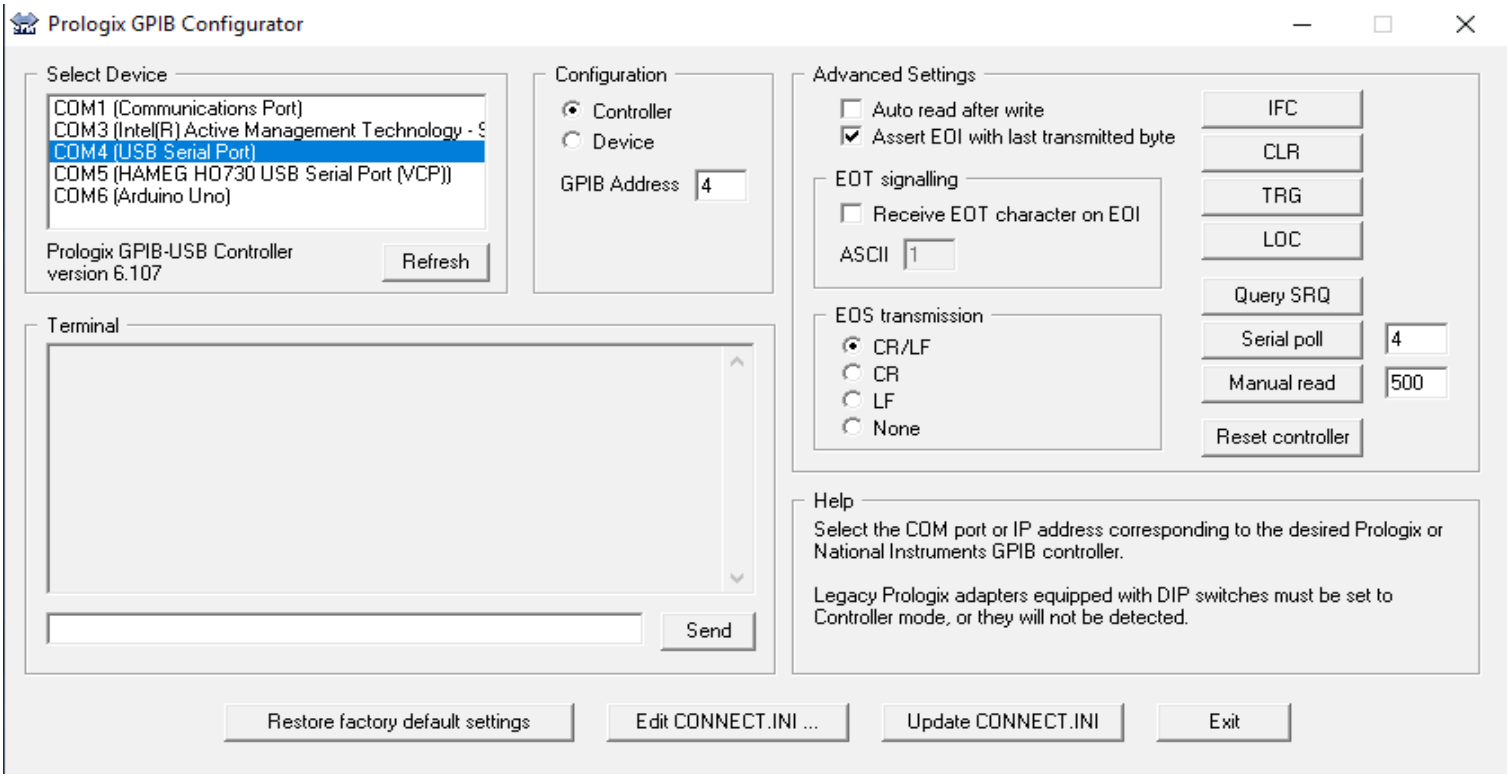
USAGE OF INSTRUMENT'S CLASS CODES

1. Keithley:

```
class Keithley(object):
    """
        This class serves to commincate with Keithley.
        I recommend to use at least 0.6 sec sleep function, after use write and read function!
    """

    def __init__(self, resource_manager, address):
        """
        resource_manager represents a function of visa module.
        Without resource manager, program can not communicate with instrument.
        An example for address of instrument : 'ASRL4::INSTR'
        """

        self.keith = resource_manager.open_resource(address)


    def keithley_specifications(self, function = 'VOLTage:DC', auto_range = 'ON', lower_range = -10, upper_range = 10):
        """
            Function variable stands for function of Keithley.
            Functions should be written inside funcion when function called.
            Some of the functions listed below:
            1. VOLTage:DC
            2. VOLTage:AC
            3. CURRent:DC
            4. CURRent:AC
            5. RESistance
            6. TEMPerature
            More function can be found in the Keithley 2000 manual.
            auto_range variable represents autoranging of the instrument.
            It can take 2 different string commands which are ON and OFF.
            If you set auto range OFF, then you can specify upper and lower range of Keithley.
        """
        self.keith.write(":SENSe:FUNCtion '{}'".format(function))
        self.keith.write("SENse:{}:RANGe:AUTO {}".format(function, auto_range))

        if auto_range == 'OFF':
            self.keith.write("SENSe:{}:RANGe:UPPer {}".format(function, upper_range))
            self.keith.write("SENSe:{}:RANGe:LOWer {}".format(function, lower_range))

    def IDN(self):
        """
        Ask instrument its name.
        """

        self.keith.write("*IDN?")
        self.keith.write('++read eoi')
        reading = self.keith.read()
        return reading

    def read_data(self):
        """
        When this function called, it is going to return the last instrument reading.
        """
        self.keith.write("SENse:DATA?")
        self.keith.write('++read eoi')
        output = self.keith.read()
        return output

    def close_channel(self, channel_no):
        """
        This function can be used if there is a multiplexer which connected to Keithley.
        When function called with channel_no variable, it closes choosen channel.
        """
        self.keith.write(":rout:close (@{})".format(channel_no))
```

```
def send_command(self, command):
    """
    Command variable can take any command, but it should be written in true format as in the manual.
    """
    self.keith.write("{}".format(command))

def read_command_output(self):
    """
    After command, it reads output of the instrument and return what it read.
    """
    self.keith.write('++read eoi')
    reading = self.keith.read()
    return reading
```

Prologix's configuration:



At Advanced Settings Section, if you check 'Auto read after write', and if you try to send just a command which has no read attribute, Keithley will rise an error. To avoid this error, I am using '++read eoi' command which basically tells prologix read instrument after send a question. Communication between Prologix and Keithley depends on the CR/LF end termination and EOI. Do not forget the check these boxes, otherwise code will always give error.

Keithley usage example:

```
from keithley_class import Keithley # From python file import Keithley Class
import pyvisa # Import Pyvisa
from time import sleep

rm = pyvisa.ResourceManager() # Open Resource Manager

keithley_address = 'ASRL4::INSTR' # Address of instrument

keith = Keithley(rm, keithley_address) # Connect Keithley via pyvisa and prologix

print(keith.IDN()) # ask ID of instrument

keith.keithley_specifications() # Specify measurement style
#keith.keithley_specifications(function = 'CURRent:DC') # Specify measurement style

keith.send_command(":FETCh?")
print(keith.read_command_output())

keith.send_command(":STATus:CLEar") # Clears all messages from Error Queue

i = 1

while True:

    keith.close_channel(i)
    sleep(1)
    print("channel {} = ".format(i))
    print(keith.read_data())
    i += 1
    if  i == 5:
        i = 1
```

2. ROHDE&SCHWARZ HMP4040(POWER SUPPLY):

Communication with power supply has made via usb connection. There is no any configuration necessary to do before usage, if you installed its driver.

```
from power_supply import HMP4040
import pyvisa

rm = pyvisa.ResourceManager()

power_sup_address = 'ASRL5::INSTR' # Address of instrument

power_sup = HMP4040(rm, power_sup_address) # Connect Keithley via pyvisa and prologix

print(power_sup.IDN()) # ask ID of instrument

power_sup.select_channel(1) # select channel

power_sup.set_voltage_value(20) # set voltage level

power_sup.turn_on_off_output('ON') # Output ON/OFF
```

```
class HMP4040(object):

    def __init__(self, resource_manager, address):
        """
        resource_manager represents a function of visa module.
        Without resource manager, program can not communicate with instrument.
        An example for address of instrument : 'ASRL5::INSTR'
        """
        self.power_sup = resource_manager.open_resource(address, send_end = True)
        self.power_sup.write("*CLS")
        self.power_sup.write("*RST")

    def select_channel(self, channel_number):
        self.power_sup.write("INSTrument OUT{}".format(channel_number))

    def set_voltage_value(self, volt_value):
        """
        This function sets voltage level of selected channel.
        """
        self.power_sup.write("SOURce:VOLTage:LEVel:IMMediate:AMPLitude {}".format(volt_value))

    def set_current_value(self, curr_value):
        """
        This function sets current level of selected channel.
        """
        self.power_sup.write("SOURce:CURRent:LEVel:IMMediate:AMPLitude {}".format(curr_value))

    def set_voltage_current_value(self, volt, curr):
        """
        Sets both voltage and current
        """
        self.power_sup.write("APPLy {},{}".format(volt, curr))

    def turn_on_off_output(self, out):
        """
        This function sets selected output as on or off
        """
        self.power_sup.write("OUTPut:STATe {}".format(out))

    def IDN(self):
        """
        Ask instrument its name.
        """
        idn = self.power_sup.query("*IDN?")
        return idn

    def send_command(self, command):
        """
        Command variable can take any command, but it should be written in true format as in the manual.
        """
        self.power_sup.write("{}".format(command))

    def read_command_output(self):
        """
        After command, it reads output of the instrument and return what it read.
        """
        reading = self.power_sup.read()
        return reading
```

3. Aim TTI TG5012A(PULSE GENERATOR):

Communication with TG5012 has made via internet. There is no any configuration necessary to do before usage, if you installed its driver.

```
class TG5012A(object):

    def __init__(self, resource_manager, address):
        """
        resource_manager represents a function of visa module.
        Without resource manager, program can not communicate with instrument.
        An example for address of instrument : 'TCPIP::128.141.154.197::9221::SOCKET
        """
        self.pg = resource_manager.open_resource(address)
        self.pg.write_termination = '\n'
        self.pg.read_termination = '\r\n'

    def choose_channel(self, channel_number):
        self.pg.write("CHN {}".format(channel_number))

    def choose_wave_type(self, wave_type):
        """
            wave_type variable stands for function of Pulse Generator.
            Functions should be written inside funcion when function called.
            Some of the functions listed below:
            1. TRIANG
            2. PULSE
            3. NOISE
            4. SQUARE
            5. SINE
            6. RAMP
        """
        self.pg.write("WAVE {}".format(wave_type))

    def frequency(self, freq):
        self.pg.write("FREQ {}".format(freq))

    def period(self, per):
        self.pg.write("PER {}".format(per))

    def amplitude_range(self, amp_range):
        """
        Set amplitude range ; auto
        """
        self.pg.write("AMPLRNG {}".format(amp_range))

    def amplitude_unit(self, amp_unit):
        """
        Amplitude units are VPP, VRMS and DBM
        """
        self.pg.write("AMPUNIT {}".format(amp_unit))

    def amplitude(self, amplitude):
        self.pg.write("AMPL {}".format(amplitude))

    def dc_offset(self, dc_off):
        self.pg.write("DCOFFS {}".format(dc_off))

    def output_state(self, out):
        self.pg.write("OUTPUT {}".format(out))

    def output_load(self, zload):
        self.pg.write("ZLOAD {}".format(zload))

    def duty_cycle_for_square_wave(self, width):
        self.pg.write("SQRSYMM {}".format(width))
```

```
rm = pyvisa.ResourceManager()
pulse_gen_address = 'TCPIP::dcct-pulse-generator.cern.ch::9221::SOCKET' # Address of instrument
pulse_gen = TG5012A(rm, pulse_gen_address) # Connect Keithley via pyvisa and prologix
print(pulse_gen.IDN()) # ask ID of instrument
pulse_gen.choose_channel(2)
pulse_gen.choose_wave_type('SQUARE')
pulse_gen.period(0.00009)
pulse_gen.amplitude(5)
pulse_gen.dc_offset(2.5)
pulse_gen.choose_channel(2)
pulse_gen.output_load(50)
pulse_gen.output_state('OFF')
#pulse_gen.amplitude_unit
```

If you do not know the address of instrument, you could find it with using Qt Console in Anaconda. For example: