elab.main

elab.main()

Defines the superclass instrument under which all instruments are initialized. Also defines the class, bundle, which enables cross talk between separate instruments while providing high level commands.

elab.main.instrument(com_port,**kwargs)

Parameters:

com_port : integer

The serial port for communication.

verbose : bool, default=False

if True, will print the commands and responses sent.

**kwargs : ‘baud_rate’, ‘verbose’, ‘timeout’

‘baud_rate’ : int, default=9600

Set a new baud rate.

‘verbose’ : bool, default=False

If True, will print out all commands and responses sent.

‘timeout’ : int, default=1

Set instrument timeout time in seconds.

Methods:

close()

Close serial communication with the instrument

elab.main.bundle(inst_list,**kwargs)

Parameters:

inst-list : list

The list of instrument objects to bundle together.

**kwargs : ‘verbose’

‘verbose’ : bool, default=False

If True, will print out all commands and responses sent.

Methods:

change_default_ports(cell_name, waste_name, air_name, flush_name)

Change the default ports for the switching valve.

'cell_name' : str, default='cell'

'waste_name' : str, default='waste'

'air_name' : str, default='air'

'flush-name' : str, default='flush'

check_types(inst_list)

Check the given instrument types to see if they are initialized in the bundle class.

Returns true if all instrument objects in the list are initialized correctly.

'inst_list' : list

A list of instrument objects.

load_ports(filepath)

Load port definitions for the switching valve. Ports can be loaded in through a .csv file or a python dictionary.

'filepath' : dict, .csv

If type is dict, load the ports as defined by the dictionary. If type is str with a .csv filepath, load the ports as defined by the csv file.

change_cell(cell_name)

Change cell to the new cell_name.

'cell_name' : str

The new cell name to name the cell.

conc(solution)

Return the concentration of the solution queried.

'solution' : str

The solution to query the concentration of.

from_to(line_from, line_to, vol)

Move the specified volume from one line to the other.

‘line_from’ : str

A string denoting the line to move the solution from, as specified by the loaded ports.

‘line_to’ : str

A string denoting the line to move the solution to, as specified by the loaded ports.

‘vol’ : int

The volume to transport between the lines.

from_to_all(line_from, line_to)

Move everything in the syringe pump from one line to the other.

‘line_from’ : str

A string denoting the line to move the solution from, as specified by the loaded ports.

‘line_to’ : str

A string denoting the line to move the solution to, as specified by the loaded ports.

init_line(solution)

Initialize a line for a solution by filling an intake line with its respective solution.

‘solution’ : str

A string denoting the solution and the intake line to initialize.

reset_to_waste()

Reset the syringe pump, emptying it and sending it to the waste line.

light_dispense(solution, volume, **kwargs)

Dispense the input volume of solution to the cell. Then send 1 mL of air into the cell, stirring it slightly.

‘solution’ : str

A string denoting the solution to dispense into the cell.

‘volume’ : int

An integer denoting the volume to dispense.

‘**kwargs’ : ‘air_volume’

‘air_volume’ : int, default = 1

The mL of air to dispense into the cell after dispensing the solution.

dispense(solution, volume, **kwargs)

Dispense the volume of solution into the cell. Prime the cell with the 1 mL of solution first. Remove the primer. Dispense the volume. Send 1 mL of air into the cell to stir the solution.

‘solution’ : str

A string denoting the solution to dispense into the cell.

‘volume’ : int

An integer denoting the volume to dispense.

‘**kwargs’ : ‘prime_volume’, ‘aspirate_volume’

‘prime_volume’ : int, default = 0.1

The mL of solution to prime the cell with.

‘aspirate_volume’ : int, default = 5

The mL of solution to withdraw from the cell.

remove_cell_contents(volume)

Remove the specified volume from the cell and dispense it into waste.

‘volume’ : int

The volume to remove from the cell.

clean_cell(volume,**kwargs)

Clean the cell. Remove the current cell contents. Rinse the cell with the solution hooked up to the ‘flush’ line. Then remove the flush solution from the cell.

‘volume’ : int

The volume to remove from the cell.

‘**kwargs’ : ‘extra_volume’

‘extra_volume’ : int, default=5

The extra volume in mL to remove when removing volume from the cell. Ensures that all solution is removed from the cell.

clear_line(solution,**kwargs)

This clears the line of all of its solution with flush, then sends 1 mL of air into the line, clearing the line of flush.

‘solution’ : string

The solution line to be cleared.

‘**kwargs’ : ‘air_volume’

‘air_volume’ : int, default = 1

The mL of air to pass into the line, clearing it of the flush used to clear the line of the solution.

bubble(**kwargs)

This dispenses air into the cell, bubbling the solution and resetting established diffusion layers and boundary conditions in the electrochemical cell.

‘**kwargs’ : ‘air_volume’

‘air_volume’ : int, default = 1

The mL of air to pass into the line, clearing it of the flush used to clear the line of the solution.

calibrate_pH(pH_list,**kwargs)

This ultimately calibrates the pH meter. This method dispenses several calibration standards into the cell and measures the voltage response of the meter for each standard. A linear regression is performed on the stored voltage responses, generating a calibration curve for the pH meter. Future pH measurements are then conducted using this calibration curve.

‘pH_list’ : list

The list of pH standard solutions to generate the calibration curve from.

‘**kwargs’ : ‘buff_volume’, ‘extra_volume’

‘buff_volume’ : int, default=5

The mL of the standard solution volumes to dispense into the cell for calibration.

‘extra_volume’ : int, default=2

Extra volume is added to a called clean_cell() method in calibrate_pH(), ensuring all solution from the cell is aspirated.

Experimental methods in elab.main.bundle()

mix_component(solution,volume,**kwargs)

Define a solution mixture that we wish to use later. Example usage:

mix = [lab.mix_component('tempo',1),lab.mix_component('buffer',4)]

‘solution’ : str

A string denoting the solution to be included in the mixture

‘volume’ : int

An integer denoting the volume of the solution to be included in the mixture. Defined in mL.

mix_dispense(components)

Dispense a mixture into the cell.

‘components’ : list

A list of mix_components as returned by the mix_component() method.

Example usage:

mix = [lab.mix_component('tempo',1),lab.mix_component('buffer',4)]

lab.mix_dispense(mix)

mix_prime(components,**kwargs)

Prime a cell with a mixture, then remove all solution from the cell.

‘components’ : list

A list of mix_components as returned by the mix_component() method.

‘**kwargs’ : ‘extra_volume’

‘extra_volume’ : int, default=5

Extra volume is added to a remove_cell_contents() ensuring all primer is aspirated from the cell.

prime(solution,volume,**kwargs)

Prime a cell with a solution, then remove all of it from the cell.

'solution' : str

A string denoting the solution to be dispensed.

'volume' : int

An integer denoting the volume of the solution to be used as primer. Defined in mL.

'**kwargs' : 'extra_volume'

'extra_volume' : int, default=5

Extra volume is added to a remove_cell_contents() ensuring all primer is aspirated from the cell.

elab.AlicatMFC

elab.AlicatMFC(com_port, address = 'A', verbose=False, **kwargs)

AlicatMFC enables python control of an Alicat mass flow controller for gas flow automation.

Parameters:

com_port : integer

The serial port to communicate with the mass flow controller.

address : str, default='A'

The string address of the mass flow controller to enable serial communication.

verbose : bool, default=False

If True, will print the commands and responses sent to the mass flow controller.

**kwargs :

If 'address' in kwargs : set new instrument address

If 'baud_rate' in kwargs : set new baud rate, default = 19200

Methods:

compile_cmd(command,**kwargs)

Send an ASCII command to the mass flow controller and read out its subsequent response.

compile_cmd has a number of predefined useful commands for interfacing with the mass flow controller. These are used in the methods described below, but compile_cmd also enables the sending of additional and custom commands to the controller. See the Alicat serial communication manual for more information.

'command' : str, default=''

The command to send to the mass flow controller.

'**kwargs' : 'parameter1', 'parameter2', 'parameter3', 'show_cmd'

'parameter1' : str, default=''

The first command parameter.

'parameter2' : str, default=''

The second command parameter.

'parameter3' : str, default=''

The third command parameter.

'show_cmd' : bool, default=False

When set to True, the encoded command sent to the controller will be printed.

query_dataframe()

Returns the live dataframe info of the mass flow controller in the following format: ID, Absolute Pressure, Temperature, Vol. Flow, Mass Flow, Setpoint, Totalizer, Gas
Example response: 'A +15.542 +24.57 +16.667 +15.444 +15.444 22741.4 N2'

query_data()

Returns the data at the time of polling.

query_avg_data(time, statistic, **kwargs)

Returns the average value of the specified statistic(s) over a given period of time.

Parameters:

time : int, default=None

The number of milliseconds to average the statistic's values. The device polls every millisecond.

statistic : int, default=None

A value from 0-703 defining which statistic should be averaged. See the Alicat manuals for more information.

**kwargs : 'statistic2'

'statistic2' : int, default=None

Return an additional statistic averaged over the specified time.

query_gas()

Returns the active gas.

query_setpoint()

Returns the current setpoint.

query_setpoint_modes()

Returns a dictionary of all of the setpoint modes and their subsequent values.

dict = {'Statistic' : 'Value', 'Absolute pressure' : 34, 'Volumetric flow' : 36, 'Mass flow' : 37, 'Gauge pressure' : 38, 'Pressure differential' : 39}

query_setpoint_range()

Returns the safe working range of the active setpoint mode.

query_max_ramp_rate()

Returns the device's set maximum ramp rate for setpoint adjustment.

list_gases()

Returns the list of gases installed on the mass flow device.

start_streaming()

Change the device to stream data.

stop_streaming()

Set the streaming device to stop streaming.

set_gas(gas_num)

Set the active gas for the mass flow controller.

‘gas_num’ : int, default={startup_gas_num}

A value from 0-210, with each value corresponding to an individual gas or gas mixture. See Alicat manual or call list_gases() for more information.

set_startup_gas(gas_num)

Set the startup active gas for the mass flow controller.

‘gas_num’ : int, default={startup_gas_num}

A value from 0-210, with each value corresponding to an individual gas or gas mixture. See Alicat manual or call list_gases() for more information.

change_setpoint(value)

Set a setpoint for the controller to adjust flow or pressure to.

‘value’ : float, default=None

Any value within the working range of the current setpoint type. Call query_setpoint_range for more information.

set_unit(statistic,unit_value)

Set the engineering units for the desired group of statistics such as pressure or flow.

‘statistic’ : int, default=None

A value from 0-703 defining which statistic's unit will be changed. See the Alicat manuals for more information on the statistics' values.

'unit_value' : int, default=0

A value from 0-63 defining what the new set unit will be. See the Alicat manuals for more information on the values.

set_setpoint(setpoint_val,**kwargs)

Set the setpoint for the mass flow controller with the optional ability to change the engineering units.

'setpoint_val' : int, default=None

A value within the safe working range of the active setpoint mode. Call query_setpoint_range for more information.

**kwargs : 'unit_val'

'unit_val' : int, default=None

A value from 0-63 defining what the new set unit will be. See the Alicat manuals for more information on the values.

set_setpoint_mode(setpoint_mode)

Set the setpoint mode for the mass flow controller.

'setpoint_mode' : int

A value from 34 - 39 denoting the mode of the mass flow controller to be used. See attached dictionary here:

dict = {'Statistic' : 'Value', 'Absolute pressure' : 34, 'Volumetric flow' : 36, 'Mass flow' : 37, 'Gauge pressure' : 38, 'Pressure differential' : 39}

tare_flow()

Tare the flow.

set_pressure_limit(pressure_limit)

Defines the maximum pressure allowed by the controller.

'pressure_limit' : int

A value >=1. Pressure limits will be dependent on the mass flow controller and setup used.

elab.E0RR80

elab.E0RR80(com_port, verbose=False)

E0RR80 enables serial communication with a {insert here} mass balance.

Parameters:

com_port : integer

The serial port to communicate with the mass balance.

verbose : bool, default=False

If True, will print the commands and responses sent to the mass balance.

Methods:

query_mass()

Returns the mass measured on the scale.

tare()

Tares the mass balance.

on()

Turn on the mass balance.

elab.gen_serial

elab.gen_serial(com_port, verbose=False)

A general serial class acting as a dummy serial instrument. Meant for sending and returning code without actually running the instrument.

Parameters:

com_port : integer

The serial port for communication.

verbose : bool, default=False

If True, will print the commands and responses sent.

Methods:

send(comm)

Sends the command to the instrument

‘comm’ : str

A string command encoded with utf-8 and sent to the instrument

readline()

Returns the decoded instrument response

elab.HS7

elab.HS7(com_port, verbose=False,**kwargs)

HS7 enables serial communication with a stirring hotplate.

Parameters:

com_port : integer

The serial port for communication.

verbose : bool, default=False

If True, will print the commands and responses sent.

**kwargs : ‘max_temp’

‘max_temp’ : int, default=50

Set the maximum temperature of the hotplate in Celsius

Methods:

query_temp()

Returns the temperature of the hotplate

set_temp(temp)

Set the desired temperature of the hotplate

‘temp’ : int, float

A value within the range of 0 and the set maximum temperature of the hotplate.

start_temp()

Start heating up the hotplate to the set temperature.

stop_temp()

Turn off the heating element in the hotplate.

set_spin(spin)

Set the spin RPM of the stirrer.

'spin' : int

A value within the range of 0 and 1500.

start_spin()

Turn on the stirring function and begin stirring at the set spin rate.

stop_spin()

Turn off the stirring function.

elab.Legato100

elab.Legato100(com_port, verbose=False,**kwargs)

Legato100 enables serial communication with a Legato100 KD Scientific syringe pump.

Parameters:

com_port : integer

The serial port for communication.

verbose : bool, default=False

If True, will print the commands and responses sent.

**kwargs : 'address', 'baud_rate'

If 'address' in kwargs : set new instrument address

If 'baud_rate' in kwargs : set new baud rate, default = 19200

Methods:

compile_cmd(command,**kwargs)

Send an ASCII command to the syringe pump and read out its subsequent response.

compile_cmd has a number of predefined useful commands for interfacing with the syringe pump. These are used in the methods described below, but compile_cmd also enables the sending of additional and custom commands to the pump. See the Legato100 serial communication manual for more information.

‘command’ : str, default=’’

The command to send to the syringe pump.

‘**kwargs’ : ‘parameter1’, ‘parameter2’, ’show_cmd’

‘parameter1’ : str, default=’’

The first command parameter.

‘parameter2’ : str, default=’’

The second command parameter.

‘show_cmd’ : bool, default=False

When set to True, the encoded command sent to the controller will be printed.

query_address()

Return the syringe pump communication address. Default is 0.

set_address(address)

Set a new address for the syringe pump.

‘address’ : int, default=0

An integer value between 0 and 99.

set_brightness(brightness)

Set the screen brightness on the syringe pump.

‘brightness’ : int, default=100

An integer value between 1 and 100.

set_force(force)

Set the force of the pump. Smaller syringes (<1mL) and glass syringes require smaller forces (30-50). See the Legato100 manual for more information

‘force’ : int

A value in the range of 1 to 100.

set_run_mode()

Set the syringe pump to begin running.

stop()

Stop the syringe pump’s movement.

display_config()

Return the current configuration of the syringe pump.

calibrate_tilt()

Recalibrate the tilt of the syringe pump.

display_syringe()

Return the current syringe parameters.

set_syringe(**kwargs)

Set the volume and/or diameter of the syringe.

‘volume’ : int

Set the volume of the new syringe in mL.

‘diameter’ : int

Set the diameter of the new syringe in mm.

set_target_volume(target,**kwargs)

Set the target volume to be dispensed.

‘target’ : int

The volume to be dispensed.

‘**kwargs’ : ‘unit’

‘unit’: str, default=’mL’

The unit of the target volume to be dispensed.

set_rate(rate,**kwargs)

Set the target dispense rate.

‘rate’ : str

The rate to dispense at. Examples: ‘mL/min, uL/sec, etc’

‘**kwargs’ : ‘unit’

‘unit’: str, default=’uL/min’

The unit of the dispense rate.

set_target_time(time)

The target time to run the syringe pump for.

‘time’ : int

dispense(volume,**kwargs)

Run the syringe pump and dispense the specified volume.

‘volume’ : int

The volume to be dispensed

‘**kwargs’ : ‘unit’, ‘rate’, ‘target_time’

‘unit’ : str, default = ‘mL’

The unit of the target volume to be dispensed.

‘rate’ : str, default = ‘uL/min’

The rate at which to dispense.

‘target_time’ : int

The time to dispense for in seconds.

elab.MUX8

elab.MUX8(com_port, verbose=False)

MUX8 enables serial communication with an arduino multiplexer enabling controller of an interdigitated electrode array.

Parameters:

com_port : integer

The serial port for communication.

verbose : bool, default=False

If True, will print the commands and responses sent.

Methods:

electrode(n)

Select which electrode to enable.

‘n’ : int

The value n must be between 1 and 8.

ida(n)

Select which IDA to communicate with.

‘n’ : int

The value n must be between 1 and 4.

gen(n)

Select which generator electrode to communicate with.

‘n’ : int

The value n must be between 1 and 4.

coll(n)

Select which collector electrode to communicate with.

‘n’ : int

The value n must be between 1 and 4.

send_comm(n)

Send and receive messages with electrode n. A submethod called by the four electrode selection methods above.

‘n’ : int

The value of n is defined by the higher level selection methods called above.

elab.pH_arduino

elab.pH_arduino(com_port, verbose=False)

pH_arduino enables serial communication with an arduino board interfaced with a pH meter.

Parameters:

com_port : integer

The serial port for communication.

verbose : bool, default=False

If True, will print the commands and responses sent.

Methods:

send_comm()

Query the voltage of the pH meter.

voltage(**kwargs)

Return the voltage output from the pH meter.

‘**kwargs’ : ‘delay’, ‘average’

‘delay’ : int, default=30

Waits the specified amount of time before returning the voltage.

‘average’ : int, default=10

Averages the voltage response over the specified number of measurements.

measure(**kwargs)

Measure the pH from the voltage output using a loaded calibration curve. Calls voltage(**kwargs) to obtain the voltage output.

‘**kwargs’ : ‘delay’, ‘average’

‘delay’ : int, default=30

Waits the specified amount of time before returning the voltage.

‘average’ : int, default=10

Averages the voltage response over the specified number of measurements.

elab.SV07

elab.SV07(com_port, **kwargs)

SV07 enables hexadecimal based serial communication with a SV07 Runze switching valve model.

Parameters:

com_port : integer

The serial port for communication.

**kwargs : ‘verbose’, ‘address’

‘verbose’ : bool, default=False

If True, will print out all commands and responses sent.

‘address’ : hex, default=0x00

Must be a hex address, set address to whatever address is specified.

Methods:

compile_cmd(command,**kwargs)

Send a hex command to the switching valve and read out its subsequent response.

compile_cmd has a library of predefined useful commands for interfacing with the valve. These are used in some of the methods described below, but compile_cmd also enables the sending of additional and custom commands to the controller, properly formatting them for hexadecimal based serial communication. See the Runze SV07 communication manual for more information.

‘command’ : str

A string denoting a predefined command to call within compile_cmd

‘**kwargs’ : ‘parameter1’, ‘parameter2’,’verbose’

‘parameter1’ : hex, default=0x00

The first hex parameter for a command.

‘parameter2’ : hex, default=0x00

The second hex parameter for a command.

‘verbose’ : bool, default=False

If True, print sent commands and received responses.

Additional library commands in compile_cmd({library_command_here}):

‘query_address’

Returns the hexadecimal address of the switching valve.

‘query_position’

Returns the current valve position of the instrument.

‘query_version’

Returns the current software version installed on the switching valve.

‘strong_stop’

Shuts down all movement of the switching valve. Will require a restart of the switching valve.

To call these library commands, call compile_cmd({additional_command_here}). Example: Calling “compile_cmd(‘query_address’)” returns the hexadecimal address of the switching valve.

build_packet(command_hex, parameter1, parameter2)

Builds the 8 byte hexadecimal command necessary to send to and receive from the switching valve.

‘command_hex’ : hex

The hex command detailing what the switching valve should do.

‘parameter1’ : hex

The hex value detailing the first parameter of the hex command.

‘parameter2’ : hex

The hex value detailing the second parameter of the hex command.

write_read(packet)

Sends the 8 byte packet to the valve. Returns the 8 byte response.

‘packet’ : bytearray

The 8 byte packet denoting a correctly formatted hexadecimal command.

check_movement()

Constantly probe the motor status of the switching valve. While the valve is executing code and actively moving, check_movement() will continue to probe the motor status.

As soon as the motor finishes moving, check_movement() will clear and more code can be sent for execution to the switching valve.

reset()

The valve resets to its default position and then stops.

origin_reset()

The valve resets to its encoder origin position, which overlaps with the default reset position.

port(port)

Set the active valve position to the port defined.

‘port’ : int

An integer value of 1 through 16.

elab.SY08

elab.SY08(com_port, **kwargs)

SY08 enables hexadecimal based serial communication with a SY08 Runze syringe pump model.

Parameters:

com_port : integer

The serial port for communication.

**kwargs : ‘verbose’, ‘address’

‘verbose’ : bool, default=False

If True, will print out all commands and responses sent.

‘address’ : hex, default=0x00

Must be a hex address, set address to whatever address is specified.

Methods:

compile_cmd(command,**kwargs)

Send a hex command to the syringe pump and read out its subsequent response.

compile_cmd has a library of predefined useful commands for interfacing with the pump. These are used in some of the methods described below, but compile_cmd also enables the sending of additional and custom commands to the controller, properly formatting them for hexadecimal based serial communication. See the Runze SY08 communication manual for more information.

‘command’ : str

A string denoting a predefined command to call within compile_cmd

‘**kwargs’ : ‘parameter1’, ‘parameter2’,’verbose’

‘parameter1’ : hex, default=0x00

The first hex parameter for a command.

‘parameter2’ : hex, default=0x00

The second hex parameter for a command.

‘verbose’ : bool, default=False

If True, print sent commands and received responses.

Additional library commands in compile_cmd({library_command_here}):

‘query_address’

Returns the hexadecimal address of the switching valve.

‘query_max_speed’

Returns the maximum operating speed of the syringe pump.

‘query_version’

Returns the current software version installed on the syringe pump.

‘query_motor_status’

Returns the motor status.

‘strong_stop’

Shuts down all movement of the syringe pump. Will require a restart of the syringe pump.

To call these library commands, call compile_cmd({additional_command_here}). Example: Calling “compile_cmd(‘query_max_speed’)” returns the hexadecimal max RPM of the syringe pump.

build_packet(command_hex, parameter1, parameter2)

Builds the 8 byte hexadecimal command necessary to send to and receive from the syringe pump.

‘command_hex’ : hex

The hex command for the syringe pump.

‘parameter1’ : hex

The hex value detailing the first parameter of the hex command.

‘parameter2’ : hex

The hex value detailing the second parameter of the hex command.

write_read(packet)

Sends the 8 byte packet to the valve. Returns the 8 byte response.

‘packet’ : bytearray

The 8 byte packet denoting a correctly formatted hexadecimal command.

check_movement()

Constantly probe the motor status of the syringe pump. While the pump is executing code and actively moving, check_movement() will continue to probe the motor status. As soon as the motor finishes moving, check_movement() will clear and more code can be sent for execution to the syringe pump.

query_position()

Return the position of the piston. 0 denotes the home position, where the syringe will have a volume of zero. 12000 denotes the max position, where the syringe will be fully withdrawn and have a full volume.

set_speed(speed)

Set the speed of the motor moving the syringe piston in RPM.

‘speed’ : int

An integer in the range of 1 to 600.

move_to_position(position,**kwargs)

Move the piston to the specified position.

‘position’ : int

An integer in the range of 0 to 12000.

‘**kwargs’ : ‘speed’

‘speed’ : int

An integer in the range of 1 to 600. Sets motor speed in RPM.

full_reset()

The syringe will run to the top and then go back a certain number of offset steps, leaving a small gap between the top of the piston and the syringe. This gap is left to improve the service life of the piston seal.

reset()

The syringe will go directly to the home position. It will then query the motor status until the system is complete.

reset_no_check()

The syringe will move directly to the home position.

aspirate(volume,**kwargs)

Pull the specified volume into the syringe pump

‘volume’ : int

An integer specifying the volume to pull into the syringe in mL.

**kwargs’ : ‘speed’

‘speed’ : int

An integer in the range of 1 to 600. Sets motor speed in RPM.

discharge(volume,**kwargs)

Push the specified volume into the syringe pump

‘volume’ : int or str

An integer specifying the volume to pull into the syringe in mL or a string equal to ‘all’, denoting that everything in the syringe should be dispensed.

**kwargs’ : ‘speed’

‘speed’ : int

An integer in the range of 1 to 600. Sets motor speed in RPM.