Pychron Documentation

Release 1.4

Jake Ross

April 16, 2012

CONTENTS

1 Scripting **3**
- 1.1 General . . . 3
- 1.2 Writing Bakeout Scripts . . . 4
- 1.3 Writing Extraction Line Scripts . . . 6

2 Procedures **7**
- 2.1 CO_2 Stage Calibration . . . 7
- 2.2 Loading CO_2 Samples . . . 7

3 Remote Hardware **9**
- 3.1 System Error Codes . . . 9
- 3.2 Laser Error Codes . . . 9
- 3.3 System Calls . . . 9
- 3.4 Laser Calls . . . 10

4 Indices and tables **13**

Index **15**

Contents:

CHAPTER

ONE

SCRIPTING

1.1 General

```
#============================================================
def main():
    info('this is an info message')
    acquire('pipette')
    info('aquired resourece - pipette')
    sleep(15)
    release('pipette')
    begin_interval(120)
    #do some stuff ...
    complete_interval() #wait until 120 seconds have passed
#============================================================
```

1.1.1 functions

info(*msg*)
add info `msg` to the log

acquire(*resource*)
reserve the `resource` for use. blocks other scripts from using `resource` until `release()` is called.

release(*resource*)
release `resource` so other scripts can use it.

sleep(*seconds*)
sleep for `seconds`. if `seconds>5` a timer will appear. decimal seconds are allowed e.g `sleep(0.5)`

gosub(*path_to_script*)
execute a pyscript located at `path_to_script`. `path_to_script` is relative to the current script. e.g `gosub(commonscripts/fuse.py)`. commonscripts must be a directory in the same directory this script is saved in.

begin_interval(*timeout*)
start an interval. if *timeout>5* a timer will appear.

complete_interval()
wait unit `timeout` has elapsed

1.2 Writing Bakeout Scripts

Here's how to write a bakeoutscript

1. launch BakeoutManager (Bo on the dock)
2. open the script editor (`Edit scripts` button). An empty script is opened as a default
3. use `Open` to open an existing script or just start writing one from scratch
4. write your script. see below
5. you may check the syntax of your script at any point by hitting `Test`. A dialog will pop up saying if there was a problem and what it was or if the script passed with no errors. The scripts syntax is also automatically checked before saving.
6. Use `Save` and `Save As` in the normal manner. script files should end with **.py** . if the file ending is omitted a **.py** is appended automatically
7. close the script editor

here is an example script

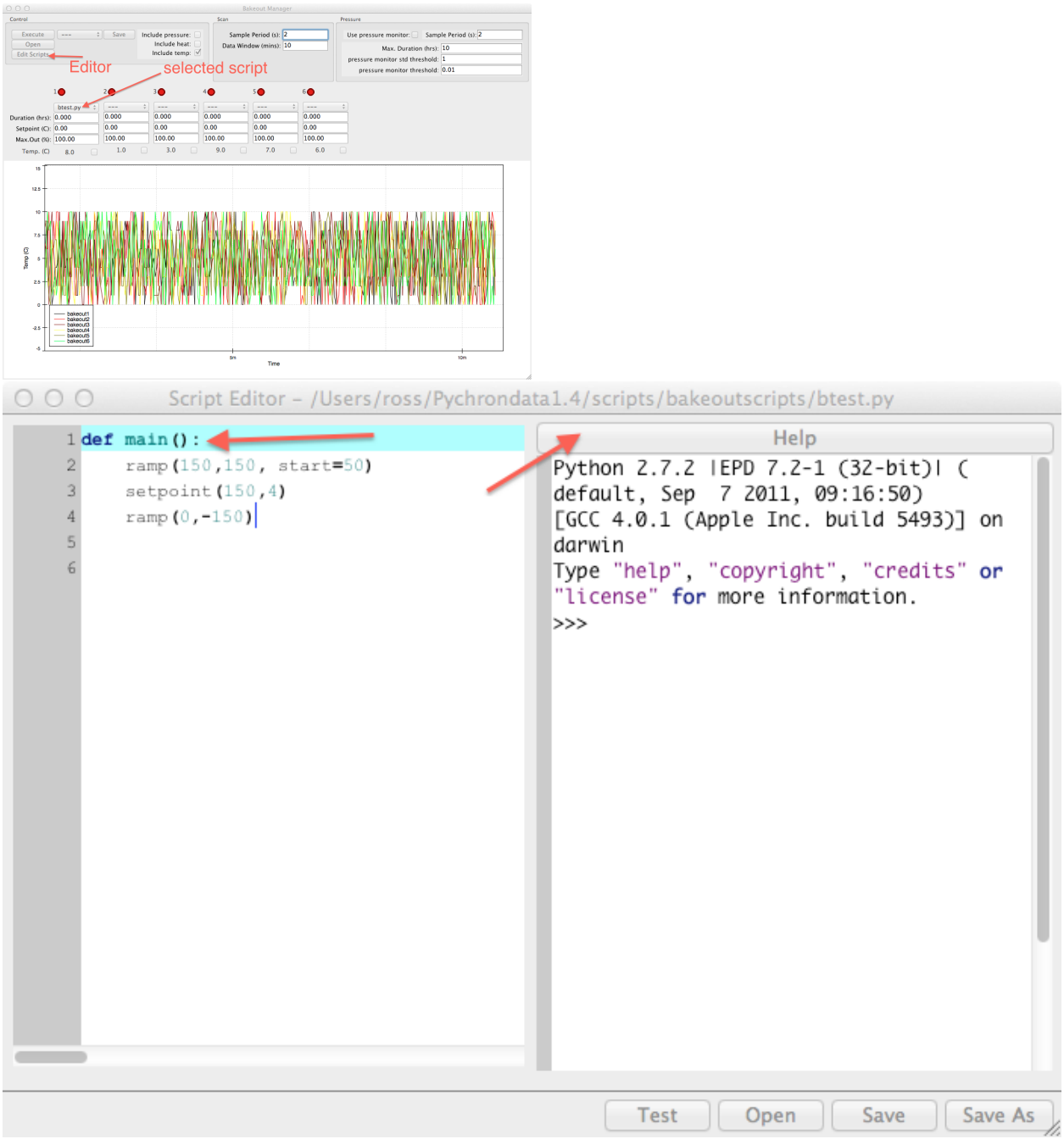
```
#================================================================================
#this is a comment in the script. use it at your leisure
#the following line def main(): is required and the entry point for the script
def main():
    #this is the body of the main function
    #it should be one tab in
    ramp(150,100,start=45)  # comments can go on the same line as functions
    setpoint(150,4)
    ramp(0,-150)
#end of script
#================================================================================
```

Lets break it down line by line.

- A comment can be added anywhere using the # character
- `def main():` defines a required function called "main" that will be executed when the script is run. **The main function is necessary** and is added to an empty script by default.
- the function `ramp(150,100,start=45)` sets the controller to 45 C, then increases the temperature setting to 150 C at a rate of 100 C/hr
- the function `setpoint(150,4)` sets the controller to a temperature of 150 C and holds it for 4 hours
- `ramp(0,-100)` decreases the temperature setting from the current temperature to 0 C at a rate of -100 C/hr.
- Notice that the `start` parameter of the `ramp()` function is optional. If omitted the controller's current temperature is used as the starting point

To activate a script for a given controller use the drop down menu located above the *Duration* box

To execute the bakeout hit `Execute`. `Stop` to Stop.



1.2.1 bakeout functions

ramp (*setpt*, *rate*[, *start=None*, *period=60*])
ramp controller's setpoint from `start` to `setpt` at a rate of `rate` C\hr. if `start=None` then the controller's current `temperature` is used. `period` defines seconds between setpoint updates. e.g. `period=30` sets the controllers setpoint every 30 seconds

setpoint (*temperature*, *duration*)
set controller's setpoint to `temperature` for `duration` hours

1.3 Writing Extraction Line Scripts

```
def main():
    open('A')
    sleep(1)
    close('A')

    info('this is an info message')

    acquire('pipette')
    info('loading air shot')
    open('X')
    sleep(15)
    close('X')
    release('pipette')
```

1.3.1 extraction line functions

open(*alias*)
: open the valve named `alias` e.g `open('A')`

close(*alias*)
: open the valve named `alias` e.g `close('A')`

CHAPTER

TWO

PROCEDURES

2.1 CO_2 Stage Calibration

1. Move the laser to the center hole.
2. Select the correct stage map e.g `221-hole`
3. Select `pychron-auto` as the calibration style
4. Hit `Calibrate`

Pychron will now automatically find up to five calibration holes. The calibration holes are specified on the third line of the stage map file e.g `221-hole.txt`. The calibration holes should be the N,E,S,W, and center holes.

Using the calibration holes Pychron calculates the center position and rotation of the tray. With an accurate calibration, Pychron will then move to each hole and determine a corrected position. This will take a few minutes.

2.1.1 Autofocus

Pychron has an auto focus feature that can produce a very sharp image. Configuration allows you to use various alogrithms to calculate the *focus measure* of an image. I find the Laplace filter with ~50% zoom produces a nice result. Autofocus is actual a misnomer in this case. What is really happenig is called passive focus. The *focus measure* is calculated by applying a mathematical filter the the image. These filters are used for example to calculate the gradient between adjacent pixels. Theoretically maximizing the gradient yields the most focused image. For more information see Autofocus

Hit `Autofocus` to perform an autofocus routine

2.2 Loading CO_2 Samples

!!Some please write me!!

CHAPTER

THREE

REMOTE HARDWARE

Remote hardware is used to allow other software clients access to pychron hardware, such as valves and laser systems. A simple messaging system is used to pass information between pychron and a client. Currently the most active client is Mass Spec which uses the remote hardware protocol to do all of its hardware tasks.

The protocol is broken in two sections *System Calls* and *Laser Calls*. Calls are simple ASCII text messages sent over the ethernet using either the UDP or TCP internet protocols

A response to a call is `OK`, a value, or an ErrorCode

3.1 System Error Codes

`InvalidValveErrorCode`
: error code 001

3.2 Laser Error Codes

InvalidValve

3.3 System Calls

`Open alias`
: Open the valve called `alias`. *InvalidValveErrorCode* return if `alias` not available

`Close alias`
: Close the valve called `alias`. *InvalidValveErrorCode* return if `alias` not available

`GetValveState alias`
: Get `alias` state. Returns `0` for closed `1` for open

`GetValveStates`
: Get all the valves states as a word. Returns a string <alias><state> e.g. `A1B0C1D1E0F0`

`GetValveLockStates`
: Get all the valves lock states as a word. Returns a string <alias><lock_state> e.g. `A1B0C1D1E0F0`

`StartMultRuns multruns_id`

`CompleteMultRuns`

`StartRun runid`

CompleteRun

PychronScript script

ScriptState

3.4 Laser Calls

Enable
Enable the laser. This is required before the laser's power can be set using *SetLaserPower*

Disable
Disable the laser

SetLaserPower power
Set the laser's power to `power`. `power` must be between 0-100.

ReadLaserPower
Read the lasers internal power meter. Returns an 8 bit value i.e 0-255

GetLaserStatus
Return OK if the laser can be enabled. If an interlock is enabled, such as insufficient coolant flow, an error will be returned

SetBeamDiameter
Set the beam diameter setting.

GetBeamDiameter
Get the beam diameter setting.

SetZoom zoom
Set zoom. `zoom` must be between 0-100.

GetZoom
Get zoom. returns value between 0-100.

GetPosition
Returns a comma separated list of positions X,Y,Z

GoToHole holenum
Go to hole `holename`. InvalidHoleErrorCode returned if hole is not in the current stage map.

GetJogProcedures
Return a list of available Jog procedures. Jog is a MassSpec term and a misnomer. Pychron internal refers to them as Patterns.

DoJog name
Launch the jog named `name`.

AbortJog
Abort the current jog

SetX xpos
Set the laser's stage controller X axis to `xpos`.

SetY ypos
Set the laser's stage controller X axis to `ypos`.

SetZ zpos
Set the laser's stage controller X axis to `zpos`.

SetXY xypos
Set the laser's stage controller X and Y axes to `xypos`. `xypos` should be a comma separated list of numbers. e.g `SetXY 10.1,-5.03`

GetXMoving

GetYMoving

GetDriveMoving

StopDrive

SetDriveHome

SetHomeX

SetHomeY

SetHomeZ

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

INDEX

A

acquire() (built-in function), 3

B

begin_interval() (built-in function), 3

C

close() (built-in function), 6
complete_interval() (built-in function), 3

G

gosub() (built-in function), 3

I

info() (built-in function), 3

O

open() (built-in function), 6

R

ramp() (built-in function), 5
release() (built-in function), 3

S

setpoint() (built-in function), 5
sleep() (built-in function), 3