# freshroastsr700 Documentation

***Release***

**Mark Spicer, Caleb Coffee**

**Oct 07, 2017**

# Contents

A Python module to control a FreshRoastSR700 coffee roaster.

CHAPTER 1

Contents

# FreshroastSR700 Package

## freshroastsr700 module

**class** freshroastsr700.**freshroastsr700**(*update_data_func=None*, *state_transition_func=None*, *thermostat=False*, *kp=0.06*, *ki=0.0075*, *kd=0.01*, *heater_segments=8*, *ext_sw_heater_drive=False*)

Bases: object

A class to interface with a freshroastsr700 coffee roaster.

**Args:** update_data_func (func): A function to call when this object receives new data from the hardware. Defaults to None.

state_transition_func (func): A function to call when time_remaining counts down to 0 and the device is either in roasting or cooling state. Defaults to None.

thermostat (bool): thermostat mode. if set to True, turns on thermostat mode. In thermostat mode, freshroastsr700 takes control of heat_setting and does software PID control to hit the demanded target_temp. Defaults to False.

ext_sw_heater_drive (bool): enable direct control over the internal heat_controller object. Defaults to False. When set to True, the thermostat field is IGNORED, and assumed to be False. Direct control over the software heater_level means that the freshroastsr700's PID controller cannot control the heater. Since thermostat and ext_sw_heater_drive cannot be allowed to both be True, this arg is given precedence over the thermostat arg. Note that

(thermostat=False, ext_sw_heater_drive=False), (thermostat=True, ext_sw_heater_drive=False), (thermostat=False, ext_sw_heater_drive=True),

**are all acceptable arg combinations. Only the** (thermostat=True, ext_sw_heater_drive=True),

cominbation is not allowed, and this software will set thermostat=False in that case.

kp (float): Kp value to use for PID control. Defaults to 0.06.

ki (float): Ki value to use for PID control. Defaults to 0.0075.

kd (float): Kd value to use for PID control. Defaults to 0.01.

heater_segments (int): the pseudo-control range for the internal heat_controller object. Defaults to 8.

**auto_connect**()
Starts a thread that will automatically connect to the roaster when it is plugged in.

**connect**()
Attempt to connect to hardware immediately. Will not retry. Check freshroastsr700.connected or freshroastsr700.connect_state to verify result. Raises:

**freshroastsr700.exeptions.RoasterLookupError** No hardware connected to the computer.

**connect_state**
A getter method for _connect_state. Indicates the current connection state this software is in for FreshRoast SR700 hardware. Returns:

**freshroastsr700.CS_NOT_CONNECTED** the software is not currenting communicating with hardware, neither was it instructed to do so. A previously failed connection attempt will also result in this state.

**freshroastsr700.CS_ATTEMPTING_CONNECT** A call to auto_connect() or connect() was made, and the software is currently attempting to connect to hardware.

**freshroastsr700.CS_CONNECTED** The hardware was found, and the software is communicating with the hardware.

**connected**
A getter method for _connected. Indicates that the this software is currently communicating with FreshRoast SR700 hardware.

**cool**()
Sets the current state of the roaster to cool. The roaster expects that cool will be run after roast, and will not work as expected if ran before.

**current_temp**
Current temperature of the roast chamber as reported by hardware.

**Returns:** (int) current temperature, in degrees Fahrenheit

**disconnect**()
Stops the communication loop to the roaster. Note that this will not actually stop the roaster itself.

**fan_speed**
Get/Set fan speed. Can be 1 to 9 inclusive.

**Args:** Setter: fan_speed (int): fan speed

**Returns:** Getter: (int): fan speed

**get_roaster_state**()
Returns a string based upon the current state of the roaster. Will raise an exception if the state is unknown.

**Returns:** 'idle' if idle, 'sleeping' if sleeping, 'cooling' if cooling, 'roasting' if roasting, 'connecting' if in hardware connection phase, 'unknown' otherwise

**heat_setting**
Get/Set heat setting, 0 to 3 inclusive. 0=off, 3=high. Do not set when running freshroastsr700 in thermostat mode.

**Args:** Setter: heat_setting (int): heat setting

**Returns:** Getter: (int): heat setting

**heater_level**
A getter method for _heater_level. When thermostat=True, value is driven by built-in PID controller. When ext_sw_heater_drive=True, value is driven by calls to heater_level(). Min will always be zero, max will be heater_segments (optional instantiation parameter, defaults to 8).

**idle**()
Sets the current state of the roaster to idle.

**roast**()
Sets the current state of the roaster to roast and begins roasting.

**set_state_transition_func**(*func*)
THIS FUNCTION MUST BE CALLED BEFORE CALLING freshroastsr700.auto_connect().

Set, or re-set, the state transition function callback. The supplied function will be called from a separate thread within freshroastsr700, triggered by a separate, internal child process. This function will fail if the freshroastsr700 device is already connected to hardware, because by that time, the timer process and thread have already been spawned.

**Args:** state_transition_func (func): the function to call for every state transition. A state transition occurs whenever the freshroastsr700's time_remaining value counts down to 0.

**Returns:** nothing

**sleep**()
Sets the current state of the roaster to sleep. Different than idle in that this will set double dashes on the roaster display rather than digits.

**state_transition_run**(*event_to_wait_on*)
This is the thread that listens to an event from the timer process to execute the state_transition_func callback in the context of the main process.

**target_temp**
Get/Set the target temperature for this package's built-in software PID controler. Only used when freshroastsr700 is instantiated with thermostat=True.

**Args:** Setter: value (int): a target temperature in degF between 150 and 551.

**Returns:** Getter: (int) target temperature in degF between 150 and 551

**terminate**()
Stops the communication loop to the roaster and closes down all communication processes. Note that this will not actually stop the roaster itself. You will need to instantiate a new freshroastsr700 object after calling this function, in order to re-start communications with the hardware.

**time_remaining**
The amount of time, in seconds, remaining until a call to the state_transition_func is made. can be set to an arbitrary value up to 600 seconds at any time. When a new value is set, freshroastsr700 will count down from this new value down to 0.

time_remaining is decremented to 0 only when in a roasting or cooling state. In other states, the value is not touched.

**Args:** Setter: time_remaining (int): tiem remaining in seconds

**Returns:** Getter: time_remaining(int): time remaining, in seconds

**total_time**
The total time this instance has been in roasting or cooling state sicne the latest roast began.

**Returns:** total_time (int): time, in seconds

**update_data_run**(*event_to_wait_on*)
This is the thread that listens to an event from the comm process to execute the update_data_func callback in the context of the main process.

**class** freshroastsr700.**heat_controller**(*number_of_segments=8*)
Bases: object

A class to do gross-level pulse modulation on a bang-bang interface.

**Args:** number_of_segments (int): the resolution of the heat_controller. Defaults to 8. for number_of_segments=N, creates a heat_controller that varies the heat between 0..N inclusive, in integer increments, where 0 is no heat, and N is full heat. The bigger the number, the less often the heat value can be changed, because this object is designed to be called at a regular time interval to output N binary values before rolling over or picking up the latest commanded heat value.

**about_to_rollover**()
This method indicates that the next call to generate_bangbang_output is a wraparound read. Use this to determine if it's time to pick up the latest commanded heat_level value and run a PID controller iteration.

**generate_bangbang_output**()
Generates the latest on or off pulse in the string of on (True) or off (False) pulses according to the desired heat_level setting. Successive calls to this function will return the next value in the on/off array series. Call this at control loop rate to obtain the necessary on/off pulse train. This system will not work if the caller expects to be able to specify a new heat_level at every control loop iteration. Only the value set at every number_of_segments iterations will be picked up for output! Call about_to_rollover to determine if it's time to set a new heat_level, if a new level is desired.

**heat_level**
Set/Get the current desired output level. Must be between 0 and number_of_segments inclusive.

**Args:** Setter: value (int): heat_level value, between 0 and number_of_segments inclusive.

**Returns:** Getter (int): heat level

## freshroastsr700.utils module

freshroastsr700.utils.**find_device**(*vidpid*)
Finds a connected device with the given VID:PID. Returns the serial port url.

freshroastsr700.utils.**frange**(*start*, *stop*, *step*, *precision*)
A generator that will generate a range of floats.

freshroastsr700.utils.**seconds_to_float**(*time_in_seconds*)
Converts seconds to float rounded to one digit. Will cap the float at 9.9 or 594 seconds.

## freshroastsr700.pid module

**class** freshroastsr700.pid.**PID**(*P*, *I*, *D*, *Derivator=0*, *Integrator=0*, *Output_max=8*, *Output_min=0*)
Bases: object

Discrete PID control.

**getDerivator**()

**getError**()

**getIntegrator**()

**getPoint**()

**setDerivator**(*Derivator*)

**setIntegrator**(*Integrator*)

**setKd**(*D*)

**setKi**(*I*)

**setKp**(*P*)

**setPoint**(*targetTemp*)
Initilize the setpoint of PID.

**update**(*currentTemp*, *targetTemp*)
Calculate PID output value for given reference input and feedback.

**update_d**(*d*)

**update_i**(*i*)

**update_p**(*p*)

### freshroastsr700.exceptions module

**exception** freshroastsr700.exceptions.**RoasterError**
Bases: Exception

A base error for freshroastsr700 errors.

**exception** freshroastsr700.exceptions.**RoasterLookupError**
Bases: *freshroastsr700.exceptions.RoasterError*

Raised when a device is not able to be found from the connected devices.

**exception** freshroastsr700.exceptions.**RoasterStateError**
Bases: *freshroastsr700.exceptions.RoasterError*

Raised when the current state of the roaster is not a known roaster state.

**exception** freshroastsr700.exceptions.**RoasterValueError**
Bases: *freshroastsr700.exceptions.RoasterError*

Raised when a class variable assigned is out of the range of acceptable values.

## FreshRoastSR700 Communication Protocol

All of the communication between the FreshRoastSR700 and the computer are happening over serial. The device contains a USB to serial adapter that uses the CH341 chip set. With this, it creates a virtual serial port that the program and roaster communicate over. Each of them send 14 byte packets back and forth between each other. Below is the basic packet structure of the serial communications between the devices.

| Header | Temperature Unit | Flags | Current State | Fan Speed | Time Remaining | Heat Setting | Current Temperature | Footer |
|---|---|---|---|---|---|---|---|---|
| 2 bytes | 2 bytes | 1 byte | 2 bytes | 1 byte | 1 byte | 1 byte | 2 bytes | 2 bytes |

An example packet would look like the following:

| Header | Temperature Unit | Flags | Current State | Fan Speed | Time Remaining | Heat Setting | Current Temperature | Footer |
|---|---|---|---|---|---|---|---|---|
| AA AA | 61 74 | 63 | 02 01 | 01 | 32 | 01 | 00 00 | AA FA |

## Packet fields

**Header (2 bytes)** - This field is 2 bytes and is almost always `AA AA`. When initializing communications with the roaster, the computer sends `AA 55`.

**Temperature Unit (2 bytes)** - The next 2 bytes are used to set the unit (Celsius or Fahrenheit) of the temperature being returned from the roaster. For Fahrenheit, this field should be `61 74`.

**Flags (1 byte)** - This field of the packet is used to determine what type of packet is being sent or received.

`63` - The packet was sent by the computer.

`00` - The packet was sent by the roaster.

`A0` - The current settings on the roaster that had been set manually.

`AA` - A beginning or middle line of a previously run recipe that had been saved to the roaster.

`AF` - Last line of a previously run recipe that had been saved to the roaster.

**Current State (2 bytes)** - This section controls the current state of the roaster. This field is responsible for making the roaster start and stop.

`02 01` - Idle (Shows current timer and fan speed values)

`04 02` - Roasting

`04 04` - Cooling

`08 01` - Sleeping (Displays "-" in both fan speed and timer fields on the roaster)

**Fan Speed (1 byte)** - This field is the current fan speed in hex. Below is a list of valid values for this field.

`01`, `02`, `03`, `04`, `05`, `06`, `07`, `08`, `09`

**Time Remaining (1 byte)** - This field is the time remaining in hex. The time remaining is a decimal representation of time as displayed on the roaster. For example, one minute and thirty seconds would appear as 1.5 on the roaster and should be set as *0F* in hex. Additionally, five minutes and fifty-four seconds would be represented as 5.9 on the roaster and `3B` in hex.

**Heat Setting (1 byte)** - This field is the heat setting for the roaster. This value will not cause the roaster to start roasting. It only dictates what the roaster will do once it begins. Below is a list of valid values.

`00` - No Heat (Cooling)

`01` - Low Heat

`02` - Medium Heat

`03` - High Heat

**Current Temperature (2 bytes)** - This field is the current temperature as recorded by the roaster encoded in hex. When the roaster does not read a temperature of 150°F or higher, it sends the following hex: `FF 00`. If the temperature is higher than 150°F, the temperature is sent encoded in hex. For example, 352°F is `01 60`.

**Footer (2 bytes)** - This field signifies the end of a packet and is always `AA FA`.

### Packet Sequences

The FreshRoastSR700 has a distinct packet sequence that must be followed in order to communicate with the roaster.

To start off, every communication between the roaster and the computer is initiated by the computer. It is initiated with a packet that looks like the packet below.

`AA 55 61 74 63 00 00 00 00 00 00 00 AA FA`

This packet is a blank packet with the header set to `55`. This then signals the roaster to send back the last recipe that had been loaded onto the roaster. Below is an example of the data that roaster would send back.

`AA AA 61 74 A0 00 00 09 3B 02 00 00 AA FA` – Manual setting currently on the roaster.

`AA AA 61 74 AA 00 00 09 03 03 00 00 AA FA` – First line of the recipe currently on the roaster.

`AA AA 61 74 AA 00 00 09 01 02 00 00 AA FA` – Second line of the recipe currently on the roaster.

`AA AA 61 74 AF 00 00 09 1C 00 00 00 AA FA` – Last line of the recipe currently on the roaster.

The roaster sends the above packets one right after another. It doesn't wait for the computer to respond until after the last line of sequence is sent. The last packet sent is denoted by `AF` in the flags field.

After this, the computer sends back the heat setting, fan speed, and time remaining it wants the roaster to be set to. This is all sent in a single packet like the following.

`AA AA 61 74 63 02 01 01 3B 01 00 00 AA FA` – heat=low, fan speed=1, time=5.9 minutes

The roaster then sends back the current settings including the current temperature of the roaster if it's 150°F or higher. The response packet would look like the following.

`AA AA 61 74 00 02 01 01 32 01 FF 00 AA FA`

This continues indefinitely until the connection is closed. A packet should be sent from the computer every quarter of a second, and no sooner. When the roaster should begin roasting, set the current state to roasting. The roaster cannot go directly to cooling, and must be first set to roasting.

## Contributing

### Setting up a development environment

```
git clone git@github.com:Roastero/freshroastsr700.git
cd freshroastsr700
virtualenv venv -p python3
source venv/bin/activate
python setup.py develop
```

### Running tests

This module uses tox to run tests and a code linter. Run the commands below in the base project directory to install everything needed and run tests on the freshroastsr700 module.

```
pip install -r test-requirements.txt
tox
```

# Release Notes

## Version 0.2.4 - Oct 2017

Resolves feature request documented in issue #31 freshroastsr700 object can now be instantiated with manual control of the software-based heater algorithm. Tested in Ubuntu 16.04.

## Version 0.2.3 - May 2017

Resolves issues #22, 23, 24 and 25, and 29 (the latter introduced by 0.2.2). Added logic to handle hardware connects and hardware disconnects properly in all supported OSes. Software now supports multiple connect()-disconnect() cycles using the same freshroastsrs700 object instance. Tested in Windows 10 64-bit and Ubuntu 14.04.

## Version 0.2.2 - May 2017

[Introduced issue #29. Inoperable in Windows environments - do not use.]

## Version 0.2.1 - March 2017

Resolves issue #20 by managing hardware discovery logic in the comm process, eliminating the need for the thread heretofore associated with auto_connect. Openroast 1.2 (currently in development) now operates properly in Windows 10 64-bit, with this fix.

## Version 0.2.0 - March 2017

Completely rewritten PID control for tighter tracking against target temperature (when freshroastsr700 is instantiated with thremostat=True). Callback functions for update_data_func and state_transition_func now called from a thread belonging to the process that instantiated freshroastsr700. This was necessary for Openroast version 1.2 code refactoring. Reduced processor load for PID control as part of code refactoring.

## Version 0.1.1 - Dec 28 2017

Added support for python 2.7.

## Version 0.1.0

(no notes)

## Version 0.0.6

(no notes)

CHAPTER 2

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## f

# Index

## U