**hyperdaq.main**

main.py

A module for defining the main User Interface and main thread of the program

Last Updated: January 2020

Trevor Arp

Gabor Lab

University of California, Riverside

All Rights Reserved

***class*hyperdaq.main.Control\_Thread(*command\_q*, *main\_UI*, *device\_dict*, *scanner*, *data\_out*)**

Bases: **threading.Thread**

The thread that controls scanning and other active functions, while allowing the GUI to continue updating

Accepts commands from a queue, while it is performing the task self.active is true, this can be used to block input from the GUI while the task is working

Will process two kinds of commands: (1) It will run a hardware command, for example moving the delay stage into position, called with command [function, args], where function is a hardware function that takes args (list of arguments)

1. It will run a scan based on a valid Scan\_Spec object passed to it

**drift\_buffer\_images()**

Buffers the current images for use in future drift correction

**drift\_correction(*outputs*)**

Calculate and apply the applicable drift correction

WARNING: RFI CORRECTION UNTESTED AS OF 2020/1/14

**init\_drift\_buffer(*outputs*)**

Resets the drift correction buffer

**log\_params(*specification*, *outputs*)**

Log the basic scan parameters and the required scanning parameters

**perform\_scan(*specification*, *outputs*, *dim*, *scanNum=''*)**

Perform a scan, either an individual scan or a data cube. parameter dim is the scan dimension to consider, if the dimension is higher than the output (scan or cube) repeat the scan recursively along that axis.

**process\_scan(*specification*)**

Executes a scan based on the given specification

**run()**

The main loop of the program, called in the new thread.

**starting\_card\_output\_values(*specification*)**

Compute the starting values for the voltage output channels

**stop()**

Stops the active process

***class*hyperdaq.main.Scan\_Spec**

Bases: **object**

A specification for a given scan, passed to the control thread for processing.

Meant to be extensible to allow for increased functionality in the future.

The critical parameter is Scan\_Spec.dimensions, a list containing the specifications for each dimension, the index determines the order of the variables.

Each specification is a list with the following entries: [variable\_identifier, start\_value, end\_value, number\_of\_points, sampling\_function] where the sampling function is numpy.linspace by default i.e. numpy.linspace(start\_value, end\_value, number\_of\_points)

**add\_axis(*parameter*, *start*, *end*, *N*)**

Adds a scanning parameter that will be scanned linearly over the range [start, end] taking N samples.

If the parameter was previously defined as a constant this will overwrite it.

**add\_constant(*parameter*, *value*)**

Adds a scanning parameter that will be held constant during the scan.

**Parameters**

* **parameter** (*str*) – the parameter to hold constant
* **value** (*float*) – the value to hold it at

**add\_cube\_axis(*parameter*, *start*, *end*, *N*)**

Adds a scanning parameter as described in add\_axis.

If this function is called the scan will save as 3D cubes instead of 2D scans.

**sampleaxis(*dim*)**

Returns the samples to an axis based on the specified sampling function, for an axis with index dim. If dim is 0 or 1, sampleaxis may not be called as that is usually set by the scanning function, be sure to check.

**set\_drift\_correction(*type*)**

Enable drift correction and set which image to use

**set\_scan\_type(*type*)**

Change the scan type from default

**verify()**

Returns true if all required scan parameters are present and there are at least two axes, false otherwise.

Required parameters are defined by SCAN\_spatial\_parameters and SCAN\_voltage\_parameters.

***class*hyperdaq.main.axis\_selector(*gui*, *master*, *label*, *options*, *optional=False*, *opt\_num\_label=None*, *default=None*)**

Bases: **object**

**get\_entries()**

Return the start and end entries (and number if it is an optional axis)

**param\_select(*event*)**

Change selection of the parameter

***class*hyperdaq.main.hyperDAQ(*images*, *auximages*, *card\_control*, *data\_writer*, *device\_dict*, *interface\_dict*, *scan\_params*, *exp\_params*, *exp\_params\_units*, *showdrift=True*)**

Bases: **object**

The main user interface and control for the DAQ. Meant to be generic, extended using inheritance.

To add new axes to this generic interface use inheritance and override the function init\_extra\_axes, which sets any additional axes beyond the fast, slow and cube

**Parameters**

* **card\_in** – is the data input object reading from the NI DAQ card
* **images** – is a list of images to display
* **auximages** – is a dictionary of auxiliary images, that are not displayed but will still be updated. The keys allow the images to be specifically accessed by various interfaces. Set to None if there are no auxiliary images
* **card\_control** – is the controller for the DAQ card
* **scan\_params** – is a dictionary containing the parameters that it is possible to scan over, dictionary is formatted as “device\_key”:[“Label”, “unit”, fast\_function, slow\_function, (default\_start, default\_end)] where the device\_key can be used to find the device in the device dict or identifier for a voltage output, label, unit and defaults are displayed and fast\_function/slow\_function are the functions that can be used to scan the device over the fast and slow axes, respectively (or None for voltage output).

**abort\_scan()**

Abort the scan

**autoscale\_images\_callback()**

Turns on auto-scaling of images

**change\_img\_size(*nx*, *ny*)**

Change the size of the images, called internally

**check\_value(*key*, *value*)**

Checks if a value is valid for a given key to self.scan\_params

**display\_Error(*s*)**

Prints the given string in red as an error

**display\_Text(*s*)**

display\_Texts the given string

**display\_Warning(*s*)**

Prints the given string in red as a warning

**drift\_update\_center\_values(*dx*, *dy*)**

Updates the center values of X and Y with drift correction, only works with the ‘xaxis’ and ‘yaxis’ parameters, will not work with voltage or hardware parameter

**edit\_params\_callback()**

Sets the value of an experimental parameter

**image\_lims\_callback(*ix*, *label*)**

Sets a maximum and minimum value for the photocurrent image

**init\_device\_wrapper\_frame(*master*, *interface\_dict*)**

Initializes the frame containing the Device UIs and the Experimental parameters

**init\_drift\_frame(*master*)**

Initializes the frame to correct for drift

**init\_extra\_axes(*options*)**

**Initialize any additional axes beyond the main three (fast, slow, cube). By default creates**

one repeat axis. Override to add additional axes or extend functionality

The parameter is the options to give the axis (same as cube axis options)

Should return a list (in order of increasing higher dimensions), that will get added to the interface in order, and be accessed by various functions in order.

**init\_logger()**

Initialize the logger

**init\_menu()**

Initializes the main file menu

**init\_message\_frame(*master*)**

Initializes the frame containing message area

**init\_parameters\_frame(*master*)**

Initialize the frame to display experimental parameters

**init\_plotting\_frame(*master*)**

Initializes the frame containing the plots

**init\_scanning\_frame(*master*)**

Initializes the Scanning Frame, Initialize the frame and all the sub-frames

**load\_scan(*n*)**

Loads a scanning configuration from file

**popup\_entry(*title*, *spec*, *entry\_width=None*)**

Function that can be used to get input from a user, for use in callback functions

**Parameters**

* **title** – The title of the popup window
* **spec** – An array that defines the behavior of the popup, for each variable to retrieve spec contains a list specifying what to say before and after and the starting value to be displayed. For example for a popup asking for two quantities spec = [[‘Quantity A’, ‘Units A’, Starting\_Value], [‘Quantity B’, ‘Units B’, Starting\_Value]]

**Returns**

When done a list is returned containing user input [Quantity\_A, Quantity\_B]

**resize\_callback()**

Resizes the images

**save\_scan(*n*)**

Saves the current scanning configuration to file

**set\_aspect(*nx*, *ny*)**

Changes the aspect ratio of the images

**set\_linerate\_callback()**

Sets the line scan variable in the scanner

**start\_scan()**

Gathers the scan parameters from the interface and passes the scan specification off to the control thread.

**stop()**

Stop the treads of all devices and close the program

**toggle\_record()**

Toggles the recording on the front panel

**update()**

Updates the plots

**update\_callback()**

Calls the update function after PLT\_update\_delay (assuming the system isn’t busy)

**voltage\_lock\_callback()**

Handles the locking of voltage for all parameters defined in SCAN\_voltage\_parameters