Polo Documentation

Release 0.0.1

Ethan T. Holleman

Jun 15, 2020

CONTENTS:

CHAPTER

ONE

POLO QUICKSTART GUIDE

1.1 Windows

Do this for Windows machines

1.2 Linux

Do this for linux (Ubuntu)

CHAPTER

TWO

FAQS

A place to include frequently asked questions

CHAPTER

THREE

USER'S GUIDE

If you have yet to install Polo on your machine, head to the installation guide here.

3.1 Importing and Opening Image Data

The first step to using Polo is adding your own data. Polo organizes data into "runs", which consist of a set of related screening images. Polo organizes runs into three different categories which are described below.

- HWI Screening Runs
- Non-HWI Screening Runs
- Raw Image Collections

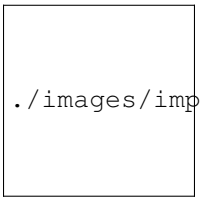
3.1.1 Getting Your Data (via FTP)

If you do not already have your data downloaded to your local machine, Polo includes an FTP file browser which utilizes Python's ftplib package. For HWI users this allows you to download your screening images from the HWI server without leaving the application.

To open the FTP file browser, on the menubar navigate to Import -> Images -> From FTP. Enter your credentials in the new window.Once you are connected to a server files available to download will be listed in the browser menu. Select the checkboxes next to the files you wish to download, or select all files in a directory at once. Pressing "Download Selected Files" will start the download.

3.1.2 Using the Run Importer

To import a new run into Polo for classification head to the menubar and click Import -> Images -> New Run. This will open a new window that will look similar to the one below.



The selection box under "Import Type" lists the three run types you can import. Only one type may be imported at a time. Fill in the fields listed for the run type you wish to import and hit "submit run" to import your image data into Polo. Once you select a directory Polo will fill in fields automatically based on any metadata it finds. This is done

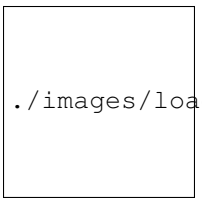
most extensively for HWI Screening Runs and unless you know what you are doing it is recommended to use the Polo suggested settings.

3.1.3 Importing a Saved Run

One of the advantages of Polo is the introduction of the .xtal file format. Xtal is a json file that can store all your run data, including images, classification and other metadata in a single, portable format. If you have an xtal file from a previously saved run you can load it into Polo by navigating to the menubar and selecting Import -> Images -> From Saved Run. A file browser will open and you can select your .xtal file and pick up where you left off.

3.1.4 Opening a Run

Once a run has been successfully imported, the run name will appear in the loaded runs list, like in the image below.

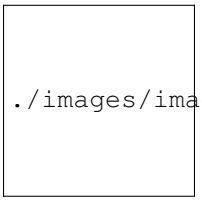


If it is a new run double clicking on the run name will run the MARCO model on the run's images. You can check on MARCO's progress by using the Classification Progress bar located just below the Loaded Runs list.

Once a run has been classified you can load it into the current view by double clicking on it again.

3.2 Using the Slideshow View

The slideshow view is the main Polo user interface. It allows you to view your screening images, label them and filter them by MARCO classifications, your own classifications or both.



3.2.1 Navigation

Once you have images loaded in you can cycle through images by pressing the next or previous image buttons under the navigation panel. You can also use the right or left arrowkeys respectively.

If you are viewing an HWI Screening Run you will be able to toogle between ordering images by index (well numbe) or by cocktail number. Also for HWI runs, you can navigate directly to an image by well number by using the By Well Number selector also in the Navigation panel.

It is also posssible to move between runs that may be of the same protein sample but imaged at a different time or with a different optical technology. See the Advanced Settings section for more details on these features.

3.2.2 Classification

Arguebly the most important Polo feature is the ability to easily label your screening images. To label the currently displayed image press the button in the classification panel with your desired label. You can classify images as Crystal, Precipitate, Clear or Other. See this link for more details on each classifiction. To increase your speed you can classify images using keyboard shortcuts which are listed below.

- A: Crystal
- W: Precipitate
- S: Clear
- D: Other

Classifiying an image will automatically move you to the next image in the slideshow.

3.2.3 Filtering

Using the checkboxes under the Image Filter's panel in the lower right corner of the window will allow you to filter the kinds of images in your current slideshow. For example if you only wanted to see images that MARCO has classified as Crystal you could check the Crystal box under Image Types and MARCO under the Classifier panel. If you had checked Human instead only images that you have classified as Crystal would be shown.

You can reset the slideshow to include all images by selecting all checkboxes or no boxes and pressing submit filters.

3.2.4 Image Metadata

Image metadata will be displayed in the Image Details and Cocktail Details windows when it is available. Image details will give you basic information about the image currently being displayed, such as well number, imaging technology imaging date and current classifications. If you are viewing an HWI run the chemical conditions the current image was plated in will be displayed in the Cocktail Details window.

3.3 Using the Table View

3.4 Using Plot Functions

3.5 Editing Run Data After Loading

3.5.1 Editing Run Date

3.5.2 Editing Run Name

3.5.3 Adding Annotations

3.5.4 Deleting a Run

3.6 Saving a Run

3.6.1 .xtal File Format

3.7 Exporting a Run

3.7.1 HTML and PDF Reports

3.7.2 CSV Exports

CHAPTER

FOUR

POLO PACKAGE

4.1 Subpackages

4.1.1 polo.crystallography package

Submodules

polo.crystallography.broke module

polo.crystallography.cocktail module

class polo.crystallography.cocktail.**Cocktail**(*number=None*, *well_assignment=None*, *commercial_code=None*, *pH=None*, *reagents=None*)

Bases: object

add_reagent(*new_reagent*)

Adds a reagent to the existing list of reagents stored in the reagents attribute.

Parameters **new_reagent** (Reagent) – Reagent to add to this cocktail

property cocktail_index

Attempt to pull out the cocktail number from the cocktail number string. Dependent on consistent formating between cocktail menus that I have not currently varrified.

Returns cocktail number

Return type int

property well_assignment

Return the current well assignment for this cocktail

Returns well assignment

Return type int

class polo.crystallography.cocktail.**Reagent**(*chemical_additive=None*, *concentration=None*, *chemical_formula=None*, *stock_con=None*)

Bases: object

property chemical_formula

Return the current chemical formula for this Reagent. It is not certain that a reagent will have a chemical formula.

Returns Chemical formula

Return type molmass.Formula

property concentration
Return the current concentration for this reagent. Concentration utimately refers back to a condition in a specific screening well.

Returns Chemical concentration

Return type *SignedValue*

property molar_mass
Attempt to calculate the molar mass of this reagent. Closely related to the molarity property. See its docstring for why this is not always possible. Return False if cannot be calculated.

Returns Molarity or False

Return type *SignedValue* or False

property molarity
Attempt to calculate the molarity of this reagent at its current concentration. This calculation is not certain to return a molarity because HWI cocktail menu files use a variety of units to describe chemical concentrations, including %w/v or %v/v. Currently, Polo is not able to convert %v/v units to molarity as it would require knowing both the molar mass of the reagent and its density. If the reagent concentration cannot be converted to mols / liter then returns false.

Returns molarity or False

Return type *SignedValue* or Bool

peg_parser(*peg_string*)
Attempts to pull out a molar mass from a chemical name since the molar mass is often included in the name of PEG species. A string is considered to be a potential PEG species if it contains 'PEG' or 'Polyethylene glycol' in it.

Parameters **peg_string** (*str*) – String to look for PEG species in

Returns molar mass if found to be valid PEG species, False otherwise.

Return type float or Bool

stock_volume(*target_volume*)
Attempt to calculate the required amount of stock solution to produce the reagent's set concentration in the given target_volume. Stock concentration is taken from the stock_con attribute. If stock_con is not set or the molarity of the reagent can not be calculated this method will return False.

Parameters **target_volume** (SignedValue) – Volume in which stock will be diluted into

Returns Volume of stock or False

Return type *SignedValue* or False

units = ['M', '(w/v)', '(v/v)']

class polo.crystallography.cocktail.**SignedValue**(*value=None*, *units=None*)
Bases: object

classmethod make_from_string(*string*)

property micro

property milli

property nano

set_from_string(*string*)

supported_units = {'L', 'M', 'X', 'v/v', 'w/v'}

property units

property value

polo.crystallography.image module

class polo.crystallography.image.**Image**(*path=None*, *bites=None*, *well_number=None*, *human_class=None*, *machine_class=None*, *prediction_dict=None*, *plate_id=None*, *date=None*, *cocktail=None*, *spectrum=None*, *previous_image=None*, *next_image=None*, *alt_image=None*)

Bases: object

DEFAULT_IMAGE = ''

classify_image()

encode_base64()

get_cocktail_number()
Returns the numerical cocktail number as an integer. If it does not exist returns 0.

get_pixel_map()

get_tool_tip()

resize(*x*, *y*, *preserve_aspect=True*)
Resizes an image given x and y resolution. Copy is true will copy the image instead of overwriting it.

polo.crystallography.make_screen module

polo.crystallography.make_screen.**calculate_row**(*volume*, *x_con*, *x_stock_con*, *y_stock_con*, *y_con*, *y_step*, *steps*)

polo.crystallography.make_screen.**get_dilution**(*target_con*, *stock_con*)

polo.crystallography.make_screen.**get_mols**(*total_volume*, *target_con*)

polo.crystallography.make_screen.**get_stock_volume**(*target_con*, *total_volume*, *stock_con*)

polo.crystallography.make_screen.**get_usable_volume**(*total_volume*)

polo.crystallography.make_screen.**make_tray**(*total_volume*, *hit_concentration*, *total_wells*, *sample_concentration*, *sample_volume_per_well*, *stock_cocentration*, *step_percent*)

polo.crystallography.run module

class polo.crystallography.run.**HWIRun**(*image_dir*, *run_name*, *cocktail_dict=None*, *images=[]*, *plate_id=None*, *annotations=None*, *save_file_path=None*, *num_wells=1536*, *image_spectrum=None*, *date=None*, *next_run=None*, *previous_run=None*, *alt_spectrum=None*, *number_grid_pages=None*, *current_grid_page=1*, *journal=None*, *current_image_index=0*, *current_image=None*, **args*, ***kwargs*)

Bases: *polo.crystallography.run.Run*

AllOWED_PLOTS = ['Classification Counts', 'MARCO Accuracy', 'Classification Progress',

Child class of Run. Is used to represent runs from the HWI screening center as images will have additional metadata like well and cocktail information. Main difference is that HWIRuns will always contain 1536 images as that is the number of wells in a HWI crystallization plate. Each well uses a different chemcial cocktail which is described in the cocktail tsv file included in the directory of images provided by HWI in each run.

add_images_from_dir()

Populates the images attribute with a list of images read from the image_dir location. Currently is dependent on having a cocktail dictionary available. This is passed into the function and would normally come from the most recently used HWI file that is stored as a dictionary in the mainWindow object.

link_to_alt_images(*other_run*)

Establish linked lists between this run and another run instance that holds images of a different imaging spectrum / technology.

Parameters **other_run** (Run) – A different run instance

link_to_alt_spectrum(*other_run*)

link_to_decendent(*other_run*)

link_to_predecessor(*other_run*)

sort_current_images_by_cocktail()

Sorts the current slideshow images by cocktail number. Allows the user to navigate by cocktail number and therefore similar chemical conditions opposed to as by well number which is proxy for physical location in the well. Oftentimes due to plate shape similar well numbers will be in different family of chemcial conditions.

class polo.crystallography.run.**Run**(*image_dir*, *run_name*, *images=None*, *save_file_path=None*, *log=None*, *date=None*, *image_spectrum=None*, *next_run=None*, *previous_run=None*, *alt_spectrum=None*, *journal={}*, *current_image=None*, *current_image_index=0*, **args*, ***kwargs*)

Bases: object

Holds data relating to an individual screening run, or one plate of images.

AllOWED_PLOTS = ['Classification Counts', 'MARCO Accuracy', 'Classification Progress']

add_images_from_dir()

Adds the contents of a directory to self.images

TODO: Add validation for file types and content. Handle if user gives a directory where there are no images or edge cases like that.

add_journal_entry(*contents*, *title*)

encode_images_to_base64()

export_to_csv(*output_dir*)
Exports run classifcation data to a csv table.

get_cocktails()
Returns list of list of cocktails assigned to this run

get_current_hits()

get_current_table_data(*image_types*, *human=True*, *marco=False*)

get_heap_map_data()
Returns data that will be needed to render the heatmap view of results

get_human_statistics()
Returns stats that would be shown in the stats tab of the viewer.

get_image_table_data(*image*, *attributes*)

get_images_by_classification(*human=True*)
Create a dictionary of image classifcations. Keys are each type of classification and values are list of images with classification of the key. The human boolean determines what classifier should be used to determine the image type. Human = True sets the human as the classifier and False sets MARCO as the classifier.

get_table_data(*image_types*, *human*, *marco*)

image_filter_engine(*image*, *image_types*, *human=False*, *marco=False*)

image_filter_query(*image_types*, *human=False*, *marco=False*)

Module contents

4.1.2 polo.marco package

Submodules

polo.marco.run_marco module

polo.marco.run_marco.**classify_image**(*tf_predictor*, *image_path*)
Given a tensorflow predictor (the MARCO model) and the path to an image, runs the model on that image. Returns a tuple where the first item is the classification with greatest confidence and the second is a dictionary where keys are image classification types and values are model confidence for that classification.

param: tf_predictor: Tensorflow predictor object. Should be MARCO model in ready to roll form. param: image_path: String. Path to an image that will be classified.

polo.marco.run_marco.**get_images**(*images_path*)
returns a list of all file paths contained in the given directory.

polo.marco.run_marco.**load_image**(*file_path*)

polo.marco.run_marco.**load_images**(*file_list*)
Loads images from a list of paths (should be from get_images) in format that can be read by the tensorflow package

Module contents

4.1.3 polo.utils package

Submodules

polo.utils.exceptions module

exception polo.utils.exceptions.**EmptyDirectoryError**
Bases: Exception

Raised when attempting to load images from an empty directory

exception polo.utils.exceptions.**EmptyRunNameError**
Bases: Exception

Raised when reading in an HWI directory but it does not contain number of images corresponding to number of wells.

exception polo.utils.exceptions.**ForbiddenImageTypeError**
Bases: Exception

Raised when user attempts to load in an image that is not in the allowed image types.

exception polo.utils.exceptions.**IncompletePlateError**
Bases: Exception

Raised when reading in an HWI directory but it does not contain number of images corresponding to number of wells.

exception polo.utils.exceptions.**InvalidCocktailFile**
Bases: Exception

Raised when user attempts to load in a file containing well cocktail information that does not confrom to existing formating standards.

exception polo.utils.exceptions.**NotASolutionError**
Bases: Exception

Raised when user attempts to load in an image that is not in the allowed image types.

exception polo.utils.exceptions.**NotHWIDirectoryError**
Bases: Exception

Raised when user attempts to read in a directory as HWI but it does not look like one.

TODO: Add utils function to determine when to raise this exception.

polo.utils.ftp_utils module

polo.utils.ftp_utils.**catch_ftp_errors**(*funct*)
General decorator function for catching any errors thrown by other ftp_utils functions

polo.utils.ftp_utils.**get_cwd**(**args*, ***kwargs*)

polo.utils.ftp_utils.**list_dir**(**args*, ***kwargs*)

polo.utils.ftp_utils.**logon**(**args*, ***kwargs*)

polo.utils.ftp_utils.**traverse_folder**(*cwd*, *tree*, *ftp*)

polo.utils.io_utils module

class polo.utils.io_utils.**BarTender**(*cocktail_dir*, *cocktail_meta*)
Bases: object

Class for organizing and accessing cocktail menus

add_menus_from_metadata()
Adds menu objects to the menus attribute.

static date_range_parser(*date_range_string*)

Utility function for converting the date ranges in the cocktail.units metadata csv file to datetime objects using the *datetime_converter* classmethod.

Date ranges should have the format

start date - end date

If the date range is for the most recent cocktail menu then there will not be an end date and the format will be

start date -

Parameters **date_range_string** (*str*) – string to pull dates out of

Returns tuple of datetime objects, start date and end date

Return type tuple

static datetime_converter(*date_string*)

General utility function for converting strings to datetime objects..units Attempts to convert the string by trying a couple of datetime.units formats that are common in cocktail menu files and other.units locations in the HWI file universe Polo runs across.

Parameters **date_string** (*str*) – string to convert to datetime

Returns datetime object

Return type datetime

get_menu_by_date(*date*, *type_*)
Get a menu instance who's usage dates include the given date and match the given screen type.

Screen types can either be 's' for 'soluble' screens or 'm' for membrane screens.

Parameters

- **date** (*datetime*) – Date to search menus with
- **type** (*str*) – Type of screen to return (soluble or membrane)

Returns menu matching the given date and type

Return type *Menu*

get_menu_by_path(*path*)
Returns a menu by its file path, which is used as the key for accessing the menus attribute normally.

Parameters **path** (*str*) – file path of a menu csv file

Returns Menu instance that is mapped to given path

Return type *Menu*

get_menus_by_type(*type_*)
Returns all menus of a given screen type.

's' for soluble screens and 'm' for membrane screens. No other characters should be passed to *type_*.

Parameters **type** (*str (max length 1)*) – Key for type of screen to return

Returns list of menus of that screen type

Return type list

class polo.utils.io_utils.**CocktailMenuReader**(*menu_file*, *delim=','*, ***kwargs*)
Bases: object

cocktail_map = {0: 'well_assignment', 1: 'number', 2: 'commercial_code', 8: 'pH'}

formula_pos = 4

property menu_file
Get the instances menu file

Returns the menu file path

Return type str or Path or IO

classmethod set_cocktail_map(*map*)
Classmethod to edit the cocktail_map. The cocktail map describes where Cocktail level information is stored in a given cocktail menu file row. It is a dictionary that maps specific indicies in a row to the Cocktail attribute to set the value of the key index to.

The default cocktail_map dictionary is below.

cocktail_map = { 0: 'well_assignment', 1: 'number', 8: 'pH', 2: 'commercial_code' }

This tells instances of CocktailMenuReader to look at index 0 of a row for the well_assignment attribute of the Cocktail class, index 1 for the number attribute of the Cocktail class, etc.

Parameters **map** (*dict*) – Dictionary mapping csv row indicies to Cocktail object attributes

classmethod set_formula_pos(*pos*)
Classmethod to change the formula_pos attribute. The formula_pos describes the location (base 0) of the chemical formula in a row of a cocktail menu file csv. For some reason, HWI cocktail menu files will only have one chemical formula per row (cocktail) no matter the number of reagents that composite that cocktail. This is why its location is represented using an int instead of a dict.

Generally, formula_pos should not be changed without a very good reason as the position of the chemical formula is consistent across all HWI cocktail menu files.

Parameters **pos** (*int*) – Index where chemical formula can be found

class polo.utils.io_utils.**HtmlWriter**(*run*, ***kwargs*)
Bases: *polo.utils.io_utils.RunSerializer*

finished_writing()
Method to connect to Qthread instance. Should not be called from anything other than a Qthread instance.

static make_template(*template_path*)
Given a path to an html file to serve as a jinja2 template, read the file and create a new template object.

write_complete_run(*output_path*, *encode_images=True*)

write_complete_run_on_thread(*output_path*, *encode_images*)
Wrapper around *write_complete_run* that executes on a seperate Qthread.

write_grid_screen(*output_path*, *plate_list*, *well_number*, *x_reagent*, *y_reagent*, *well_volume*, *run_name=None*)

Write the contents of optimization grid screen to an html file

Parameters

- **output_path** (*str*) – Path to html file
- **plate_list** (*list*) – list containing grid screen data
- **well_number** (*int or str*) – well number of hit screen is created from
- **x_reagent** (Reagent) – reagent varried in x direction
- **y_reagent** (Reagent) – reagent varried in y direction
- **well_volume** (*int or str*) – Volume of well used in screen
- **run_name** (*str, optional*) – name of run, defaults to None

class polo.utils.io_utils.**Menu**(*path*, *start_date*, *end_date*, *type_*)

Bases: object

property cocktails

Property to return the Menu instance's cocktail dict

Returns cocktail attribute

Return type dict

property path

Property to return the Menu instance's path attribute

Returns The path attribute

Return type str or IO

class polo.utils.io_utils.**RunDeserializer**(*xtal_path*)

Bases: object

static clean_base64_string(*string*)

Image instances may contain byte strings that store their actual crystallization image encoded as base64. Previously, these byte strings were written directly into the json file as strings causing the b' byte string identifier to be written along with the actual base64 data. This method removes those artifacts if they are present and returns a clean byte string with only the actual base64 data.

Parameters **string** (*str*) – a string to interogate

Returns byte string with non-data artifacts removed

Return type bytes

static dict_to_obj(*our_dict*)

Oposite of the obj_to_dict method in XtalWriter class, this method takes a dictionary instance that has been previously serialized and attempts to convert it back into an object instance.

Parameters **our_dict** (*dict*) – dictionary to convert back to object

Returns an object

Return type object

xtal_header_reader(*xtal_file_io*)

Reads the header section of an open xtal file. Should always be called before reading the json content of an xtal file. Note than xtal files must always have a line of equal signs before the json content even if there is no header content otherwise this method will read one line into the json content causing the json reader to throw an error.

Parameters **xtal_file_io** (*TextIoWrapper*) – xtal file currently being read

Returns xtal header contents

Return type list

xtal_to_run()
Method that actually does the heavy lifting of converting the json contents of xtal files back into Run instances. Currently is pretty brittle and ugly so looking for a more flexible recursive solution.

Raises **e** – Error raised while reading the xtal file

Returns The contents of xtal file as a Run instance

Return type *Run*

class polo.utils.io_utils.**RunSerializer**(*run*)
Bases: object

static make_message_box(*message*, *icon=1*, *buttons=1024*, *connected_function=None*)
Return a QMessageBox instance to show to the user.

Parameters

- **message** (*str*) – Message to be displayed to the user
- **icon** (*QMessageBoxIcon, optional*) – Icon for message box, defaults to QtWidgets.QMessageBox.Information
- **buttons** (*[type], optional*) – [description], defaults to QtWidgets.QMessageBox.Ok
- **connected_function** (*[type], optional*) – [description], defaults to None

Returns [description]

Return type [type]

classmethod make_thread(*job_function*, ***kwargs*)
Creates a new qthread object. The job function is the function the thread will execute and and arguements that the job function requires should be passed has keyword arguements. These are stored as a dictionary in the new thread object until the thread is activated and they are passed as arguements.

static path_suffix_checker(*path*, *desired_suffix*)
Check is a file path has a desired suffix, if not then replace the current suffix with the desired suffix. Useful for checking filenames that are taken from user input.

Parameters **desired_suffix** – File extension for given file path.

static path_validator(*path*, *parent=False*)
Tests to ensure a path exists. Passing parent = True will check for the existance of the parent directory of the path.

class polo.utils.io_utils.**XtalWriter**(*run*, ***kwargs*)
Bases: *polo.utils.io_utils.RunSerializer*

clean_run_for_save()
Remove circular references from a Run instance to avoid errors when serializing using json. Uses the run stored in the run attribute

Returns The cleaned run

Return type *Run*

file_ext = '.xtal'

finished_save()

header_flag = '<>'

header_line = '{}{}:{}.unitsn'

static json_encoder(*obj*)
Use instead of the defauly json encoder. If the encoded object is from a module within Polo will include a module and class identifier so it can be more easily deserialized when loaded back into the program.

Param obj: An object to serialize to json.

Returns A dictionary or string version of passed object

run_to_dict()
Create a json string from the run stored in the run attribute.

Returns Run instance serialized to json

Return type str

write_xtal_file(*output_path*)
Method to serialize run object to xtal file format.

Parameters **output_path** (*str*) – Xtal file path

Returns path to xtal file

Return type str

write_xtal_file_on_thread(*output_path*)
Wrapper method around *write_xtal_file* that executes on a Qthread instance to prevent freezing the GUI when saving large xtal files

Parameters **output_path** (*str*) – Path to xtal file

property xtal_header
Creates the header for xtal file when called. Header lines are indicated as such by the string in the header_line constant, which should be '<>'. The last line of the header will be a row of equal signs and then the actual json content begins on the next line.

polo.utils.io_utils.**check_for_missing_images**(*dir_path*, *expected_image_count*)

polo.utils.io_utils.**datetime_converter**(*date_string*)

polo.utils.io_utils.**directory_validator**(*dir_path*)

polo.utils.io_utils.**export_run_to_csv**(*run*, *output_path*)

polo.utils.io_utils.**get_available_cocktails**()
Returns a list of cocktail csv files that are included with the Polo program. List is sorted from most recently used cocktails to least recently used. The year the cocktails were first used are the first two characters of the csv file name if downloaded from HWI website.

TODO: Inculde metadata file on available cocktails that gives when they were used at the screening center. Add another function to read in that metadata and create dialog so user can select cocktail that would go with their screen.

NOTE: Cocktail TSV file had unicode error in it when read using UTF-8 incoding but one on webiste did not. Need to ask Bowman about that one.

polo.utils.io_utils.**get_cocktail_number_as_int**(*cocktail_number_string*)

polo.utils.io_utils.**if_dir_not_exists_make**(*parent_dir*, *child_dir=None*)
If only parent_dir is given attempts to make that directory. If parent and child are given tries to make a directory child_dir within parent dir.

polo.utils.io_utils.**list_dir_abs**(*parent_dir*, *allowed=False*)

polo.utils.io_utils.**make_dict_from_run_via_json**(*run*)

polo.utils.io_utils.**parse_HWI_filename_meta**(*HWI_image_file*)

HWI images have a standard file nameing schema that gives info about when they are taken and well number and that kind of thing. This function returns that data

polo.utils.io_utils.**parse_cocktail_csv**(*file_path*)

polo.utils.io_utils.**parse_cocktail_metadata**()

polo.utils.io_utils.**parse_hwi_dir_metadata**(*dir_name*)

polo.utils.io_utils.**parse_reagents**(*row*)

polo.utils.io_utils.**read_import_descriptors**()

polo.utils.io_utils.**run_name_validator**(*new_run_name*, *current_run_names*)

polo.utils.io_utils.**write_screen_html**(*plate_list*, *well_number*, *run_name*, *x_reagent*, *y_reagent*, *well_volume*, *output_path*)

polo.utils.math_utils module

polo.utils.math_utils.**best_aspect_ratio**(*w*, *h*, *n*)

polo.utils.math_utils.**factors**(*n*)

polo.utils.math_utils.**get_cell_image_dims**(*w*, *h*, *n*)

polo.utils.math_utils.**get_image_cell_size**(*cell_aspect*, *w*, *h*)

Module contents

4.2 Module contents

polo.**make_default_logger**(*name*)

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p