paws Documentation

Release 0.4.0

Lenson A. Pellouchoud

Feb 17, 2017

CONTENTS

Contents:

CHAPTER ONE

INTRODUCTION

The `paws` package aims to provide a fast and lean platform for building and executing workflows for data processing. It was originally developed to perform analysis of diffraction images for research purposes at SLAC/SSRL. At the core of `paws` is a workflow engine that uses a library of operations to crunch through data and expose select results while attempting to minimize resource consumption.

`paws` is currently written in Python, based on Qt via the PySide bindings. Internally, `paws` keeps track of data in Qt-based tree models, which can be controlled either directly (through the paws api) or through a gui (employing the Qt model-view framework).

`paws` also provides an interface to `xi-cam`, a synchrotron x-ray diffraction data analysis package written by the CAMERA Institute and Pandolfi, et al at the Lawrence Berkeley National Lab.

Some the core goals of `paws`:

- Eliminate redundant development efforts
- Streamline and standardize routine data analysis
- Simplify data storage and provide large-scale analysis
- Perform data analysis in real time for results-driven feedback

The `paws` developers would love to hear from you if you have wisdom, thoughts, haikus, bugs, artwork, or suggestions. Limericks are also welcome. Get in touch with us at `paws-developers@slac.stanford.edu`.

CHAPTER

TWO

QUICK START

Minimal and usually-effective installation instructions.

Here is a reference to the *brief introduction*.

This chapter is for setting up `paws` quickly in an environment that is prepared to install Python packages with `pip`.

CHAPTER

THREE

INSTALLATION

Here are instructions for installing `paws` from PyPI, or downloading and testing the `paws` source code.

Installing with pip

Instructions will go here for installing `paws` using the Python package installer `pip` (currently not implemented).

Downloading Source

The source code for `paws` is hosted on github. Clone the repository from `https://github.com/slaclab/paws.git`. You should then be able to run `paws` by invoking `python main.py` from the root directory.

Testing

`paws` comes with a tests that can be used to ensure the platform runs as expected. After *downloading the source*, invoke `python -m unittest discover` from the root directory.

CHAPTER

FOUR

API DOCUMENTATION

This is the complete auto-generated documentation of the `paws` package, made with sphinx-apidoc.

paws package

Subpackages

paws.core package

Subpackages

paws.core.operations package

Subpackages

paws.core.operations.COMM package

Submodules

paws.core.operations.COMM.TestTCP module

Operation for testing a TCP client

class `paws.core.operations.COMM.TestTCP.`**`TestTCP`**

Bases: *`paws.core.operations.operation.Operation`*

Given a filename and a TCP client, this operation sends some stuff to the TCP client.

`run``()`

Module contents

paws.core.operations.DISPLAY package

Submodules

paws.core.operations.DISPLAY.simple_plots module

class `paws.core.operations.DISPLAY.simple_plots.`**`MPLFigFromXYData`**

Bases: *`paws.core.operations.operation.Operation`*

`run`()

class `paws.core.operations.DISPLAY.simple_plots.`**`SimplePlot`**

Bases: *`paws.core.operations.operation.Operation`*

Plot a 1d array against another 1d array.

`run`()

`simple_plot`(*x*, *y*)

paws.core.operations.DISPLAY.zip module

class `paws.core.operations.DISPLAY.zip.`**`LogLogZip`**

Bases: *`paws.core.operations.operation.Operation`*

Takes the logarithm of two 1d ndarrays, then zips them together.

Logarithm is taken in base ten.

Any elements with non-positive values are removed, so this operation may not be appropriate for computational purposes.

`run`()

class `paws.core.operations.DISPLAY.zip.`**`Zip`**

Bases: *`paws.core.operations.operation.Operation`*

Zips two 1d ndarrays together.

`run`()

`paws.core.operations.DISPLAY.zip.`**`zip`**(*x*, *y*)

Zips input 1d vectors together for display.

Should be pre-checked.

`paws.core.operations.DISPLAY.zip.`**`zip_check`**(*x*, *y*)

Checks that inputs are 1d vectors of the same size and shape.

Module contents

paws.core.operations.EXECUTION package

Submodules

paws.core.operations.EXECUTION.batch_execution module

class `paws.core.operations.EXECUTION.batch_execution.`**`BatchFromFiles`**

Bases: *`paws.core.operations.operation.Batch`*

Provides a sequence of inputs to be used in repeated execution of a workflow. Collects the outputs produced for each of the inputs.

batch_ops()
Provide a list of uri's of ops to be included in batch execution

input_list()

input_routes()
Provide the input route in a list- must return list.

output_list()

run()
Build a list of [uri:value] dicts to be used in the workflow.

saved_items()
List uris to be saved/stored after execution

paws.core.operations.EXECUTION.realtime_execution module

class `paws.core.operations.EXECUTION.realtime_execution.`**RealtimeFromFiles**
Bases: *paws.core.operations.operation.Realtime*

Provides inputs to be used in repeated execution of a workflow from files with names matching a regex, as they arrive in a specified directory. Collects the outputs produced for each of the inputs.

static delay()
Amount of time to wait between execution attempts, in milliseconds

input_iter()

input_routes()
Use the Realtime.input_locators to list uri's of all input routes- must return list.

output_list()

realtime_ops()
Use the Realtime.input_locator to list uri's of ops to be saved/stored after execution

run()
This should create an iterator whose next() gives a {uri:value} dict built from the latest-arrived file

saved_items()
Use the Realtime.input_locator to list uri's of ops to be included in realtime execution

Module contents

paws.core.operations.INPUT package

Subpackages

paws.core.operations.INPUT.BL1-5 package

Submodules

paws.core.operations.INPUT.BL1-5.SSRL_1_5_readers module

Module contents

paws.core.operations.INPUT.WXDIFF package

Submodules

paws.core.operations.INPUT.WXDIFF.read_wxdiff_files module

class `paws.core.operations.INPUT.WXDIFF.read_wxdiff_files.`**`WXDCalibToDict`**

Bases: *`paws.core.operations.operation.Operation`*

Input is the path to a WXDiff .calib file output is a dict containing calib parameters

`run``()`

Module contents

Submodules

paws.core.operations.INPUT.fabio_input module

class `paws.core.operations.INPUT.fabio_input.`**`LoadFabIO`**

Bases: *`paws.core.operations.operation.Operation`*

Takes a filesystem path and calls fabIO to load it.

`run``()`

Call on fabIO to extract image data

paws.core.operations.INPUT.pillow_tif_input module

class `paws.core.operations.INPUT.pillow_tif_input.`**`LoadTif`**

Bases: *`paws.core.operations.operation.Operation`*

Takes a filesystem path that points to a .tif, outputs image data and metadata from the file.

`istiff``()`

`run``()`

Call on image rendering libs to extract image data *.tiff or *.tif images: use PIL

`tifftest` **= <_sre.SRE_Pattern object>**

class `paws.core.operations.INPUT.pillow_tif_input.`**`LoadTifToGrayscale`**

Bases: *`paws.core.operations.operation.Operation`*

Takes a filesystem path that points to a .tif, outputs image data and metadata from the file.

`istiff``()`

`run``()`

Call on image rendering libs to extract image data *.tiff or *.tif images: use PIL

`tifftest` **= <_sre.SRE_Pattern object>**

paws.core.operations.INPUT.read_csv module

class `paws.core.operations.INPUT.read_csv.`**`ReadCSV_q_I_dI`**

Bases: *`paws.core.operations.operation.Operation`*

Read q, I, and (if available) dI from a csv-formatted file.

If the csv has no third column, returns None for dI.

`run``()`

`paws.core.operations.INPUT.read_csv.`**`read_csv_q_I_maybe_dI`**`(`*nameloc*`)`

Module contents

paws.core.operations.OUTPUT package

Subpackages

paws.core.operations.OUTPUT.PIF package

Submodules

paws.core.operations.OUTPUT.PIF.citrination_shipment module

class `paws.core.operations.OUTPUT.PIF.citrination_shipment.`**`CheckDataSet`**

Bases: *`paws.core.operations.operation.Operation`*

Take a Citrination client as input and use it to query a data set. Output some indication of whether or not the query was successful.

`run``()`

class `paws.core.operations.OUTPUT.PIF.citrination_shipment.`**`CreateDataSet`**

Bases: *`paws.core.operations.operation.Operation`*

Take a Citrination client as input and use it to create a data set. Output the index of the created data set.

`run``()`

class `paws.core.operations.OUTPUT.PIF.citrination_shipment.`**`ShipJSON`**

Bases: *`paws.core.operations.operation.Operation`*

Take a .json file containing a pif, and ship the pif to a given Citrination data set.

`run``()`

class `paws.core.operations.OUTPUT.PIF.citrination_shipment.`**`ShipToDataSet`**

Bases: *`paws.core.operations.operation.Operation`*

Take a pypif.obj.System object and ship it to a given Citrination data set.

`run``()`

Module contents

Submodules

paws.core.operations.OUTPUT.write_csv module

class `paws.core.operations.OUTPUT.write_csv.`**`WriteArrayCSV`**

Bases: *`paws.core.operations.operation.Operation`*

Write a 2d array to a csv file

`run``()`

class `paws.core.operations.OUTPUT.write_csv.`**`WriteCSV_q_I_dI`**

Bases: *`paws.core.operations.operation.Operation`*

Write q, I, and (if available) dI to a csv-formatted file.

`run``()`

`paws.core.operations.OUTPUT.write_csv.`**`replace_extension`**`(`*old_name*, *new_extension*`)`

Return a file name that is identical except for extension.

Parameters

- **`old_name`** – string path or file name
- **`new_extension`** – string extension, e.g. ".txt"

Returns

Accepts extensions with or without an initial ".".

`paws.core.operations.OUTPUT.write_csv.`**`write_csv_q_I_maybe_dI`**`(`*q*, *I*, *dI*, *nameloc*`)`

Module contents

paws.core.operations.PACKAGING package

Subpackages

paws.core.operations.PACKAGING.PIF package

Submodules

paws.core.operations.PACKAGING.PIF.pif_saxs module

class `paws.core.operations.PACKAGING.PIF.pif_saxs.`**`PifNPSynth`**

Bases: *`paws.core.operations.operation.Operation`*

Package results from nanoparticle solution synthesis into a pypif.obj.ChemicalSystem object.

`run``()`

`saxs_to_pif_properties``(`*q*, *I_q*, *T_C*`)`

paws.core.operations.PACKAGING.PIF.pif_ternary module

class `paws.core.operations.PACKAGING.PIF.pif_ternary.`**`PifTernary`**

Bases: *`paws.core.operations.operation.Operation`*

Package results from ternary wafer synthesis into a pypif.obj.ChemicalSystem object.

`intensity_features_to_pif_properties`(*I_feats*)

`pks_to_pif_properties`(*q_pk*, *I_pk*)

`run`()

`texture_to_pif_properties`(*q*, *tex_q*)

`xrd_to_pif_properties`(*q*, *I_q*)

Module contents

Submodules

paws.core.operations.PACKAGING.data_packing module

class `paws.core.operations.PACKAGING.data_packing.`**`TimeTempFromHeader`**

Bases: *`paws.core.operations.operation.Operation`*

Get time and temperature from a detector output header file. Return string time, float time (utc in seconds), and float temperature. Time is assumed to be in the format Day Mon dd hh:mm:ss yyyy.

`run`()

class `paws.core.operations.PACKAGING.data_packing.`**`WindowZip`**

Bases: *`paws.core.operations.operation.Operation`*

From input iterables of x and y, produce an n-by-2 array where x is bounded by the specified limits

`run`()

class `paws.core.operations.PACKAGING.data_packing.`**`Window_q_I_2`**

Bases: *`paws.core.operations.operation.Operation`*

From input iterables of *q_list_in* and *I_list_in*, produce two 1d vectors where q is greater than *q_min* and less than *q_min*

`run`()

class `paws.core.operations.PACKAGING.data_packing.`**`XYDataFromBatch`**

Bases: *`paws.core.operations.operation.Operation`*

Given a batch output and appropriate keys, use the uris to harvest x and y data from the batch.

`run`()

Module contents

paws.core.operations.PROCESSING package

Subpackages

paws.core.operations.PROCESSING.SAXS package

Submodules

paws.core.operations.PROCESSING.SAXS.background_subtraction module

class `paws.core.operations.PROCESSING.SAXS.background_subtraction.`**`BgSubtractByTemperature`**

Bases: *`paws.core.operations.operation.Operation`*

Find a background spectrum from a batch of background spectra, where the temperature of the background spectrum is as close as possible to the (input) temperature of the measured spectrum. Then subtract that background spectrum from the input spectrum. The measured and background spectra are expected to have the same domain.

`run``()`

class `paws.core.operations.PROCESSING.SAXS.background_subtraction.`**`SubtractMaximumBackground`**

Bases: *`paws.core.operations.operation.Operation`*

Subtract a background from a foreground, with scaling to prevent over-subtraction.

Has optional arguments for error vectors. If no error estimate is available, set these to *None*. If either error vector is *None*, the output error vector will also be *None*.

Can also, optionally, window to a region in q.

`run``()`

`paws.core.operations.PROCESSING.SAXS.background_subtraction.`**`subtract_maximum_background`**`(`*q_f,*
q_b
I_f,
I_b
*dI_*
*dI_*
q1=
q2=

paws.core.operations.PROCESSING.SAXS.saxs module

class `paws.core.operations.PROCESSING.SAXS.saxs.`**`FetchReferences`**

Bases: *`paws.core.operations.operation.Operation`*

Fetch previously generated and stored metrics used for guessing diffraction pattern properties.

This function is for examination, and is not necessary for computation. The file in question is auto-fetched by operations that rely on it.

`run``()`

class `paws.core.operations.PROCESSING.SAXS.saxs.`**`GenerateReferences`**

Bases: *`paws.core.operations.operation.Operation`*

Generate metrics used for guessing diffraction pattern properties.

DOES NOT STORE. IS NOT AUTOMATICALLY AVAILABLE TO OTHER OPERATIONS.

Use OverwriteReferences to store the resulting dictionary, if desired.

`run``()`

class `paws.core.operations.PROCESSING.SAXS.saxs.`**`GenerateSphericalDiffraction`**

Bases: *`paws.core.operations.operation.Operation`*

Generate a SAXS diffraction pattern for spherical nanoparticles.

Uses r0 in real units and, if asked to generate q, gives q in real units.

`run``()`

class `paws.core.operations.PROCESSING.SAXS.saxs.`**`GuessProperties`**

Bases: *`paws.core.operations.operation.Operation`*

Guess the polydispersity, mean size, and amplitude of spherical diffraction pattern.

Assumes the data have already been background subtracted, smoothed, and otherwise cleaned.

`run``()`

class `paws.core.operations.PROCESSING.SAXS.saxs.`**`OptimizeSphericalDiffractionFit`**

Bases: *`paws.core.operations.operation.Operation`*

From an initial guess, optimize r0, I0, and fractional_variation.

`run``()`

class `paws.core.operations.PROCESSING.SAXS.saxs.`**`OverwriteReferences`**

Bases: *`paws.core.operations.operation.Operation`*

Store previously generated metrics used for guessing diffraction pattern properties.

USE WITH CAUTION as it will OVERWRITE any existing reference file.

`run``()`

`paws.core.operations.PROCESSING.SAXS.saxs.`**`arbitrary_order_solution`**`(`*order*, *x*, *y*, *dy=None*`)`

Solves for a polynomial “fit” of arbitrary order.

`paws.core.operations.PROCESSING.SAXS.saxs.`**`blur`**`(`*x*, *factor*`)`

`paws.core.operations.PROCESSING.SAXS.saxs.`**`chi_squared`**`(`*y1*, *y2*`)`

`paws.core.operations.PROCESSING.SAXS.saxs.`**`choose_dips_and_shoulders`**`(`*q*, *I*, *dI=None*`)`

Find the location of dips (low points) and shoulders (high points).

`paws.core.operations.PROCESSING.SAXS.saxs.`**`clean_extrema`**`(`*dips*, *shoulders*, *y*`)`

`paws.core.operations.PROCESSING.SAXS.saxs.`**`downwards_weaksauce_identifier`**`(`*four_indices*, *y*`)`

`paws.core.operations.PROCESSING.SAXS.saxs.`**`dummy`**`(`*array1d*`)`

Turn 1d array into dummy-index vector for 2d matrix computation.

Sum over the dummy index by taking *object.sum(axis=0)*.

`paws.core.operations.PROCESSING.SAXS.saxs.`**`dump_references`**`(`*references*`)`

`paws.core.operations.PROCESSING.SAXS.saxs.`**`factorStretched`**`(`*x*, *factor*`)`

`paws.core.operations.PROCESSING.SAXS.saxs.`**`first_dip`**`(`*q*, *I*, *dips*, *dI=None*`)`

`paws.core.operations.PROCESSING.SAXS.saxs.`**`fullFunction`**`(`*x*`)`

`paws.core.operations.PROCESSING.SAXS.saxs.`**`gauss`**`(`*x*, *x0*, *sigma*`)`

`paws.core.operations.PROCESSING.SAXS.saxs.`**`gauss_guess`**`(`*signalMagnitude*, *signalCurvature*`)`

Guesses a gaussian intensity and width from signal magnitude and curvature.

Parameters

- **signalMagnitude** – number-like with units of magnitude
- **signalCurvature** – number-like with units of magnitude per distance squared

Return intensity, sigma

The solution given is not fitted; it is a first estimate to be used in fitting.

`paws.core.operations.PROCESSING.SAXS.saxs.`**`gen_q_vector`**(*qmin*, *qmax*, *qstep*)

An inclusive-range version of np.arange.

`paws.core.operations.PROCESSING.SAXS.saxs.`**`generateRhoFactor`**(*factor*)

Generate a guassian distribution of number densities (rho).

Parameters **factor** – float

Returns

factor should be 0.1 for a sigma 10% of size

`paws.core.operations.PROCESSING.SAXS.saxs.`**`generate_references`**(*x*, *factorVals*)

`paws.core.operations.PROCESSING.SAXS.saxs.`**`generate_spherical_diffraction`**(*q*, *i0*, *r0*, *poly*)

`paws.core.operations.PROCESSING.SAXS.saxs.`**`guess_nearest_point_on_nonmonotonic_trace_normal`**

Finds the nearest point to location *loclist* on a trace *tracelist*.

Uses normalized distances, i.e., the significance of each dimension is made equivalent.

loclist and *tracelist* are lists of the same length and type. Elements of *loclist* are floats; elements of *tracelist* are arrays of the same shape as *coordinate*. *coordinate* is the independent variable along which *tracelist* is varying.

tracelist must have decent sampling to start with or the algorithm will likely fail.

`paws.core.operations.PROCESSING.SAXS.saxs.`**`guess_noise_floor`**(*q*, *I*, *r0*)

`paws.core.operations.PROCESSING.SAXS.saxs.`**`guess_polydispersity`**(*q*, *I*, *dI=None*)

`paws.core.operations.PROCESSING.SAXS.saxs.`**`guess_size`**(*fractional_variation*, *first_dip_q*)

`paws.core.operations.PROCESSING.SAXS.saxs.`**`horizontal`**(*array1d*)

Turn 1d array into 2d horizontal vector.

`paws.core.operations.PROCESSING.SAXS.saxs.`**`load_references`**()

`paws.core.operations.PROCESSING.SAXS.saxs.`**`local_maxima_detector`**(*y*)

Finds local maxima in ordered data y.

Parameters **y** – 1d numpy float array

Return maxima 1d numpy bool array

maxima is *True* at a local maximum, *False* otherwise.

This function makes no attempt to reject spurious maxima of various sorts. That task is left to other functions.

`paws.core.operations.PROCESSING.SAXS.saxs.`**`local_minima_detector`**(*y*)

Finds local minima in ordered data *y*.

Parameters **`y`** – 1d numpy float array

Return minima 1d numpy bool array

minima is *True* at a local minimum, *False* otherwise.

This function makes no attempt to reject spurious minima of various sorts. That task is left to other functions.

`paws.core.operations.PROCESSING.SAXS.saxs.`**`make_poly_matrices`**(*x*, *y*, *error*, *order*)

Make the matrices necessary to solve a polynomial fit of order *order*.

Parameters

- **`x`** – 1d array representing independent variable
- **`y`** – 1d array representing dependent variable
- **`error`** – 1d array representing uncertainty in *y*
- **`order`** – integer order of polynomial fit

Return matrix, vector MC=V, where M is *matrix*, V is *vector*, and C is the polynomial coefficients to be solved for. *matrix* is an array of shape (order+1, order+1). *vector* is an array of shape (order+1, 1).

`paws.core.operations.PROCESSING.SAXS.saxs.`**`polydispersity_metric_heightAtZero`**(*qFirstDip*, *q*, *I*, *dI=None*)

`paws.core.operations.PROCESSING.SAXS.saxs.`**`polydispersity_metric_sigmaScaledFirstDip`**(*q*, *I*, *dips*, *shoulders*, *qFirstDip*, *heightFirstDip*, *dI=None*

`paws.core.operations.PROCESSING.SAXS.saxs.`**`polynomial_value`**(*coefficients*, *x*)

Finds the value of a polynomial at a location.

`paws.core.operations.PROCESSING.SAXS.saxs.`**`power_law_solution`**(*x*, *y*, *dy=None*)

Solves for a power law by solving for a linear fit in log-log space.

`paws.core.operations.PROCESSING.SAXS.saxs.`**`prep_for_pickle`**(*factorVals*, *xFirstDip*, *sigmaScaledFirstDip*, *heightFirstDip*, *heightAtZero*)

`paws.core.operations.PROCESSING.SAXS.saxs.`**`quadratic_extremum`**(*coefficients*)

Finds the location in independent coordinate of the extremum of a quadratic equation.

`paws.core.operations.PROCESSING.SAXS.saxs.`**`refine_guess`**(*q*, *I*, *I0*, *r0*, *frac*, *q1*, *I1*)

`paws.core.operations.PROCESSING.SAXS.saxs.`**`take_polydispersity_metrics`**(*x*, *y*, *dy=None*) paws.core.operations.PROCESSING.SAXS.saxs.**upwards_weaksauce_identifier**(*four_indices*, *y*)

paws.core.operations.PROCESSING.SAXS.saxs.**vertical**(*array1d*)
Turn 1d array into 2d vertical vector.

paws.core.operations.PROCESSING.SAXS.saxs_features module

class paws.core.operations.PROCESSING.SAXS.saxs_features.**NanoparticleFeatures**
Bases: *paws.core.operations.operation.Operation*

Use a saxs spectrum (I(q) vs. q) to extract key features of a nanoparticle colloid: average size, standard deviation of sizes, and nanoparticle density. This module assumes spherical nanoparticles.

static monodisperse_saxs(*q*, *r*)
Generate spherical nanoparticle saxs spectrum for monodisperse particle radius r at scattering vectors q.

static polydisperse_saxs(*q*, *r0*, *p*)
Generate spherical nanoparticle saxs spectrum for mean particle radius r0, size polydispersity (sigma(r)/r0) p, and scattering vectors q.

run()

Module contents

Submodules

paws.core.operations.PROCESSING.HiTp_feature_extraction module

Created on Mon Jun 06 18:02:32 2016

author: Fang Ren, Apurva Mehta For details, refer to the recent paper submitted to ACS Combinatorial Science Fang Ren implemented all the methods on scripting level, and originally contributed to slacx

Contributor: fangren (please add if there is more)

class paws.core.operations.PROCESSING.HiTp_feature_extraction.**FindPeaksByWindow**
Bases: *paws.core.operations.operation.Operation*

Walk a 1d array and find its local maxima. A maximum is found if it is the highest point within windowsize of itself. An optional threshold for the peak intensity relative to the window-average can be used to filter out peaks due to noise.

run()

class paws.core.operations.PROCESSING.HiTp_feature_extraction.**IntensityFeatures**
Bases: *paws.core.operations.operation.Operation*

Extract the maximum intensity, average intensity, and a ratio of the two from data

run()

class paws.core.operations.PROCESSING.HiTp_feature_extraction.**PeakFeatures**
Bases: *paws.core.operations.operation.Operation*

Extract the locations and intensities of peaks from a 1D spectrum

get_extrema(*x*, *y*, *delta*)
Given vectors x and y, return an n-by-2 array of x,y pairs for the minima and maxima of y(x)

run ()

class paws.core.operations.PROCESSING.HiTp_feature_extraction.**TextureFeatures**

Bases: *paws.core.operations.operation.Operation*

Analyze the texture

run ()

paws.core.operations.PROCESSING.VoigtPeakFit module

class paws.core.operations.PROCESSING.VoigtPeakFit.**VoigtPeakFit**

Bases: *paws.core.operations.operation.Operation*

Fit a set of x and y values to a Voigt distribution. Solves the distribution over the space of hwhm (half width at half max) of the gaussian and lorentzian distributions and distribution center. Takes as input a guess for the distribution center and hwhm. Range of fit is determined by weighting the objective by a Hann window centered at the distribution center, with a window width of the distribution's estimated full width at half max.

static gaussian (*x*, *hwhm_g*)

gaussian distribution at points x, center 0, hwhm hwhm_g

static hann_voigt_fit (*x*, *y*, *xc*, *hwhm_g*, *hwhm_l*, *scl*)

static lorentzian (*x*, *hwhm_l*)

lorentzian distribution at points x, center 0, hwhm hwhm_l

run ()

static solve_voigt (*x*, *y*, *xc*, *hwhm_g*, *hwhm_l*, *scl*)

iteratively minimize an objective to fit x, y curve to a voigt profile

static voigt (*x*, *hwhm_g*, *hwhm_l*)

voigt distribution resulting from convolution of a gaussian with hwhm hwhm_g and a lorentzian with hwhm hwhm_l

paws.core.operations.PROCESSING.arithmetic module

class paws.core.operations.PROCESSING.arithmetic.**Add**

Bases: *paws.core.operations.operation.Operation*

Add two objects.

run ()

class paws.core.operations.PROCESSING.arithmetic.**Divide**

Bases: *paws.core.operations.operation.Operation*

Divide two objects.

run ()

class paws.core.operations.PROCESSING.arithmetic.**Exponentiate**

Bases: *paws.core.operations.operation.Operation*

Exponentiate an object by another object.

run ()

class paws.core.operations.PROCESSING.arithmetic.**Logarithm**

Bases: *paws.core.operations.operation.Operation*

Take the logarithm of an object by some base.

run()

class paws.core.operations.PROCESSING.arithmetic.**Multiply**

Bases: *paws.core.operations.operation.Operation*

Multiply two objects.

run()

class paws.core.operations.PROCESSING.arithmetic.**Subtract**

Bases: *paws.core.operations.operation.Operation*

Subtract one object from another.

run()

paws.core.operations.PROCESSING.calib_and_reduce module

Operations for remeshing and and reducing an image

author: Fang Ren, Apurva Mehta, Ron Pandolfi For details, please refer to the recent paper submitted to ACS Combinatorial Science F. Ren implemented the remeshing on scripting level, contributed by R. Pandolfi from LBL. F. Ren originally contributed it to slacx

Please add yours too, Amanda, Lenson.

contributors: fangren, apf, lensonp

class paws.core.operations.PROCESSING.calib_and_reduce.**CalByWXDDict**

Bases: *paws.core.operations.operation.Operation*

Input image data (ndarray) and a dict of calibration parameters from a WxDiff .calib file Return q, chi, I(q,chi)

run()

class paws.core.operations.PROCESSING.calib_and_reduce.**ReduceByWXDDict**

Bases: *paws.core.operations.operation.Operation*

Input image data (ndarray) and a dict of calibration parameters from a WxDiff .calib file Return q, I(q)

run()

class paws.core.operations.PROCESSING.calib_and_reduce.**ReduceByWXDDict_mask_error**

Bases: *paws.core.operations.operation.Operation*

Input image data (ndarray) and a dict of calibration parameters from a WxDiff .calib file Return q, I(q)

run()

paws.core.operations.PROCESSING.mirror module

class paws.core.operations.PROCESSING.mirror.**ArrayMirrorHorizontal**

Bases: *paws.core.operations.operation.Operation*

Mirror an array across a horizontal plane, i.e., exchange indices along axis 0.

run()

class `paws.core.operations.PROCESSING.mirror.`**`ArrayMirrorVertical`**

Bases: *`paws.core.operations.operation.Operation`*

Mirror an array across a vertical plane, i.e., exchange indices along axis 1.

`run``()`

class `paws.core.operations.PROCESSING.mirror.`**`MirrorHorizontally`**

Bases: *`paws.core.operations.operation.Operation`*

Mirror an image, exchanging left and right.

I.e., mirror an ndarray along axis = 0.

class `paws.core.operations.PROCESSING.mirror.`**`MirrorVertically`**

Bases: *`paws.core.operations.operation.Operation`*

Mirror an image, exchanging top and bottom.

I.e., mirror an ndarray along axis = 1.

`run``()`

paws.core.operations.PROCESSING.rotation module

class `paws.core.operations.PROCESSING.rotation.`**`Rotation`**

Bases: *`paws.core.operations.operation.Operation`*

Rotate an array by 90, 180, or 270 degrees.

`run``()`

Rotate self.inputs['image_data'] and save as self.outputs['image_data']

paws.core.operations.PROCESSING.smoothing module

class `paws.core.operations.PROCESSING.smoothing.`**`RectangularSmooth`**

Bases: *`paws.core.operations.operation.Operation`*

Applies rectangular (moving average) smoothing filter to 1d data.

User-specified error estimate used to weight points. Set *dy* to *None* if unavailable.

`run``()`

class `paws.core.operations.PROCESSING.smoothing.`**`SavitzkyGolay`**

Bases: *`paws.core.operations.operation.Operation`*

Applies a Savitzky-Golay (polynomial fit approximation) filter to 1d data.

Uses error bars on intensity if available. Set *dy* to *None* otherwise.

`run``()`

class `paws.core.operations.PROCESSING.smoothing.`**`TriangularSmooth`**

Bases: *`paws.core.operations.operation.Operation`*

Applies triangular-weighted (moving average) smoothing filter to 1d data.

User-specified error estimate used to weight points. Set *dy* to *None* if unavailable.

`run``()`

`paws.core.operations.PROCESSING.smoothing.`**`choose_m`**(*order*, *base*)

Choose a number of points to use for SG fit.

Parameters

- **`order`** – integer order of polynomial fit
- **`base`** – an edge condition specification parameter

Return m integer number of data points to use

Helper function for *savitzky_golay*.

`paws.core.operations.PROCESSING.smoothing.`**`choose_start_and_end`**(*m*, *base*, *ii*, *size*)

`paws.core.operations.PROCESSING.smoothing.`**`dummy`**(*array1d*)

Turn 1d array into dummy-index vector for 2d matrix computation.

Sum over the dummy index by taking *object.sum(axis=0)*.

`paws.core.operations.PROCESSING.smoothing.`**`horizontal`**(*array1d*)

Turn 1d array into 2d horizontal vector.

`paws.core.operations.PROCESSING.smoothing.`**`make_poly_matrices`**(*x*, *y*, *error*, *order*)

Make the matrices necessary to solve a polynomial fit of order *order*.

Parameters

- **`x`** – 1d array representing independent variable
- **`y`** – 1d array representing dependent variable
- **`error`** – 1d array representing uncertainty in *y*
- **`order`** – integer order of polynomial fit

Return matrix, vector MC=V, where M is *matrix*, V is *vector*, and C is the polynomial coefficients to be solved for. *matrix* is an array of shape (order+1, order+1). *vector* is an array of shape (order+1, 1).

`paws.core.operations.PROCESSING.smoothing.`**`moving_average`**(*data*, *m*, *shape*, *error=None*)

`paws.core.operations.PROCESSING.smoothing.`**`no_specified_error_weights`**(*data*)

`paws.core.operations.PROCESSING.smoothing.`**`polynomial`**(*x*, *coefficients*)

Evaluate a polynomial with given coefficients at location x.

Parameters

- **`x`** – numeric value of coordinate
- **`coefficients`** – 1d ndarray of one or more elements; zeroth element is zero-order coefficient, 1st is 1st-order coefficient, etc.

Return value value of the specified polynomial at *x*

In *coefficients*, zeroth element is zero-order coefficient (i.e. constant offset), 1st is 1st-order coefficient (i.e. linear slope), etc.

`paws.core.operations.PROCESSING.smoothing.`**`savitzky_golay`**(*x*, *y*, *order*, *base*, *dy=None*)

`paws.core.operations.PROCESSING.smoothing.`**`specified_error_weights`**(*error*)

`paws.core.operations.PROCESSING.smoothing.`**`square_weighting`**(*n*)

`paws.core.operations.PROCESSING.smoothing.`**`triangular_weighting`**(*n*) `paws.core.operations.PROCESSING.smoothing.`**`vertical`**`(`*array1d*`)`

Turn 1d array into 2d vertical vector.

Module contents

paws.core.operations.TESTS package

Submodules

paws.core.operations.TESTS.identity module

class `paws.core.operations.TESTS.identity.`**`Identity`**

Bases: *`paws.core.operations.operation.Operation`*

An Operation testing class, loads its input into its output

`run``()`

Module contents

Submodules

paws.core.operations.op_manager module

class `paws.core.operations.op_manager.`**`OpManager`**`(`***kwargs*`)`

Bases: *`paws.core.treemodel.TreeSelectionModel`*

Tree structure for categorized storage and retrieval of Operations.

`add_cat``(`*new_cat, parent*`)`

Add a category to the tree under parent if not already there. Return its index.

`add_op``(`*op, parent*`)`

add op to the tree under QModelIndex parent

`data``(`*item_indx, data_role*`)`

`get_op``(`*indx*`)`

`get_op_byname``(`*op_name*`)`

`headerData``(`*section, orientation, data_role*`)`

`idx_of_cat``(`*catname, parent*`)`

If cat exists under parent, return its index, else return an invalid QModelIndex

`list_op_names``()`

`load_cats``(`*cat_list*`)`

`load_ops``(`*cat_op_list*`)`

Load OpManager tree from input cat_op_list. Format of cat_op_list is [(category1,op1),(category2,op2),...]. i.e. each operation in cat_op_list is specified by a tuple, where the first element is a category, and the second element is the Operation itself. load_cats() MUST be called before load_ops() and MUST ensure that all cats in cat_op_list exist in the tree.

`remove_op``(`*removal_indx*`)`

save_config()

staticMetaObject = **<PySide.QtCore.QMetaObject object>**

paws.core.operations.operation module

class paws.core.operations.operation.**Batch**(*input_names*, *output_names*)
Bases: *paws.core.operations.operation.Operation*

batch_ops()
Return a list of operation uris to be included in the Batch execution stack.

input_list()
Produce a list of OrderedDicts representing each set of inputs for the Batch to run. Each OrderedDict should be populated with [input_uri:input_value] pairs.

input_routes()
Produce a list of the input routes used by the Batch, in the same order as each of the OrderedDicts provided by Batch.input_list()

output_list()
Produce a list of OrderedDicts representing the outputs for each batch input. Each OrderedDict should be populated with [input_uri:input_value] pairs.

saved_items()
Return a list of items to be saved after each execution.

class paws.core.operations.operation.**Operation**(*input_names*, *output_names*)
Bases: object

description()
self.description() returns a string documenting the input and output structure and usage instructions for the Operation

doc_as_string()

input_description()

load_defaults()

output_description()

run()
Operation.run() should use all of the items in Operation.inputs and set values for all of the items in Operation.outputs.

class paws.core.operations.operation.**Realtime**(*input_names*, *output_names*)
Bases: *paws.core.operations.operation.Operation*

delay()
Return the number of MILLIseconds to pause between iterations. Overload this method to change the pause time- default is 1 second.

input_iter()
Produce an iterator over OrderedDicts representing each set of inputs to run. Each dict should be populated with [input_uri:input_value] pairs. When there is no new set of inputs to run, should return None.

input_routes()
Produce a list of [input_uri] routes in the same order as the OrderedDicts produced by Realtime.input_iter()

output_list()

Produce a list of OrderedDicts representing the outputs for each realtime input. Each OrderedDict should be populated with [input_uri:input_value] pairs.

realtime_ops()

Return a list of operation uris to be included in the Realtime execution stack.

saved_items()

Return a list of item uris to be saved after each execution.

paws.core.operations.optools module

Operations config and processing routines

class paws.core.operations.optools.**InputLocator**(*src=0*, *tp=0*, *val=None*)

Bases: object

Objects of this class are used as containers for inputs to an Operation, and should by design contain the information needed to find the relevant input data. After the data is loaded, it should be stored in InputLocator.data.

paws.core.operations.optools.**cast_type_val**(*tp*, *val*)

Perform type casting for operation inputs. This should be called only for source = text_input.

paws.core.operations.optools.**dict_contains_uri**(*uri*, *d*)

paws.core.operations.optools.**get_uri_from_dict**(*uri*, *d*)

paws.core.operations.optools.**op_dict**(*op*)

paws.core.operations.optools.**op_inputs_dict**(*op*)

paws.core.operations.optools.**parameter_doc**(*name*, *value*, *doc*)

paws.core.operations.optools.**plugin_dict**(*pgin*)

paws.core.operations.optools.**print_stack**(*stk*)

paws.core.operations.optools.**stack_contains**(*itm*, *stk*)

paws.core.operations.optools.**stack_size**(*stk*)

Module contents

paws.core.operations.**load_cfg**(*cfg_file*)

paws.core.operations.**load_ops_from_module**(*mod*, *cat_root*)

paws.core.operations.**load_ops_from_path**(*path_*, *pkg*, *cat_root='MISC'*)

paws.core.operations.**save_cfg**(*cfg_data*, *cfg_file*)

paws.core.plugins package

Submodules

paws.core.plugins.CitrinationPlugin module

class paws.core.plugins.CitrinationPlugin.**CitrinationPlugin**

Bases: *paws.core.plugins.plugin.PawsPlugin*

Wrapper contains a Citrination client and implements the PawsPlugin abc interface.

content()

description()

ship_dataset(*pifs*)

start()

stop()

paws.core.plugins.SpecClientPlugin module

class paws.core.plugins.SpecClientPlugin.**SpecClientPlugin**

Bases: *paws.core.plugins.plugin.PawsPlugin*

content()

description()

receiveLine()

sendCmd(*cmd*)

sendLine(*line*)

send_commands(*cmd_list*)

send_text(*txt*)

start()

stop()

paws.core.plugins.TCPClientPlugin module

class paws.core.plugins.TCPClientPlugin.**TCPClientFactory**(*protocol*)

Bases: twisted.internet.protocol.ClientFactory

buildProtocol(*addr*)

clientConnectionFailed(*connector*, *reason*)

Clients call this when they are unable to initialize their connection.

clientConnectionLost(*connector*, *reason*)

Clients call this when their connections are lost.

class paws.core.plugins.TCPClientPlugin.**TCPClientPlugin**

Bases: *paws.core.plugins.plugin.PawsPlugin*

content()

description()

send_text(*txt*)

start()

stop()

class paws.core.plugins.TCPClientPlugin.**TCPTestProtocol**

Bases: twisted.protocols.basic.LineReceiver

addCommand(*cmd*)

connectionLost()

connectionMade()

lineReceived(*line*)

send_lines()

paws.core.plugins.plugin module

class paws.core.plugins.plugin.**PawsPlugin**(*input_names*)

Bases: object

content()

PawsPlugin.content() returns a dict containing the meaningful objects contained in the plugin. The default implementation returns the plugin itself, keyed by 'plugin'.

description()

PawsPlugin.description() returns a string documenting the functionality of the PawsPlugin. The default implementation returns no description.

start()

PawsPlugin.start() should perform any setup required by the plugin, for instance setting up connections and reading files used by the plugin. The default implementation does nothing.

stop()

PawsPlugin.stop() should provide a clean end for the plugin, for instance closing all connections and files used by the plugin. The default implementation does nothing, assumes the plugin can be cleanly terminated by dereferencing.

paws.core.plugins.plugin_manager module

class paws.core.plugins.plugin_manager.**PluginManager**(***kwargs*)

Bases: *paws.core.treemodel.TreeSelectionModel*

Tree structure for managing paws plugins.

add_plugin(*pgin_tag*, *pgin*)

Add a Plugin to the tree as a new top-level TreeItem.

build_dict(*x*)

Overloaded build_dict to handle Plugins

get_plugin_byname(*pgin_name*)

headerData(*section*, *orientation*, *data_role*)

load_from_dict(*pgin_dict*)

Load plugins from a dict that specifies their setup parameters.

remove_plugin(*rm_idx*)

Remove a Plugin from the tree

staticMetaObject = <PySide.QtCore.QMetaObject object>

write_log(*msg*)

Module contents

paws.core.plugins.**load_plugins**(*path_*, *pkg*)

paws.core.workflow package

Submodules

paws.core.workflow.wf_manager module

class paws.core.workflow.wf_manager.**WfManager**(*qapp_reference*, *plugin_manager*, ***kwargs*)

Bases: *paws.core.treemodel.TreeSelectionModel*

Tree structure for managing a Workflow built from paws Operations.

add_op(*uri*, *new_op*)

Add an Operation to the tree as a new top-level TreeItem.

batch_op_stack(*b_itm*, *valid_wf_inputs*)

Use b_itm.data.batch_ops() and a list of valid_wf_inputs (a list of uris that are good inputs from the workflow) to build a stack (list) of lists of operations such that all Operations in the stack have their dependencies satisfied by the Operations above them.

build_dict(*x*)

Overloaded build_dict to handle Operations. Base class method builds dicts from other data types.

check_wf()

Check the dependencies of the workflow. Ensure that all loaded operations have inputs that make sense. Return a status code and message for each of the Operations.

columnCount(*parent*)

Let WfManager have two columns, one for item tag, one for item type

execution_stack()

Build a stack (list) of lists of TreeItems, such that each TreeItem list contains a set of Operations whose dependencies are satisfied by the operations above them. For Batch or Realtime operations, the layer should be of the form[batch_item,batch_stack], where batch_item.data is the batch controller Operation, and batch_stack is built from self.batch_op_stack().

finish_thread(*th_idx*)

get_valid_wf_inputs(*itm*)

Return all of the TreeModel uris of itm and its children which can be used as downstream inputs in the workflow.

headerData(*section*, *orientation*, *data_role*)

inputs_tag = 'inputs'

is_op_ready(*itm*, *valid_wf_inputs*, *batch_routes=[]*)

is_running()

load_from_dict(*opman*, *opdict*)

Load things in to the Workflow from an OpManager using a dict that specifies operation setup.

load_inputs(*op*)

Loads input data for an Operation from that Operation's input_locators. It is expected that op.input_locator[name] will refer to an InputLocator.

`locate_input` (*il*)
  Return the data pointed to by a given InputLocator object.

`loopwait` (*interval*)
  Create an event loop to delay some time without busywaiting. Time interval is specified in milliseconds.

`next_available_thread` ()

`outputs_tag` = **'outputs'**

`remove_op` (*rm_idx*)
  Remove an Operation from the workflow tree

`run_wf` ()

`run_wf_batch` (*b_itm*, *stk*)
  Executes the items in the stack stk under the control of one Batch controller Operation

`run_wf_realtime` (*rt_itm*, *stk*)
  Executes the workflow under the control of one Realtime controller Operation, where the realtime controller Operation is found at rt_itm.data.

`run_wf_serial` (*stk*, *thd_idx=None*)
  Serially execute the operations contained in the stack stk.

`set_op_input_at_uri` (*uri*, *val*)
  Set an op input, indicated by uri, to provided value. uri must be of the form op_name.inputs.input_name. Currently shallower uris (e.g. op_name.inputs) and deeper uris (e.g. op_name.inputs.input_list.0) are not supported.

`staticMetaObject` = **<PySide.QtCore.QMetaObject object>**

`stop_wf` ()

`updateOperation` (*tag*, *op*)

`update_io_deps` ()
  Remove any broken dependencies in the workflow. Should only be called after all current data have been stored in the tree.

`update_op` (*uri*, *new_op*)
  Update Operation in treeitem indicated by uri. It is expected that new_op is a reference to the Operation stored at uri.

static `update_uri_dict` (*d*, *d_new*)

`uri_to_dict` (*uri*, *data*)

`wait_for_thread` (*th_idx*)
  Wait for the thread at self._wf_threads[th_idx] to be finished

`wait_for_threads` ()
  Wait for all workflow execution threads to finish

`wfdone` = **<PySide.QtCore.Signal object>**

`write_log` (*msg*)

paws.core.workflow.wf_worker module

class `paws.core.workflow.wf_worker.`**`WfWorker`** (*to_run=None*, *parent=None*)
  Bases: `PySide.QtCore.QObject`

Container for storing and executing parts of a workflow, to be pushed onto QtCore.QThread(s) as needed.

finished = <PySide.QtCore.Signal object>

opDone = <PySide.QtCore.Signal object>

staticMetaObject = <PySide.QtCore.QMetaObject object>

work ()

Module contents

Submodules

paws.core.listmodel module

class `paws.core.listmodel.`**ListModel** (*input_list=[]*, *parent=None*)
Bases: `PySide.QtCore.QAbstractListModel`

Class for list management with a QAbstractListModel. Implements required virtual methods rowCount() and data(). Resizeable ListModels must implement insertRows(), removeRows(). If a nicely labeled header is desired, implement headerData().

append_item (*thing*)

columnCount (*parent=<PySide.QtCore.QModelIndex(-1*, *-1*, *0x0*, *QObject(0x0)) >*)

data (*idx*, *data_role*)

flags (*idx*)

get_item (*idx*)

headerData (*section*, *orientation*, *data_role*)

insertRows (*row*, *count*)

list_data ()

removeRows (*row*, *count*, *parent=<PySide.QtCore.QModelIndex(-1*, *-1*, *0x0*, *QObject(0x0)) >*)

remove_item (*row*)

rowCount (*parent=<PySide.QtCore.QModelIndex(-1*, *-1*, *0x0*, *QObject(0x0)) >*)

set_disabled (*row*)

set_enabled (*row*)

staticMetaObject = <PySide.QtCore.QMetaObject object>

paws.core.pawstools module

class `paws.core.pawstools.`**FileSystemIterator** (*dirpath*, *regex*)
Bases: `_abcoll.Iterator`

next ()

exception `paws.core.pawstools.`**LazyCodeError** (*msg*)
Bases: `exceptions.Exception`

`paws.core.pawstools.`**`dtstr`**`()`
Return date and time as a string

`paws.core.pawstools.`**`timestr`**`()`
Return time as a string

paws.core.treeitem module

class `paws.core.treeitem.`**`TreeItem`**`(`*row*, *column*, *parent*`)`
Bases: `object`

Container for packing objects into a TreeModel.

This is a container to facilitate data storage in a TreeModel. A TreeItem keeps references to a parent QModelIndex, and to its row and column within the QAbstractItemModel structure. The objective content of the TreeItem is stored at TreeItem.data. Every TreeItem must have a tag() for display and uri creation.

`children_checked``()`

`insert_child``(`*new_child*, *row*`)`

`is_checked``()`

`n_children``()`

`remove_child``(`*row*`)`

`set_checked``(`*val*`)`

`set_tag``(`*tag_in*`)`

`tag``()`

paws.core.treemodel module

class `paws.core.treemodel.`**`TreeModel`**
Bases: `PySide.QtCore.QAbstractItemModel`

Class for tree management with a QAbstractItemModel. Implements required virtual methods index(), parent(), rowCount(). Other required virtual methods are columnCount() and data(): these should be implemented by subclassing of TreeModel. Resizeable TreeModels must implement: insertRows(), removeRows(), insertColumns(), removeColumns(). If nicely labeled headers are desired, one should implement headerData().

`auto_tag``(`*prefix*`)`
Generate the next unique tag from prefix by appending '_x' to it, where x is a minimal nonnegative integer.

`build_dict``(`*x*`)`
Build a dict from structured data object x

`build_uri``(`*idx*`)`
Build a URI for the TreeItem at idx by prepending its parent tags with '.' as a delimiter.

`columnCount``(`*parent=<PySide.QtCore.QModelIndex(-1, -1, 0x0, QObject(0x0)) >*`)`
Let TreeModels by default have one column, to display the local TreeItem's tag.

`data``(`*itm_idx*, *data_role*`)`

`get_from_uri``(`*uri*`)`
Get from this tree the item at the given uri.

get_item(*idx*)
Just a prettier face in front of idx.internalPointer()

headerData(*section*, *orientation*, *data_role*)

index(*row*, *col*, *parent*)
Returns QModelIndex address of int row, int col, under QModelIndex parent. If a row, column, parent combination points to an invalid index, returns invalid QModelIndex().

insertRows(*row*, *count*, *parent=<PySide.QtCore.QModelIndex(-1*, *-1*, *0x0*, *QObject(0x0)) >*)

is_good_tag(*testtag*, *parent=<PySide.QtCore.QModelIndex(-1*, *-1*, *0x0*, *QObject(0x0)) >*)
Checks for usable tags, returns a (bool,string) tuple where the bool indicates whether the tag is good, and the string provides explanation if the tag is not good.

is_good_uri(*uri*)
Returns whether or not input uri points to an item in this tree.

iter_indexes(*parent=<PySide.QtCore.QModelIndex(-1*, *-1*, *0x0*, *QObject(0x0)) >*)
Provide a list of the QModelIndexes held in the tree

list_tags(*parent*)
Get a list of tags for TreeItems under parent.

parent(*idx*)
Returns QModelIndex of parent of item at QModelIndex index

print_tree(*rowprefix='‘*, *parent=<PySide.QtCore.QModelIndex(-1*, *-1*, *0x0*, *QObject(0x0)) >*)

removeRows(*row*, *count*, *parent=<PySide.QtCore.QModelIndex(-1*, *-1*, *0x0*, *QObject(0x0)) >*)

rowCount(*parent=<PySide.QtCore.QModelIndex(-1*, *-1*, *0x0*, *QObject(0x0)) >*)
Either give the number of top-level items, or count the children of parent

setData(*idx*, *value*, *data_role*)

staticMetaObject = <PySide.QtCore.QMetaObject object>

static **tag_error**(*tag*, *err_msg*)
Provide a human-readable error message for bad tags.

tree_dataChanged(*idx*)

tree_update(*idx*, *x_new*)
Call this function to store x_new in the TreeItem at idx and then build/update/prune the subtree rooted at that item. Take measures to change as little as possible of the tree, since this can be a big operation and is called frequently.

class paws.core.treemodel.**TreeSelectionModel**
Bases: *paws.core.treemodel.TreeModel*

columnCount(*parent*)
Let TreeSelectionModel have two columns: one for the TreeItem tag, one for selection status.

data(*itm_idx*, *data_role*)

flags(*idx*)

get_all_selected(*idx=<PySide.QtCore.QModelIndex(-1*, *-1*, *0x0*, *QObject(0x0)) >*)

headerData(*section*, *orientation*, *data_role*)

setData(*idx*, *value*, *data_role*)

set_all_unselected(*idx=<PySide.QtCore.QModelIndex(-1*, *-1*, *0x0*, *QObject(0x0)) >*)

staticMetaObject = <PySide.QtCore.QMetaObject object>

Module contents

paws.ui package

Subpackages

paws.ui.widgets package

Submodules

paws.ui.widgets.op_widget module

class paws.ui.widgets.op_widget.**OpWidget**(*op*)
Bases: PySide.QtGui.QWidget

paintEvent(*evnt*)

staticMetaObject = <PySide.QtCore.QMetaObject object>

paws.ui.widgets.pif_widget module

class paws.ui.widgets.pif_widget.**PifWidget**(*itm*)
Bases: PySide.QtGui.QTextEdit

print_comp(*itm*, *indent*)

print_id(*id_*, *indent*)

print_matrix(*itm*, *indent*)

print_pif(*itm*, *indent*)

print_procstep(*itm*, *indent*)

print_prop(*itm*, *indent*)

print_qty(*itm*, *indent*)

print_scalar(*itm*, *indent*)

print_src(*itm*, *indent*)

print_value(*itm*, *indent*)

print_vector(*itm*, *indent*)

staticMetaObject = <PySide.QtCore.QMetaObject object>

paws.ui.widgets.text_widgets module

Widgets for displaying text

paws.ui.widgets.text_widgets.**display_text**(*itm*, *indent='&nbsp;&nbsp;&nbsp;&nbsp;'*)

paws.ui.widgets.text_widgets.**display_text_fast**(*itm*, *indent='&nbsp;&nbsp;&nbsp;&nbsp;'*)

Module contents

widgets package

Submodules

paws.ui.data_viewer module

`paws.ui.data_viewer.`**`display_item`**(*itm*, *uri*, *qlayout*, *logmethod=None*)

paws.ui.input_loader module

class `paws.ui.input_loader.`**`InputLoader`**(*name*, *src*, *trmod=None*, *parent=None*)

Bases: `object`

This class controls the input_loader.ui to select inputs from various sources.

`add_items`()

`add_value`(*val*)

`lock_list_toggle`()

`rm_selection`()

`set_list_toggle`()

`setup_ui`()

`unlock_list_toggle`()

`unset_list_toggle`()

paws.ui.opuiman module

class `paws.ui.opuiman.`**`OpUiManager`**(*opman*)

Bases: `object`

`build_source`()

`clear_op`()

`edit_op`()

`enable_ops`()

`get_op`(*idx*)

`set_op`(*op*)

Set up ui elements around existing input op

`setup_ui`()

paws.ui.plotmaker_mpl module

paws.ui.plotmaker_mpl.**array_plot_1d**(*data_in*)

paws.ui.plotmaker_mpl.**array_plot_2d**(*data_in*)

paws.ui.plotmaker_mpl.**mpl_array_plot_1d**(*data_in*)

paws.ui.plotmaker_mpl.**mpl_array_plot_2d**(*data_in*)

paws.ui.plotmaker_mpl.**plot_mpl_fig**(*fig_in*)

paws.ui.plotmaker_pqg module

paws.ui.plotmaker_pqg.**array_plot_1d**(*data_in*)

paws.ui.plotmaker_pqg.**array_plot_2d**(*data_in*)

paws.ui.plotmaker_pqg.**plot_mpl_fig**(*fig_in*)

paws.ui.plotmaker_pqg.**pqg_array_plot_1d**(*data_in*)

paws.ui.plotmaker_pqg.**pqg_array_plot_2d**(*data_in*)

paws.ui.pluguiman module

class paws.ui.pluguiman.**PluginListModel**(*input_list=[]*, *parent=None*)

Bases: *paws.core.listmodel.ListModel*

Just a ListModel with overloaded headerData

headerData(*section*, *orientation*, *data_role*)

staticMetaObject = <PySide.QtCore.QMetaObject object>

class paws.ui.pluguiman.**PluginUiManager**(*plugman*)

Bases: object

build_input()

clear_input()

fetch_data(*name*)

fetch_from_input_ui(*ui*)

load_plugin()

Package the finished Plugin, ship to self.plugman

reset_input_headers()

reset_type_widget(*name*, *row*, *src=None*)

reset_val_widget(*name*, *row*, *src=None*, *tp=None*)

set_input(*name*, *src_ui=None*)

set_plugin(*pgin*)

setup_ui()

start_plugin(*idx*)

stop_plugin()
remove the selected plugin from self.plugman

paws.ui.ui_manager module

class paws.ui.ui_manager.**UiManager**(*opman*, *wfman*, *plugman*)
Bases: object

Stores a reference to a QMainWindow, performs operations on it

connect_actions()
Set up the works for buttons and menu items

display_item(*idx*)
Display selected item from the workflow tree in image_viewer

edit_ops(*item_indx=None*)
interact with user to enable existing Operations and edit or develop new Operations

edit_wf(*trmod*, *itm_idx=<PySide.QtCore.QModelIndex(-1, -1, 0x0, QObject(0x0)) >*)
Interact with user to edit the workflow. Pass in a TreeModel and index to open the editor with the item at that index loaded.

final_setup()

make_title()
Display the paws logo in the image viewer

msg_board_log(*msg*, *timestamp=<function timestr>*)
Print timestamped message to msg board

start_load_ui()

start_plugins_ui()

start_save_ui()

start_wf_editor(*trmod=None*, *indx=<PySide.QtCore.QModelIndex(-1, -1, 0x0, QObject(0x0)) >*)
Create a WfUiManager (QMainWindow), return it

toggle_run_wf()

paws.ui.uitools module

Configuration flags, widgets, and functions for paws gui control.

paws.ui.uitools.**bigtext_widget**(*text*)

paws.ui.uitools.**hdr_widget**(*text*)

paws.ui.uitools.**load_path**(*ui*, *idx=<PySide.QtCore.QModelIndex(-1, -1, 0x0, QObject(0x0)) >*)

paws.ui.uitools.**message_ui**(*parent=None*)

paws.ui.uitools.**name_widget**(*name*)

paws.ui.uitools.**r_hdr_widget**(*text*)

paws.ui.uitools.**save_path**(*ui*, *idx=<PySide.QtCore.QModelIndex(-1, -1, 0x0, QObject(0x0)) >*, *oldidx=<PySide.QtCore.QModelIndex(-1, -1, 0x0, QObject(0x0)) >*)

paws.ui.uitools.**save_to_file**(*d*, *filename*)
Save the items in dict d as YAML in filename

`paws.ui.uitools.`**`smalltext_widget`**`(`*text*`)`

`paws.ui.uitools.`**`src_selection_widget`**`()`

`paws.ui.uitools.`**`start_load_ui`**`(`*uiman*`)`
Start a modal window dialog to load a previously saved workflow

`paws.ui.uitools.`**`start_save_ui`**`(`*uiman*`)`
Start a modal window dialog to choose a save destination for the current workflow

`paws.ui.uitools.`**`stop_load_ui`**`(`*ui*, *uiman*`)`

`paws.ui.uitools.`**`stop_save_ui`**`(`*ui*, *uiman*`)`

`paws.ui.uitools.`**`text_widget`**`(`*text*`)`
Produce a Read-only Center-aligned QtGui.QLineEdit from input text.

`paws.ui.uitools.`**`toggle_expand`**`(`*trview*, *idx*`)`

`paws.ui.uitools.`**`toggle_load_button`**`(`*ui*, *txt*`)`

`paws.ui.uitools.`**`toggle_save_button`**`(`*ui*, *txt*`)`

`paws.ui.uitools.`**`type_selection_widget`**`(`*src*, *widg=None*`)`

paws.ui.wfuiman module

class `paws.ui.wfuiman.`**`WfUiManager`**`(`*wfman*, *opman*, *plugman*`)`
Bases: `object`

`build_input_locator``(`*name*, *ui=None*`)`
Create an InputLocator for named input. If a ui is provided, it should be an input_loader.ui, and this should load data from that ui.

`build_io``()`

`clear_io``()`

`create_op``(`*op*`)`
Instantiate op, call self.set_op()

`fetch_data``(`*name*`)`

`fetch_from_input_ui``(`*ui*`)`

`get_op``(`*trmod*, *itm_idx*`)`

`load_op``()`
Package the finished Operation, ship to self.wfman

`reset_input_headers``()`

`reset_output_headers``()`

`reset_type_widget``(`*name*, *row*, *src=None*`)`

`reset_val_widget``(`*name*, *row*, *src=None*, *tp=None*`)`

`rm_op``()`
remove the selected operation from the workflow

`set_input``(`*name*, *src_ui=None*`)`
Call build_input_locator to package an InputLocator for named input. Store it in self.op.input_locator[name]. This is called on all inputs when an Operation is loaded, so it should not alter an identical already-loaded input.

set_op(*op*, *uri*)
Set up ui elements around existing input op

setup_ui()

Module contents

Module contents

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX