# paws Documentation

*Release 0.7.1*

Lenson A. Pellouchoud

Jul 20, 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 library of operations, which are essentially wrappers around useful pieces of Python code.

PAWS itself is a pure Python package. The PAWS API is meant to provide the functionality of PAWS such that it can be used in data processing scripts or employed in the back end of applications. Limited graphical interfaces are under development to support specific processing workflows. Some of these interfaces are included in the PAWS package, but are not used unless explicitly called upon. To keep the package dependencies low, the PAWS GUI modules are not supported by the package dependencies. The PAWS GUI modules are based on Qt, written by way of the PySide bindings. Installing PySide (by `pip install PySide`, for example) should provide support for the PAWS GUI. In future releases, the GUI modules may be distributed in a separate package.

Some of the core goals of PAWS:

- Providing turnkey workflows for routine data analysis
- Scaling up of workflows to analyze batches of data
- Portable infrastructure for moving workflows from machine to machine
- Coupling data analysis to instrumentation for real-time results-driven feedback

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

# CHAPTER TWO

# INSTALLATION

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. To install PAWS, enter `pip install pypaws` at the command line from within your Python environment.

The only dependency of PAWS is pyyaml (TODO: link to pyyaml), used for serializing and de-serializing workflow data. pip should automatically install this along with PAWS.

The PAWS GUI modules are not explicitly supported by the package dependencies. To use PAWS GUI modules, install PySide into your Python environment with `pip install PySide` (TODO: link to PySide)

CHAPTER

THREE

# MODULES DOCUMENTATION

Full documentation of all PAWS modules, built with sphinx-apidoc.

## 3.1 paws

### 3.1.1 paws package

#### Subpackages

#### paws.api package

#### Module contents

This module defines a class that presents an API for paws.

**class** paws.api.**PawsAPI**

Bases: object

A container to facilitate interaction with a set of paws objects: an Operations Manager, a Workflow Manager, and a Plugins Manager.

**add_op**(*op_tag*, *op_spec*, *wfname=None*)

**add_plugin**(*pgin_tag*, *pgin_name*)

**add_wf**(*wfname*)

**current_wf**()

**disable_op**(*op_spec*)

Disable the Operation indicated by op_spec. The Operation becomes unavailable until it is enabled again.

**enable_op**(*op_spec*)

Import and enable the Operation indicated by op_spec. The Operation becomes available to add to workflows by paws.api.add_op()

**enable_plugin**(*pgin_name=''*)

This tests the compatibility between the environment and the named plugin by attempting to import the plugin. If this does not throw an ImportError, then the environment satisfies the plugin dependencies.

**execute**(*wfname=None*)

**get_op**(*opname*, *wfname=None*)

**get_output**(*opname*, *output_name*, *wfname=None*)

**get_plugin**(*pgin_name*)

**get_wf**(*wfname=None*)

**info**()

**list_op_tags**(*wfname=None*)

**op_count**(*wfname=None*)

**remove_op**(*op_tag*, *wfname=None*)

**save_config**()

**save_plugins**(*wfl_filename*)
Save the current set of plugins to a .wfl file, as specified by wfl_filename. If the given filename does not have the .wfl extension, it will be appended.

**save_workflow**(*wfl_filename*)
Save the current workflow to a .wfl file, as specified by wfl_filename. If the given filename does not have the .wfl extension, it will be appended.

**select_wf**(*wfname*)

**set_input**(*opname*, *input_name*, *wfname=None*, ***kwargs*)

**set_plugin_input**(*pgin_tag*, *input_name*, ***kwargs*)

**start_plugin**(*pgin_name*)

**write_log**(*msg*)

paws.api.**start**()
Instantiate and return a PawsAPI object.

paws.api.start() calls the PawsAPI constructor.

**Returns** a PawsAPI object

**Return type** paws.api.PawsAPI

## paws.core package

### Subpackages

### paws.core.models package

#### Submodules

#### paws.core.models.DictTree module

**class** paws.core.models.DictTree.**DictTree**(*data={}*)
Bases: object

A tree as an ordered dictionary (root), extended by embedding other objects that are amenable to tree storage. Fetches items by a uri string that is a sequence of dict keys, connected by '.'s.

Child items (end nodes of the tree) can be anything. Parent items, in order to index their children, must be either lists, dicts, or objects implementing keys(), __getitem__(key) and __setitem__(key,value).

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

**delete_uri**(*uri=''*)
Delete the given uri, i.e., remove the corresponding key from the embedded dict. This should not be relied on to be fast. It has to go through all of the uris to remove children.

**get_from_uri**(*uri=''*)
Return the data stored at uri. Each data item in the lineage of the uri must implement __getitem__() with support for string-like keys, unless it is a list, in which case the key is cast as int(key) before using it as an index in the list.

**is_tag_valid**(*tag*)
Check for validity of a tag. The conditions for a valid tag are the same as for a valid uri, except that a tag should not contain period (.) characters.

**is_uri_unique**(*uri*)
Check for uniqueness of a uri.

**is_uri_valid**(*uri*)
Check for validity of a uri. Uris may contain upper case letters, lower case letters, numbers, dashes (-), and underscores (_). Periods (.) are used as delimiters between tags in the uri. Any whitespace or any character in the string.punctuation library (other than -, _, or .) results in an invalid uri.

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

**print_tree**(*root_uri=''*, *rowprefix=''*)
Print the content of the tree rooted at root_uri, with each row of the string preceded by rowprefix.

**root_keys**()

**set_uri**(*uri=''*, *val=None*)
Set the data at the given uri to provided value val.

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

**uri_error**(*uri*)
Provide a human-readable error message for bad uris.

## paws.core.models.ListModel module

**class** paws.core.models.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**()

**n_items**()

**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>**

**class** paws.core.models.ListModel.**PluginListModel**(*input_list=[]*, *parent=None*)

Bases: *paws.core.models.ListModel.ListModel*

Just a ListModel with overloaded headerData

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

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

## paws.core.models.TreeItem module

**class** paws.core.models.TreeItem.**TreeItem**(*parent_itm*, *tag*)

Bases: object

A structured container for indexing a TreeModel. A TreeItem keeps references to a parent TreeItem and a list of child TreeItems. It is labeled by a tag (TreeItem.tag) which must be unique across its sibling TreeItems.

**n_children**()

## paws.core.models.TreeModel module

**class** paws.core.models.TreeModel.**TreeModel**

Bases: object

This class indexes a DictTree with a set of TreeItems. TreeItems keep track of their lineage in the DictTree, and can be modified for additional functionality in subclasses of TreeModel by adding TreeItem.flags.

**build_tree**(*x*)

TreeModel.build_tree is called on some object x before x is stored in the tree. For subclasses of TreeModel to build tree data for data types other than dicts and lists, build_tree should be reimplemented. If data types other than dicts and lists have child items that should be accessible by TreeModel uris, they should implement __getitem__(tag).

**build_uri**(*itm*)

Build a URI for TreeItem itm by combining the tags of the lineage of itm, with '.' as a delimiter.

**contains_uri**(*uri*)

**create_tree_item**(*parent_itm*, *itm_tag*)

Build a TreeItem for use in this tree. Reimplement create_tree_item() in subclasses of TreeModel to add features to TreeItems, such as default values for TreeItem.flags. TreeModel implementation returns TreeItem(parent_itm,itm_tag).

**get_data_from_idx**(*idx*)

**get_data_from_uri**(*uri*)

**get_from_uri**(*uri*)

**is_tag_valid**(*tag*)

**is_uri_valid**(*uri*)

**make_unique_uri**(*prefix*)

**n_children**(*parent_uri=''*)

**remove_item**(*itm_uri*)

**root_tags**()

**set_item**(*itm_uri*, *itm_data=None*)

**tag_error**(*tag*)

**tree_update**(*parent_itm*, *itm_tag*, *itm_data*)
Update the tree structure rooted at parent_itm.children[itm_tag], such that TreeItems get built to index all of the items in itm_data that are supported by self.build_tree(). Assume build_tree was called on itm_data before passing it as an argument, so only need to recurse if itm_data is a dict.

### Module contents

## paws.core.operations package

### Subpackages

## paws.core.operations.EXECUTION package

### Subpackages

## paws.core.operations.EXECUTION.BATCH package

### Submodules

### paws.core.operations.EXECUTION.BATCH.BatchFromFiles module

**class** paws.core.operations.EXECUTION.BATCH.BatchFromFiles.**BatchFromFiles**
Bases: `paws.core.operations.Operation.Batch`

Read a directory and filter its contents with a regular expression to form a list of file paths to be used as inputs for the repeated execution of a workflow. Specify, by workflow uri, where this file path will be fed to the workflow. Collect specified outputs from the workflow for each of the inputs.

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

**batch_outputs_tag**()

**input_list**()

**input_routes**()
Provide the input route- a list is expected

**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

**set_batch_ops**(*wf=None*)

**set_input_routes**(*wf=None*)

### paws.core.operations.EXECUTION.BATCH.BatchPostProcess module

**class** paws.core.operations.EXECUTION.BATCH.BatchPostProcess.**BatchPostProcess**
Bases: *paws.core.operations.Operation.Batch*

Take the batch output (list of dicts) from a previously completed Batch, and use each dict to form inputs for the execution of a post-processing workflow. For each input to be taken from the dict, two uris are needed: one to locate it within the (previous) batch outputs, and another to specify where it will be fed to the (current) workflow. Collect specified outputs from the workflow for each of the inputs.

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

**batch_outputs_tag**()

**input_list**()

**input_routes**()
Provide the input route- currently batch execution expects a 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

**set_batch_ops**(*wf=None*)

**set_input_routes**(*wf=None*)

### Module contents

### paws.core.operations.EXECUTION.REALTIME package

### Submodules

### paws.core.operations.EXECUTION.REALTIME.RealtimeFromFiles module

**class** paws.core.operations.EXECUTION.REALTIME.RealtimeFromFiles.**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.

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

**batch_outputs_tag**()

**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**()

**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

**set_batch_ops**(*wf=None*)

**set_input_routes**(*wf=None*)

### Module contents

### Module contents

### paws.core.operations.IO package

### Subpackages

### paws.core.operations.IO.BL15 package

### Submodules

### paws.core.operations.IO.BL15.ReadHeader_SSRL15 module

**class** paws.core.operations.IO.BL15.ReadHeader_SSRL15.**ReadHeader_SSRL15**
Bases: *paws.core.operations.Operation.Operation*

Read a .txt header from beamline 1-5 at SSRL into a dict.

**run**()

### paws.core.operations.IO.BL15.ReadImageAndHeader_SSRL15 module

**class** paws.core.operations.IO.BL15.ReadImageAndHeader_SSRL15.**ReadImageAndHeader_SSRL15**
Bases: *paws.core.operations.Operation.Operation*

Read an image and header generated by beamline 1-5 at SSRL. Returns ndarray image and dictionary header.

**run**()

### Module contents

### paws.core.operations.IO.CALIBRATION package

#### Submodules

#### paws.core.operations.IO.CALIBRATION.NikaToPONI module

**class** `paws.core.operations.IO.CALIBRATION.NikaToPONI.`**`NikaToPONI`**

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

Converts Nika calibration output (saved in a text file) to a dict of PyFAI PONI parameters, by first converting from Nika to Fit2D, then using a pyFAI.AzimuthalIntegrator to convert from Fit2D to PONI format.

WARNING: the map from Nika's horizontal and vertical tilts to Fit2D's tilt and tiltPlanRotation has not yet been verified by the developers. Use this operation with nonzero tilts at your own risk.

Input a text file expressing results of Nika automated calibration, and manually input polarization factor. Output a dict of pyFAI PONI calibration parameters. Format of text file for Nika output is expected to be: sample_to_CCD_mm=____ pixel_size_x_mm=____ pixel_size_y_mm=____ beam_center_x_pix=____ beam_center_y_pix=____ horizontal_tilt_deg=____ vertical_tilt_deg=____ wavelength_A=____

**run**()

#### paws.core.operations.IO.CALIBRATION.ReadPONI module

**class** `paws.core.operations.IO.CALIBRATION.ReadPONI.`**`ReadPONI`**

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

Read in a dict of PyFAI PONI parameters. Input path to a .poni file representing a calibrated measurement geometry.

**run**()

#### paws.core.operations.IO.CALIBRATION.WXDToPONI module

**class** `paws.core.operations.IO.CALIBRATION.WXDToPONI.`**`WXDToPONI`**

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

Convert WXDIFF .calib output to a dict of PyFAI PONI parameters, by first converting from WXDIFF to Fit2D, then using a pyFAI.AzimuthalIntegrator to convert from Fit2D to PONI format.

The conversion from WXDiff parameters to Fit2D parameters was originally contributed to paws by Fang Ren.

Input .calib file from WXDIFF automated calibration, input pixel size and polarization factor, output dict of pyFAI PONI calibration parameters.

**run**()

#### Module contents

The INPUT.CALIBRATION category has operations for reading in calibration parameters and converting them between different formats. Some of the common formats are described here. Over time, these descriptions should improve. Contact the paws developers to contribute information or report inconsistencies.

### PONI (PyFAI) FORMAT

PONI: point of normal incidence. This is the format used internally by the PyFAI (Python Fast Azimuthal Integration) package. PONI format projects the point-shaped sample orthogonally onto projector plane, and gives the coordinates of that projection as the PONI, such that the sample to PONI distance is the shortest distance from sample to detector plane. coordinate axes: x1 vertical, x2 and x3 horizontal, x3 along beam. detector axes: with zero rotations, d1 vertical, d2 horizontal, d3 along beam. axes defined on C format, first dimension is vertical, second dimension is horizontal. the first dimension (vertical) is fast, the second dimension (horizontal) is slow.

PONI dict keys and definitions: - 'dist': distance in meters from sample to PONI on detector plane - 'poni1': vertical coordinate of PONI on detector axes, in meters - 'poni2': horizontal coordinate of PONI on detector axes, in meters - 'rot1': rotation of detector body about x1, applied first, radians - 'rot2': rotation of detector body about x2, applied second, radians - 'rot3': rotation of detector body about beam axis x3, applied third, radians - 'pixel1': pixel dimension along d1 (vertical), meters - 'pixel2': pixel dimension along d2 (horizontal), meters - 'wavelength': wavelength in meters - 'fpolz': polarization factor- not actually a PONI parameter, but it's ok to put it here - 'detector': optional pyFAI detector object - 'splineFile' optional spline file describing detector distortion

### NIKA FORMAT

The calibration performed by the Nika software package uses a calibrant image, the rectangular pixel dimensions (in mm), and the wavelength (in Angstrom), to solve the sample to CCD distance in mm, the position at which the beam axis intersects the detector plane in pixels, and the horizontal and vertical tilts of the detector in degrees.

Nika does not generate a file to save calibration parameters, so they have to be recorded by hand in a file. Paws Operations should be written to read them from a file in the following format (one parameter=value per line, no spaces): - sample_to_CCD_mm=____ - pixel_size_x_mm=____ - pixel_size_y_mm=____ - beam_center_x_pix=____ - beam_center_y_pix=____ - horizontal_tilt_deg=____ - vertical_tilt_deg=____ - wavelength_A=____

### FIT2D FORMAT

Detector plane origin is the bottom left corner of the detector.

Fit2D dict keys and definitions: - 'directDist': direct distance to detector plane along beam axis, in mm - 'centerX': horizontal position on the detector plane where the beam intersects, in px - 'centerY': vertical position on the detector plan where the beam intersects, in px - 'pixelX': horizontal size of pixel, in um - 'pixelY': vertical size of pixel, in um - 'tilt': detector tilt in degrees (TODO:clarify) - 'tiltPlanRotation': detector rotation in degrees = 360 minus WXDIFF alpha (TODO:clarify) - 'splineFile' optional spline file describing detector distortion

### WXDIFF FORMAT

Similar to Fit2D format, but knowledge about WXDIFF is hard to come by. I hope it can be cleanly documented here over time. Detector plane origin is the bottom left corner of the detector.

.calib file lines (and notes): - imagetype=uncorrected-q TODO: describe - dtype=uint16 img data type = unsigned 16-bit integers - horsize=___ horizontal extent of image, in pixels - versize=___ vertical extent of image, in pixels - region_ulc_x=___ TODO: describe - region_ulc_y=___ TODO: describe - bcenter_x=___ horizontal coordinate where the beam axis intersects the detector plane - bcenter_y=___ vertical coordinate where the beam axis intersects the detector plane - detect_dist=___ direct distance from the sample to the detector plane intersection, along the beam axis, in pixels - detect_tilt_alpha=___ rotation of detector tilt axis plane in radians = 360 minus Fit2D tiltPlanRotation - detect_tilt_delta=___ detector tilt in radians (TODO:clarify) - wavelenght=___ the typo 'wavelenght' is built into wxdiff, and it is reported in angstroms - Qconv_const=___ TODO: describe

### paws.core.operations.IO.CSV package

#### Submodules

#### paws.core.operations.IO.CSV.CSVToArray module

**class** `paws.core.operations.IO.CSV.CSVToArray.`**`CSVToArray`**

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

Read a csv-formatted file into a numpy array.

**`run`**()

#### paws.core.operations.IO.CSV.CSVToXYData module

**class** `paws.core.operations.IO.CSV.CSVToXYData.`**`CSVToXYData`**

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

Read a csv-formatted file into x values and y values.

**`run`**()

#### paws.core.operations.IO.CSV.ReadCSV_q_I_dI module

**class** `paws.core.operations.IO.CSV.ReadCSV_q_I_dI.`**`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.IO.CSV.WriteArrayCSV module

**class** `paws.core.operations.IO.CSV.WriteArrayCSV.`**`WriteArrayCSV`**

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

Write a 2d array to a csv file

**`run`**()

#### paws.core.operations.IO.CSV.WriteCSV_q_I_dI module

**class** `paws.core.operations.IO.CSV.WriteCSV_q_I_dI.`**`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.IO.CSV.WriteCSV_q_I_dI.`**`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.IO.CSV.WriteCSV_q_I_dI.**write_csv_q_I_maybe_dI**(*q*, *I*, *dI*, *nameloc*)

### Module contents

### paws.core.operations.IO.IMAGE package

### Submodules

### paws.core.operations.IO.IMAGE.FabIOOpen module

**class** paws.core.operations.IO.IMAGE.FabIOOpen.**FabIOOpen**

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.IO.IMAGE.LoadTif module

**class** paws.core.operations.IO.IMAGE.LoadTif.**LoadTif**

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

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

**run**()

### paws.core.operations.IO.IMAGE.LoadTif_PIL module

**class** paws.core.operations.IO.IMAGE.LoadTif_PIL.**LoadTif_PIL**

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

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

**run**()

### Module contents

### paws.core.operations.IO.MISC package

### Submodules

### paws.core.operations.IO.MISC.ReadNPSynthRecipe module

**class** paws.core.operations.IO.MISC.ReadNPSynthRecipe.**ReadNPSynthRecipe**

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

Read in a text file describing nanoparticle synthesis parameters. Package the description in a dict.

**run**()

### Module contents

### paws.core.operations.IO.PIF package

### Submodules

### paws.core.operations.IO.PIF.CheckDataSet module

**class** `paws.core.operations.IO.PIF.CheckDataSet.`**`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**()

### paws.core.operations.IO.PIF.CreateDataSet module

**class** `paws.core.operations.IO.PIF.CreateDataSet.`**`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**()

### paws.core.operations.IO.PIF.ShipJSON module

**class** `paws.core.operations.IO.PIF.ShipJSON.`**`ShipJSON`**

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

Take a .json file containing a pif or array of pifs, ship it to a Citrination data set.

**run**()

### paws.core.operations.IO.PIF.ShipToDataSet module

**class** `paws.core.operations.IO.PIF.ShipToDataSet.`**`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.IO.BuildFilePath module

**class** `paws.core.operations.IO.BuildFilePath.`**`BuildFilePath`**

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

This operation helps to build file paths from workflow data. It takes a directory (full path), a filename, and an extension. The filename can optionally have a prefix or suffix inserted, to help with iteration of batches of files with similar names.

**`run`**`()`

### Module contents

### paws.core.operations.PACKAGING package

### Subpackages

### paws.core.operations.PACKAGING.BATCH package

### Submodules

### paws.core.operations.PACKAGING.BATCH.BuildListFromBatch module

**class** `paws.core.operations.PACKAGING.BATCH.BuildListFromBatch.`**`BuildListFromBatch`**

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

Given a batch output and a batch output uri, harvest a list of outputs from the batch.

**`run`**`()`

### paws.core.operations.PACKAGING.BATCH.XYDataFromBatch module

**class** `paws.core.operations.PACKAGING.BATCH.XYDataFromBatch.`**`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.PACKAGING.BL15 package

#### Submodules

#### paws.core.operations.PACKAGING.BL15.TimeTempFromHeader module

**class** `paws.core.operations.PACKAGING.BL15.TimeTempFromHeader.`**`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`**()

#### Module contents

### paws.core.operations.PACKAGING.PIF package

#### Submodules

#### paws.core.operations.PACKAGING.PIF.EmptyPif module

**class** `paws.core.operations.PACKAGING.PIF.EmptyPif.`**`EmptyPif`**

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*, *T_C*)

#### paws.core.operations.PACKAGING.PIF.PifNPSynth module

**class** `paws.core.operations.PACKAGING.PIF.PifNPSynth.`**`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*, *temp_C*)

#### paws.core.operations.PACKAGING.PIF.PifTernary module

**class** `paws.core.operations.PACKAGING.PIF.PifTernary.`**`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.LogLogZip module

**class** paws.core.operations.PACKAGING.LogLogZip.**LogLogZip**

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

Take the base-10 logarithm of two 1d arrays, then zip them together. Any elements with non-positive values are removed.

**run**()

### paws.core.operations.PACKAGING.WindowZip module

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

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

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

**run**()

paws.core.operations.PACKAGING.WindowZip.**logsafe_zip**(*x*, *y*)

paws.core.operations.PACKAGING.WindowZip.**window_zip**(*x*, *y*, *x_min*, *x_max*)

paws.core.operations.PACKAGING.WindowZip.**zip**(*x*, *y*)

### paws.core.operations.PACKAGING.Zip module

**class** paws.core.operations.PACKAGING.Zip.**Zip**

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

Zip two 1d arrays together.

**run**()

### Module contents

### paws.core.operations.PROCESSING package

### Subpackages

### paws.core.operations.PROCESSING.BACKGROUND package

### Submodules

### paws.core.operations.PROCESSING.BACKGROUND.BgSubtractByTemperature module

**class** `paws.core.operations.PROCESSING.BACKGROUND.BgSubtractByTemperature.`**`BgSubtractByTempe`**

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

Originally contributed by Amanda Fournier.

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**()

### paws.core.operations.PROCESSING.BACKGROUND.SubtractMaximumBackground module

**class** `paws.core.operations.PROCESSING.BACKGROUND.SubtractMaximumBackground.`**`SubtractMaximum`**

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

Originally contributed by Amanda Fournier.

Subtract a background from a foreground, with scaling to prevent over-subtraction. Has optional arguments for error vectors (default None).

**run**()

### Module contents

### paws.core.operations.PROCESSING.BASIC package

### Submodules

### paws.core.operations.PROCESSING.BASIC.ArrayMirrorHorizontal module

**class** `paws.core.operations.PROCESSING.BASIC.ArrayMirrorHorizontal.`**`ArrayMirrorHorizontal`**

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

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

**run**()

#### paws.core.operations.PROCESSING.BASIC.ArrayMirrorVertical module

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

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

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

**`run`**`()`

#### paws.core.operations.PROCESSING.BASIC.Rotation module

**class** `paws.core.operations.PROCESSING.BASIC.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']

### Module contents

### paws.core.operations.PROCESSING.CALIBRATION package

#### Submodules

#### paws.core.operations.PROCESSING.CALIBRATION.AutoCal module

Operation for automatic discovery of diffraction image calibration parameters.

**class** `paws.core.operations.PROCESSING.CALIBRATION.AutoCal.`**`AutoCal`**

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

Input image data (ndarray), pixel size, polz factor. Output dictionary of calibration parameters.

**`run`**`()`

#### paws.core.operations.PROCESSING.CALIBRATION.CalReduce module

Calibrate and reduce an image, given calibration parameters.

This module calls on PyFAI.AzimuthalIntegrator to calibrate an input image to I(q,chi), and then reduce it to I(q).

**class** `paws.core.operations.PROCESSING.CALIBRATION.CalReduce.`**`CalReduce`**

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

Input image data (ndarray) and a dict of .poni format calibration parameters Output q, I(q)

**`run`**`()`

#### paws.core.operations.PROCESSING.CALIBRATION.Calibrate module

Calibrate and reduce an image, given calibration parameters.

This module calls on PyFAI.AzimuthalIntegrator to calibrate an input image to I(q,chi).

**class** paws.core.operations.PROCESSING.CALIBRATION.Calibrate.**Calibrate**

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

Input image data (ndarray) and a dict of calibration parameters Return q, chi, I(q,chi)

**run**()

#### Module contents

### paws.core.operations.PROCESSING.FEATURE_EXTRACTION package

#### Submodules

#### paws.core.operations.PROCESSING.FEATURE_EXTRACTION.IntensityFeatures module

Created on Mon Jun 06 18:02:32 2016

author(s): Fang Ren, Apurva Mehta Module originally contributed by Fang Ren. For details, refer to the recent paper submitted to ACS Combinatorial Science. TODO: Get this citation

**class** paws.core.operations.PROCESSING.FEATURE_EXTRACTION.IntensityFeatures.**IntensityFeature**

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

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

**run**()

#### paws.core.operations.PROCESSING.FEATURE_EXTRACTION.TextureFeatures module

Created on Mon Jun 06 18:02:32 2016

author(s): Fang Ren, Apurva Mehta Module originally contributed by Fang Ren. For details, refer to the recent paper submitted to ACS Combinatorial Science. TODO: Get this citation

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

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

Analyze the texture

**run**()

#### Module contents

### paws.core.operations.PROCESSING.PEAKS package

#### Submodules

#### paws.core.operations.PROCESSING.PEAKS.FindPeaksByWindow module

**class** paws.core.operations.PROCESSING.PEAKS.FindPeaksByWindow.**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**()

## paws.core.operations.PROCESSING.PEAKS.VoigtPeakFit module

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

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

Fit a set of x and y values to a Voigt distribution. Solves the best-fitting hwhm (half width at half max) of the gaussian and lorentzian distributions and shared 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, half width at half max 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, half width at half max 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

## Module contents

## paws.core.operations.PROCESSING.SAXS package

## Submodules

## paws.core.operations.PROCESSING.SAXS.SpectrumFit module

**class** `paws.core.operations.PROCESSING.SAXS.SpectrumFit.`**`SpectrumFit`**

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

Use a measured SAXS spectrum (I(q) vs. q), to optimize the parameters of a theoretical SAXS spectrum for one or several populations of scatterers. Works by minimizing an objective function that compares the measured spectrum against the theoretical result. TODO: document the algorithm here.

Input arrays of q and I(q), a string indicating choice of objective function, a dict of features describing the spectrum, and a list of strings indicating which keys in the dict should be used as optimization parameters. The input features dict includes initial fit parameters as well as the flags indicating which populations to include. The features dict is of the same format as SpectrumProfiler and SpectrumParameterization outputs.

Outputs a return code and the features dict, with entries updated for the optimized parameters. Also returns the theoretical result for I(q), and a renormalized measured spectrum for visual comparison.

**run**()

### paws.core.operations.PROCESSING.SAXS.SpectrumParameterization module

**class** `paws.core.operations.PROCESSING.SAXS.SpectrumParameterization.`**`SpectrumParameterizatio`**

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

The algorithm for guessing parameters for the size distributions of spherical nanoparticles was developed and originally contributed by Amanda Fournier.

This operation uses a SAXS spectrum (I(q) vs. q), along with some profiling information, to guess a set of parameters for fitting the SAXS spectrum to theoretical scattering models. TODO: document scattering models.

A precursor is modeled by a dilute, monodisperse spherical form factor. A spherical nanoparticle population is modeled by a discrete sum over a probability distribution of dilute, monodisperse spherical form factors. A crystalline arrangement is modeled by a sum of pseudo-Voigt peaks.

Outputs a return code and heuristic guesses for SAXS model parameters. Also outputs the theoretical result for I(q) with the guessed parameters.

This Operation is somewhat robust for noisy data, but any preprocessing (background subtraction, smoothing, or other cleaning) should be performed beforehand.

**`run`**`()`

### paws.core.operations.PROCESSING.SAXS.SpectrumProfiler module

**class** `paws.core.operations.PROCESSING.SAXS.SpectrumProfiler.`**`SpectrumProfiler`**

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

This operation profiles a SAXS spectrum (I(q) vs. q) to determine some characteristics of the sample. Based on some measures of the overall shape of the spectrum, the spectrum is tested for scattering from precursors (approximated as small dilute monodisperse spheres), scattering from dilute nanoparticles of various form factors (currently only spheres), diffraction peaks (Voigt-like profiles due to crystalline arrangements), or some combination of the three. TODO: document algorithm here.

Output a return code and a dictionary of the results.

This Operation is somewhat robust for noisy data, but any preprocessing (background subtraction, smoothing, or other cleaning) should be performed beforehand.

**`run`**`()`

### Module contents

### paws.core.operations.PROCESSING.SMOOTHING package

### Submodules

### paws.core.operations.PROCESSING.SMOOTHING.MovingAverage module

**class** `paws.core.operations.PROCESSING.SMOOTHING.MovingAverage.`**`MovingAverage`**

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

Applies moving average smoothing filter to 1d array, optionally weighted by window shape and error values.

**`run`**`()`

#### paws.core.operations.PROCESSING.SMOOTHING.SavitzkyGolay module

`class paws.core.operations.PROCESSING.SMOOTHING.SavitzkyGolay.`**`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 (default None).

**`run`**`()`

#### Module contents

### paws.core.operations.PROCESSING.ZINGERS package

#### Submodules

#### paws.core.operations.PROCESSING.ZINGERS.EasyZingers1D module

`class paws.core.operations.PROCESSING.ZINGERS.EasyZingers1D.`**`EasyZingers1D`**

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

This Operation attempts to remove zingers from 1-D spectral data (I(q) versus q). Zingers are replaced with the average intensity in a window around where the zinger was found.

**`run`**`()`

#### Module contents

#### Module contents

### paws.core.operations.TESTS package

#### Submodules

#### paws.core.operations.TESTS.BigLoad module

`class paws.core.operations.TESTS.BigLoad.`**`BigLoad`**

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

An Operation testing class, creates and outputs a big array of noise

**`run`**`()`

#### 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.OpManager module

**class** paws.core.operations.OpManager.**OpManager**

Bases: *paws.core.models.TreeModel.TreeModel*

Tree structure for categorized storage and retrieval of Operations.

**add_op**(*cat*, *opname*)
Add op name to the tree under category cat.

**load_cats**(*cat_list*)

**load_ops**(*cat_op_list*)
Load OpManager tree from input cat_op_list. Format of cat_op_list is [(category1,opname1),(category2,opname2),. . . ]. 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 name of the Operation. load_cats() should be called before load_ops() and should ensure that all cats in cat_op_list exist in the tree.

**n_ops**()

**print_cat**(*cat_uri*, *rowprefix=' '*)
Generate a string that lists the contents of the operations category specified by cat_uri

**remove_op**(*op_uri*)
Remove op from the tree by its full category.opname uri

**set_op_enabled**(*op_uri*, *flag=True*)

**write_log**(*msg*)

### paws.core.operations.Operation module

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

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

**batch_outputs_tag**()
Return the output name (one of the self.outputs.keys()) that indicates where the batch outputs should be stored.

**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 [output_uri:output_value] pairs.

**saved_items**()
Return a list of items (as workflow uris) to be saved after each execution.

**set_batch_ops**(*wf=None*)
: Set enough information in this Operation's inputs so that self.batch_ops() returns the correct list of operations to be run under the batch. Takes a Workflow as optional second argument, so that it can be used to call optools.locate_input()

**set_input_routes**(*wf=None*)
: Set enough information in this Operation's inputs so that self.input_routes() returns the correct list of workflow uris where the batch will set its inputs Takes a Workflow as optional second argument, so that it can be used to call optools.locate_input()

**class** `paws.core.operations.Operation.`**InputLocator**(*src=0*, *tp=0*, *val=None*)
Bases: `object`

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

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

Class template for implementing paws operations.

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

**doc_as_string**()

**input_description**()

**keys**()

**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`*

**batch_outputs_tag**()
: Return the output name (one of the self.outputs.keys()) that indicates where the outputs should be stored.

**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, input_iter().next() should return None.

**input_routes**()
: Produce a list of the input routes used by the Realtime, in the same order as each of the OrderedDicts provided by Realtime.input_iter().

**output_list**()
: Produce a list of OrderedDicts representing the outputs for each Realtime input. Each OrderedDict should be populated with [output_uri:output_value] pairs.

**saved_items**()
Return a list of items (as workflow uris) to be saved after each execution.

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

### paws.core.operations.optools module

Operations config and processing routines

**exception** paws.core.operations.optools.**ExecutionError**(*msg*)
Bases: exceptions.Exception

**class** paws.core.operations.optools.**FileSystemIterator**(*dirpath*, *regex*, *include_existing_files=True*)
Bases: _abcoll.Iterator

**next**()

paws.core.operations.optools.**batch_op_stack**(*wf*, *batch_op_tag*, *valid_wf_inputs*)
Use batch_op.batch_ops() and a list of valid_wf_inputs to build a stack (list) of lists of operations suitable for serial execution.

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.**check_wf**(*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.

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

paws.core.operations.optools.**execution_stack**(*wf*)
Build a stack (list) of lists of Operation uris, such that each list indicates 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_name,[batch_stack]], where batch_name indicates the batch controller Operation, and batch_stack is built from batch_op_stack().

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

paws.core.operations.optools.**get_valid_wf_inputs**(*op_tag*, *operation*)
Return the TreeModel uris of the op and its inputs/outputs that are eligible as downstream inputs in the workflow.

paws.core.operations.optools.**is_op_ready**(*wf*, *op_tag*, *valid_wf_inputs*, *batch_routes=[]*)

paws.core.operations.optools.**load_inputs**(*operation*, *wf=None*, *plugin_manager=None*)
Loads input data for an Operation from its input_locators. A Workflow and a PluginManager can be provided as optional arguments, in which case they are used to fetch data.

paws.core.operations.optools.**locate_input**(*il*, *wf=None*, *plugin_manager=None*)
Return the data pointed to by a given InputLocator object. A Workflow and a PluginManager can be provided as optional arguments, in which case they are used to fetch data.

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.**disable_ops**(*disable_root*) `paws.core.operations.`**`load_ops_from_path`**`(`*path_*, *pkg*, *cat_root=''*`)`

`paws.core.operations.`**`save_config`**`()`

Call save_config() before closing to save the state of which ops are enabled/disabled.

`paws.core.operations.`**`update_load_flags`**`()`

### paws.core.plugins package

### Submodules

### paws.core.plugins.CitrinationPlugin module

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

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

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

**`content`**`()`

**`description`**`()`

**`ship_dataset`**`(`*pifs*`)`

**`start`**`()`

**`stop`**`()`

### paws.core.plugins.PawsPlugin module

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

Bases: `object`

**`content`**`()`

PawsPlugin.content() returns a dict containing the meaningful objects contained in the plugin. The default implementation returns an empty dict.

**`description`**`()`

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

**`keys`**`()`

**`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.PluginManager module

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

Bases: *`paws.core.models.TreeModel.TreeModel`*

Tree structure for managing paws plugins.

**build_tree**(*x*)
Reimplemented TreeModel.build_tree() so that TreeItems are built from PawsPlugins and Workflows and Operations.

**get_plugin**(*pgin_type*)

**list_plugins**()

**load_from_dict**(*plugin_dict*)
Load plugins from a dict that specifies their setup parameters.

**plugin_setup_dict**(*pgin*)

**write_log**(*msg*)

## paws.core.plugins.SpecClientPlugin module

**class** paws.core.plugins.SpecClientPlugin.**SpecClientPlugin**
Bases: *paws.core.plugins.PawsPlugin.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.PawsPlugin.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.WfManagerPlugin module

**class** paws.core.plugins.WfManagerPlugin.**WfManagerPlugin**

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

This plugin exposes the content of the workflow manager.

**content**()

**description**()

**start**()

**stop**()

### Module contents

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

### paws.core.tools package

### Submodules

### paws.core.tools.saxstools module

paws.core.tools.saxstools.**compute_pearson**(*y1*, *y2*)

paws.core.tools.saxstools.**compute_saxs**(*q*, *params*)

Given q and a dict of parameters, compute the saxs spectrum. Supported parameters are the same as SaxsParameterization Operation outputs, and should include at least the following keys: I_at_0, precursor_flag, form_flag, structure_flag.

TODO: Document the equation.

paws.core.tools.saxstools.**compute_saxs_with_substitutions**(*q*, *d*, *x_keys*, *x_vals*)

paws.core.tools.saxstools.**compute_spherical_normal_saxs**(*q*, *r0*, *sigma*)

Given q, a mean radius r0, and the fractional standard deviation of radius sigma, compute the saxs spectrum assuming spherical particles with normal size distribution. The returned intensity is normalized such that I(q=0) is equal to 1.

paws.core.tools.saxstools.**fit_I0**(*q*, *I*, *order=4*)

Find an estimate for I(q=0) by polynomial fitting. All of the input q, I(q) values are used in the fitting.

`paws.core.tools.saxstools.`**`fit_with_slope_constraint`**(*q*, *I*, *q_cons*, *dIdq_cons*, *order*, *weights=None*)

Perform a polynomial fitting of the low-q region of the spectrum with dI/dq(q=0) constrained to be zero. This is performed by forming a Lagrangian from a quadratic cost function and the Lagrange-multiplied constraint function.

TODO: Explicitly document cost function, constraints, Lagrangian.

Inputs q and I are not standardized in this function, so they should be standardized beforehand if standardized fitting is desired. At the provided constraint point, q_cons, the returned polynomial will have slope dIdq_cons.

Because of the form of the Lagrangian, this constraint cannot be placed at exactly zero. This would result in indefinite matrix elements.

`paws.core.tools.saxstools.`**`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 any sort.

`paws.core.tools.saxstools.`**`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 any sort.

`paws.core.tools.saxstools.`**`precursor_heuristics`**(*q*, *I*, *I_at_0=None*)

Makes an educated guess for the radius of a small scatterer that would produce the input q, I(q). Result is bounded between 0 and 10 Angstroms.

`paws.core.tools.saxstools.`**`saxs_Iq4_metrics`**(*q*, *I*)

From an input spectrum q and I(q), compute several properties of the I(q)*q^4 curve. This was designed for spectra that are dominated by a dilute spherical form factor term. The metrics extracted by this Operation were originally intended as an intermediate step for estimating size distribution parameters for a population of dilute spherical scatterers.

Returns a dict of metrics. Dict keys and meanings: q_at_Iqqqq_min1: q value at first minimum of I*q^4 I_at_Iqqqq_min1: I value at first minimum of I*q^4 Iqqqq_min1: I*q^4 value at first minimum of I*q^4 pIqqqq_qwidth: Focal q-width of polynomial fit to I*q^4 near first minimum of I*q^4 pIqqqq_Iqqqqfocus: Focal point of polynomial fit to I*q^4 near first minimum of I*q^4 pI_qvertex: q value of vertex of polynomial fit to I(q) near first minimum of I*q^4 pI_Ivertex: I(q) at vertex of polynomial fit to I(q) near first minimum of I*q^4 pI_qwidth: Focal q-width of polynomial fit to I(q) near first minimum of I*q^4 pI_Iforcus: Focal point of polynomial fit to I(q) near first minimum of I*q^4

TODO: document the algorithm here.

`paws.core.tools.saxstools.`**`saxs_fit`**(*q*, *I*, *objfun*, *features*, *x_keys*, *constraints=[]*)

Fit a saxs spectrum (I(q) vs q) to the theoretical spectrum for one or several scattering populations. Input objfun (string) specifies objective function to use in optimization. Input features (dict) describes spectrum and scatterer populations. Input x_keys (list of strings) are the keys of variables that will be optimized. Every item in x_keys should be a key in the features dict. Input constraints (list of strings) to specify constraints.

Supported objective functions: (1) 'full_spectrum_chi2': sum of difference squared across entire q range. (2) 'full_spectrum_chi2log': sum of difference of logarithm, squared, across entire q range. (3)

'full_spectrum_chi2norm': sum of difference divided by measured value, squared, aross entire q range. (4) 'low_q_chi2': sum of difference squared in only the lowest half of measured q range. (5) 'low_q_chi2log': sum of difference of logarithm, squared, in lowest half of measured q range. (6) 'pearson': pearson correlation between measured and modeled spectra. (7) 'pearson_log': pearson correlation between logarithms of measured and modeled spectra. (8) 'low_q_pearson': pearson correlation between measured and modeled spectra. (9) 'low_q_pearson_log': pearson correlation between logarithms of measured and modeled spectra.

Supported constraints: (1) 'fix_I0': keeps I(q=0) fixed while fitting x_keys.

TODO: document the objective functions, etc.

`paws.core.tools.saxstools.`**`spherical_normal_heuristics`**`(`*q*, *I*, *I_at_0=None*`)`

This algorithm was developed and originally contributed by Amanda Fournier.

Performs some heuristic measurements on the input spectrum, in order to make educated guesses for the parameters of a size distribution (mean and standard deviation of radius) for a population of spherical scatterers.

TODO: Document algorithm here.

`paws.core.tools.saxstools.`**`spherical_normal_heuristics_setup`**`()`

## Module contents

## paws.core.workflow package

## Submodules

## paws.core.workflow.WfManager module

**`class`** `paws.core.workflow.WfManager.`**`WfManager`**`(`*plugin_manager*`)`

Bases: `object`

Manager for paws Workflows. Stores a list of Workflow objects, performs operations on them. Keeps a reference to a PluginManager for access to PawsPlugins.

**`add_wf`**`(`*wfname*`)`

Add a workflow to self.workflows, with key specified by wfname. If wfname is not unique (i.e. a workflow with that name already exists), this method will overwrite the existing workflow with a new one.

**`build_op_from_dict`**`(`*op_setup*, *opman*`)`

**`execute_batch`**`(`*wfname*, *batch_op_tag*, *batch_stk*`)`

**`execute_realtime`**`(`*wfname*, *rt_op_tag*, *rt_stk*`)`

**`execute_serial`**`(`*wfname*, *op_list*`)`

**`load_from_dict`**`(`*wfname*, *opman*, *opdict*`)`

Create a workflow with name wfname. If wfname is not unique, self.workflows[wfname] is overwritten. Input opdict specifies operation setup, where each item in opdict provides enough information to get an Operation from OpManager opman and set up its Operation.input_locators.

**`n_wf`**`()`

**`op_setup_dict`**`(`*op*`)`

**`run_wf`**`(`*wfname*`)`

Serially execute the operations of WfManager.workflows[wfname]. Uses optools.execution_stack() to determine execution order.

**update_embedded_dict** (*d*, *d_new*)

**uri_to_embedded_dict** (*uri*, *data=None*)

**write_log** (*msg*)

### paws.core.workflow.Workflow module

**class** paws.core.workflow.Workflow.**Workflow**
Bases: *paws.core.models.TreeModel.TreeModel*

Tree structure for a Workflow built from paws Operations.

**build_tree** (*x*)
Reimplemented TreeModel.build_tree() so that TreeItems are built from Operations.

**keys** ()

**list_op_tags** ()

**n_ops** ()

**op_dict** ()

**set_op_input_at_uri** (*uri*, *val*)
Set an op input at uri to provided value val. The uri must be a valid uri in the TreeModel, of the form opname.inputs.inpname.

### Module contents

### Submodules

### paws.core.pawstools module

**exception** paws.core.pawstools.**LazyCodeError**
Bases: exceptions.Exception

**exception** paws.core.pawstools.**OperationDisabledError**
Bases: exceptions.Exception

**exception** paws.core.pawstools.**WorkflowAborted**
Bases: exceptions.Exception

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

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

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

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

paws.core.pawstools.**update_file** (*filename*, *d*)
Save the items in dict d into filename, without removing members not included in d.

#### Module contents

### Submodules

#### paws.paws_config module

#### Module contents

CHAPTER

# FOUR

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX