Orange pylost structure and features

# Basic structure

A simple structure describing the orange-pylost software is presented below.

Dictionary example

* Comment (string)
* Module (string)
* Height (MetrologyData)

Data Viewer Orange

Orange Plot1D/Plot2D

Orange Data Mining

Silx

Pylost widget

Dictionary data

Dictionary data

PylostBase

Most pylost widgets are derived from PylostBase, which manages many background functions for the widgets. Data is managed typically in dictionaries, and primary datasets like slopes/heights are managed through MetrologyData class.

# Data structures

Primary datasets like height, slopes\_x and slopes\_y are used as type MetrologyData, which extends numpy ndarray and astropy Quantity.

MetrologyData

Astropy.Quantity

Numpy ndarray

The parameters in MetrologyData are

* Unit: derived from astropy Quantity. Adding, subtracting or other operations on datasets with incompatible units are not allowed. To ignore units, use dataset.value.
* Dim\_detector: Array describing detector dimensions.  
  e.g. dim\_detector = [False, True, True] for the 3D subaperture images. The image dimensions are last two, and they are measured by a detector from a Metrology instrument
* Pix\_size: Array describing pixel sizes for the detector dimensions. It is set to one along non-detector dimensions. Each pix\_size element is a Quantity, i.e. it has a value and unit  
  e.g. pix\_size = [Quantity <1.0>, Quantity <0.03 mm>, , Quantity <0.03 mm>]
* Axis\_names: Names for all axes. E.g. [‘Motor’, ‘Y’, ‘X’]
* Axis\_values: Partial alternative to pix\_size. Axis values can be specified for all axes. The number of values for an axis should match the dataset shape along that axis. Values can be specified for detector/non-detector axes. Currently axis values are used when pixel size not available (e.g. LTP data).   
  Axis value can point to other parameters like motors. E.g. axis\_values = [‘motor\_x’, ‘’, [0, 0.5, 1, 1.5, …]]  
  TODO: Prioritize using axis values wherever possible, instead of pixel size
* Init\_shape: Initial shape of dataset, especially the detector shape (e.g. CCD size). It is typically loaded from raw instrument data
* Index\_list: Stores index positions of data.   
  E.g. Assume a dataset with shape (3, 10, 10) with start position (4, 5) in pixels.  
  index\_list = [ np.array([0,1,2]),   
   np.array([4,5,6,7,8,9,10,11,12,13]),   
   np.array([5,6,7,8,9,10,11,12,13,14]) ]  
  Index list is updated after operations like slicing or flipping.  
  E.g. slicing above dataset with [:, ::2, :], i.e. take alternative points along axis 1, would also update index list.   
  index\_list = [ np.array([0,1,2]),   
   np.array([4,6,8,10,12]),   
   np.array([5,6,7,8,9,10,11,12,13,14]) ]
* Motors: List of motor positions for X/Y/Z/RX/RY/RZ, or any other motors  
  E.g. [ {Name : motor\_X, values : np.array([0,0.5,1,1.5,…]), unit : mm, axis : 0},  
   {Name : motor\_Y, values : np.array([0,0.5,1,1.5,…]), unit : mm, axis : 0} ]  
  TODO: May be use dictionary instead of list
* Flags: Dictionary with any additional flags  
  E.g. { gravity\_removed : True,  
   flip\_data\_along\_Y : True }

The start position of the dataset is represented within index\_list and hence no other parameter is added for that.

# PylostBase

The PylostBase implements primarily five different sections, described as follows.

1. InitWidgetManager: This class provides frequently used initialization options like (a) info box at the top of widgets showing comment log, (b) Modules dropdown and (c) Scans dropdown.  
     
   Modules define a loose structure to the dictionary data passed between pylost widgets. For example   
   E.g.1. module = custom, load sequence of subapertures / load a height calibration file or any simple file loading with data represented as a simple dictionary with no sub-dictionaries  
   data = { module : custom,  
    height : MetrologyData([]),  
    filename : //xyz.dat   
   }  
   E.g.2. module = scan\_data, measurements done in multiple scans typically with same instruments  
   data = { module : scan\_data,  
    scan\_data : {  
    Scan\_0 : { height : MetrologyData([]),}  
    Scan\_1 : { height : MetrologyData([]),}  
    }   
   }  
     
   - Modules differ from hard structure provided MetrologyData, which pre-defines the parameters associated with dataset like height/slopes such as units or pixel size.   
   - Modules may just specify skeleton tree structure for nested dictionaries for inputs/outputs of pylost widgets. New modules could defined to include nexus format.
2. ModuleManager: This class manages modules. Currently only ‘custom’/’scan\_data’ module are used, and previously used modules ‘stitch\_data’/’stitch\_avg’/scan\_avg’ are redundant and not used anymore. More modules can be defined in future based on the need.   
   - Module items are processed in parallel, i.e. applying mask/fit or other functions would equally apply for all scans while using ‘scan\_data’ module.  
   - Modules are divided into two classes MODULE\_SINGLE / MODULE\_MULTIPLE depending on the repletion of primary dataset, e.g. ‘custom’ belongs to MODULE\_SINGLE as it allows only a single height/slopes\_x/slopes\_y to be defined, whereas ‘scan\_data’ can allow multiple height datasets under different scans.  
   - The class manages the interaction between data of different modules, e.g. removing a ‘custom’ reference data from all datasets in ‘scan\_data’  
   - Multiple modules are possible within a single dictionary data. In such cases, module dropdown is added in pylost widgets, to select the module for applying the widget functionalities
3. ScanDataManager: This class manages actions within a single scan, such as integration of slopes or differentiation of heights.
4. DataVisualizationManager: This class displays different datasets defined in DATA\_NAMES (e.g. heights, slopes\_x, slopes\_y) using pylost data viewers (DataViewerFrameOrange) inherited from silx data viewer frame. If the pylost widget accepts multiple inputs, their datasets are stacked before displaying.
5. PylostBase: This class inherits the four classes mentioned above. It also implements some common functionalities useful across all widgets.   
   E.g.1. Update comment log, display log in the info box  
   E.g.2. Loop over scans and apply widget function for each scan

Update comment ()

Apply scan item

* For subap\_j in **scan**:  
  apply item ( subap\_j )

Apply scans

* For scan\_i in all\_scans:  
  apply ( scan\_i )

**PylostBase**

Slopes y

Slopes x

Height

DataVisualizationManager

ScanDataManager

If (height in input\_data) & enable\_slopes:  
 output = {height, slopes\_x=dh/dx, slopes\_y=dh/dy}

If (slopes x/y in input\_data) & enable\_height:  
 output = {height=ʃ (sx\*dx + sy\*dy), slopes\_x, slopes\_y}

InitWidgetManager

ModuleManager

Manage modules of input/output data

1: Input data 1

* module = custom
* height = MetrologyData([])

Info: comment log

2: Input data 2

* module = scan\_data
* scan\_data
  + Scan\_0
    - height = MetrologyData([])
  + Scan\_1
    - height = MetrologyData([])

Select Module

Select Scan

# Data Viewer Frame Orange

The data viewer from silx is extended with additional options such as mask drawing tools and MetrologyData visualization, as shown below.

**DataViewerFrameOrange**

Info view (silx.DataView)

Preview data (silx.DataView)

* Stats: rms, pv etc.
* Customized turbo colormap

MetrologyData options:

* Scale X/Y with pixel size
* Offset start position
* Edit params like motors, unit in **Info View**

Mask tools

Silx . DataViewerFrame

# Dictionary Viewer

A new widget is created to visualize dictionary tree structure. It extends PyQt5.QTreeWidget with some additional functions.

* Display for each node item name. For dataset node, show also its shape/size, units
* Manage item selection. E.g. get selected path, get selected data, edit/move item etc.

# OrangePlot1D, OrangePlot2D

Silx plot widgets for 1D and 2D are extended with additional functions to create these classes. Some of the additional functions are given below.

* Display legends in 2D plots
* Offset X/Y, export data to excel, change displayed line width/color, change alpha on selection etc.
* Many functions such as rename legend or offset x are applied for all the displayed plots within the Visualize Stitch results / compare widgets. Hence, some functions are defined within those widgets.  
  TODO: Create a new PlotManager which can propagate changes across different plots

# VisCompareBase

A base class is created for visualization classes, which perform comparison of datasets. Currently Visualize Compare and Visualize Stitch Results widgets extends this base class.

Results Tab

Vis Compare Base

Visualize Stitch Results widget

Visualize Compare 1D/2D widget

Compare Tab Base

Statistics Tab

Compare Data Tab

Noise Stats Tab

* Results tab merges stitched datasets like heights, reference from different input channels in the Visualize Stitch Results widget.
* Statistics tab shows comparison of plot rms, pv, radius etc., it also shows same parameters for plots in noise tab
* Compare and Noise tabs show exactly similar plots and hence are derived from a common base. The tab base manages display of plots, merging color maps of multiple images, pin the plots at origin or at center, normalize each plot, show plot stats (rms, pv, x-offsets, normalization scale etc). If the datasets are MetrologyData, they are displayed in the units of first item
* In the noise tab, average of 2D images or 1D curves is subtracted respectively.

All the Visualization widgets are (re)loaded when the widget is opened, and hence minimizes the overhead of reloading each time any upstream widget is modified. For faster loading only the default tab is loaded (compare tab) when the widget is opened, and other tabs are loaded only when clicked on the tab.

# Operator base

A base class Operator is created for operator widgets like add, subtract, multiply, divide and merge. It primarily manages how to apply operation on two or more inputs, with different modules.

All operators are by default enabled for any number of inputs, and it is applied as follows

Out = inp1 Ꚛ inp2 Ꚛ inp3 Ꚛ …

Where Ꚛ represents the operation

# Some selected widgets

## File loaders: Data (File)

File / file sequence loader

Output

Input

* Height/slopes (MetrologyData)
* Module = custom

Import selected / all

Convert to MetrologyData

Load File sequence

Load File

Data

* Height/slopes
* Motors
* Header data
* …

Data can be loaded from a file / file sequence or from an another pylost widget and it is converted to MetrologyData with known names for pylost (height / slopes\_x / slopes\_y). A dataviewer is added to visualize the data on clicking a dataset. Finally, in the output, any other necessary information is added/modified within Info View in the dataviewer.

## File loaders: Data (Scans)

Scans loader

Optionally flip some scans (last subap to first)

Finally, load scans

Optionally include subfolders

* Each folder is a scan

Split scans with format string or by an integer

Sort filenames

* By time or name or specified string

Load folder

* Loads all filenames

Select a filereader

Loads many repeated measurements as scans and outputs data as scan\_data module.