felapps Documentation

Release 1.5.13

Tong Zhang

Sep 08, 2016

CONTENTS

felapps Python package

`felapps`: FEL High-level Applications.

Install: `pip install felapps`

PDF documentation: `Download`

Author Tong Zhang

E-mail zhangtong@sinap.ac.cn

Date 2015-2016

CHAPTER

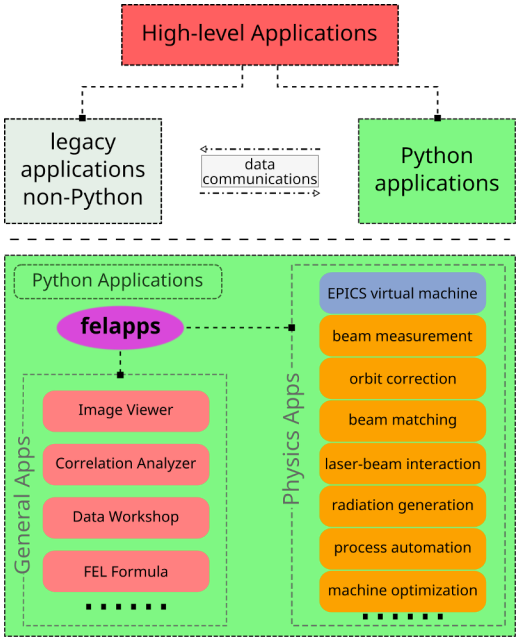
ONE

INTRODUCTION

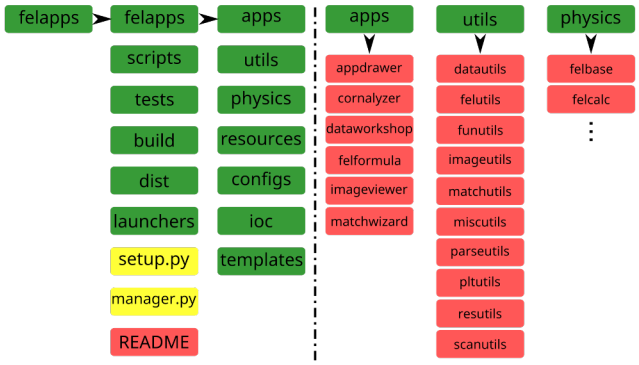
`felapps` is a Python package developed for the commissioning of free-electron laser facility, with the intention of providing useful tools for the high-level applications, both command line interface (CLI) and graphical user interface (GUI) are developed/under development.

The package name stands for "FEL High-level Applications", the core building language is `Python`, of the version of `2.7.x`. Since `Python` is a dynamic interpreted high-level language, pure object-oriented, various powerful third-party packages support, all of these great features make it possible to build software system in a real efficient way.

Below is the basic framework of `felapps`, general-purpose applications and specific-purpose applications are both included, specific-purpose applications mean physics-related, both LINAC and FEL physics.



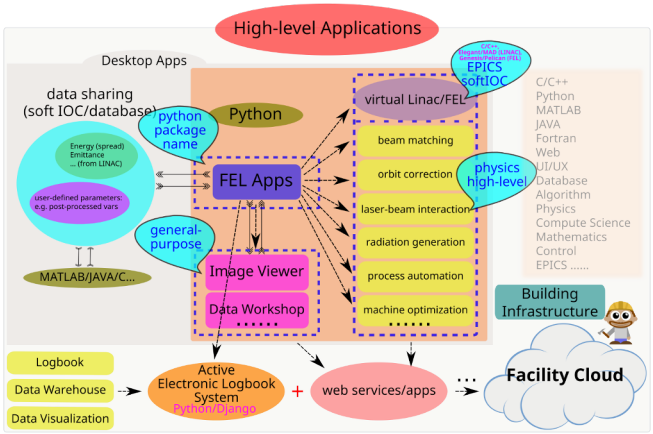
To make the package be well-organized, the source code is structured by the following style:



Building such Python-based software system requires much more effort to understanding the following key points (including but not limited to):

- potentially demandings of the FEL facility commissioning;
- software architecture of the system to be alike;
- comprehesive relationship between the lower-level control system and the high-level applications system;
- interfaces between various different layers;
- deployment of the entire software system;
- documentations;
- etc.

Picture of felapps



This the incubated picture of `felapps` package, which including *Desktop Apps* and *Web Apps* categories, also indicating the developing orientation, i.e. to take the advantages of Python, to build desktop apps to be an eco-system with bundle of well-defined APIs , to serve the web apps as service infrastructures, the ultimate goal should be a facility cloud with plenty of services that could be utilized by the facility, e.g. free-electron laser facility.

Specifically, in the beginning of such vision, Desktop Apps should be built, into which general-purpose and physics-related apps/classes/functions/methods should be integrated, just as the picture illustrates. As for the user, the deployment of `felapps` should be easy and robust. The future developing phase should

be web-based apps, to achieve such goal, the suggested beginning point should be the web-based log book, i.e. to develop an e-log web system by `Python`, incorporating the power from Python web-development community.

Progress of felapps

Up to now (2016.8), the ready-to-deploy/use apps from `felapps` are:

Graphical User Interface Apps

- `appdrawer` [1]: alias: `runfelapps`, portal to reach other GUI apps;
- `cornalyzer`: parameters correlation analysis app;
- `dataworkshop`: data post-processing app, feeding the data generated by `imageviewer`'s `Auto Save` operation module;
- `felformula`: efficient calculator for FEL physics;
- `imageviewer`: general-purpose profile/CCD monitor/DAQ app;
- `latticeviewer` [2]: latice online modeling framework;
- `wxmpv`: (to be) general data plotting & processing app.

Command Line Interface Apps

- `lte2json` [2]: standardize the `.lte` file for `Elegant` simulation, i.e. convert `.lte` file to JSON string file;
- `json2lte` [2]: convert (valid) JSON string file to `.lte` file;
- `update-felapps-menu`: script of integrating `felapps` into system menu, as well as the reversed operation, i.e. uninstalling.

[1] App name that could be treated as the command name in the command line interface, to get the app running, the same rule applies to other apps.
[2] Python package `beamline` is required.

CHAPTER

TWO

DEPLOYMENT

Deploy `felapps` to different operating systems is quite simple, both online and offline approaches are provided. Before installing this package into system, there may be packages/libraries dependence issues to be resolved first.

Prerequisites

Required Python packages: `numpy`, `scipy`, `matplotlib`, `pyepics`, `h5py`, `pyrpn`, `beamline`, `lmfit`, `wxPython`.

Packages with version information:

```
numpy      : 1.11.1
scipy      : 0.17.1
matplotlib : 1.5.1
pyepics    : 3.2.6
h5py       : 2.6.0
pyrpn      : 1.0.3
beamline   : 1.3.5
lmfit      : 0.9.3
wxPython   : 3.0.2.0
```

Note:

- `wxPython` should be built with unicode support
- Install them by `pip install` <package_name> (recommanded)

Installation

Online approach

```
pip install felapps --prefix=/opt/high-level-apps -i https://pypi.python.org/pypi
```

or precisely define the version number:

```
pip install felapps==1.5.13 --prefix=/opt/high-level-apps -i https://pypi.python.org/pypi
```

Offline approach

Download felapps from PyPI.

```
pip install felapps-1.5.13-py2.py3-none-any.whl --prefix=/opt/high-level-apps
```

Note:

- `root` privilege may be asked.

Configurations

Environmental Variables

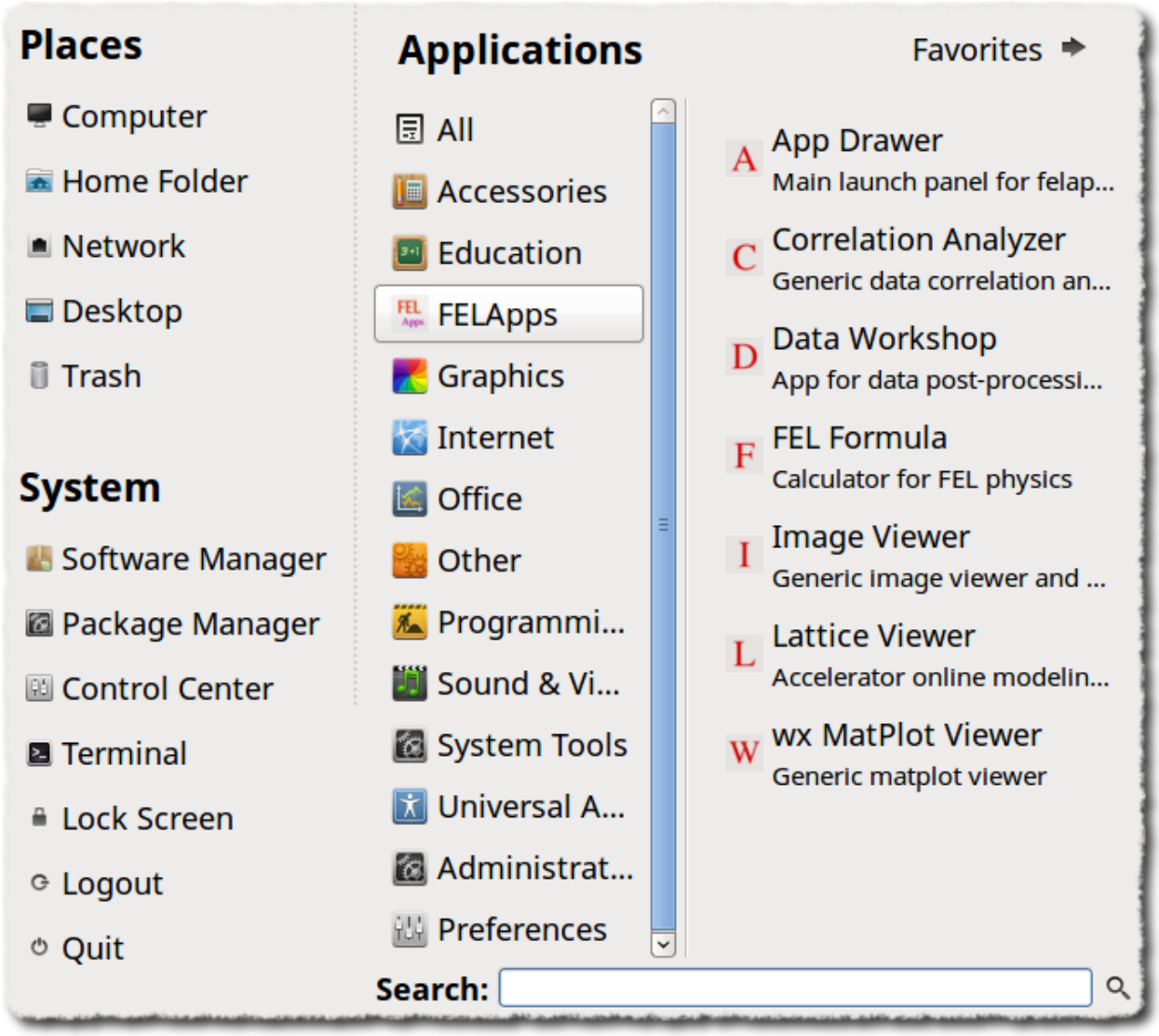
After installing `felapps`, system-wide environment variables should be configured correctly, say save the following shell script to a file named `felapps.sh`, put it into folder: `/etc/profile.d`, or append them into the bash configuration files.

```
HLAPATH=/opt/high-level-apps
HLAPYLIB=${HLAPATH}/lib/python2.7/site-packages
HLAPYBIN=${HLAPATH}/bin/
export PYTHONPATH=${HLAPYLIB}
export PATH=${PATH}:${HLAPYBIN}
```

System-menu Integration

To integrate the available apps into system menu would greatly improve the user experience, just like the following image shows [1]: all the apps are categorized into `FEL Apps` menu, through which you can reach different apps to accomplish different tasks.

[1] The menu screenshot may differ from other operating systems.



`felapps` distributes a script called: `update-felapps-menu` to handle the system-menu integration issues, here is the usage:

- Install `felapps` into system-menu:

```
update-felapps-menu --appdir /opt/high-level-apps/
```

- Uninstall `felapps` from system-menu:

```
update-felapps-menu --appdir /opt/high-level-apps/ --operation uninstall
```

Warning: If executing the above command with the `root` privilege, all the users would see the menu, or only the current user could see them.

CHAPTER

THREE

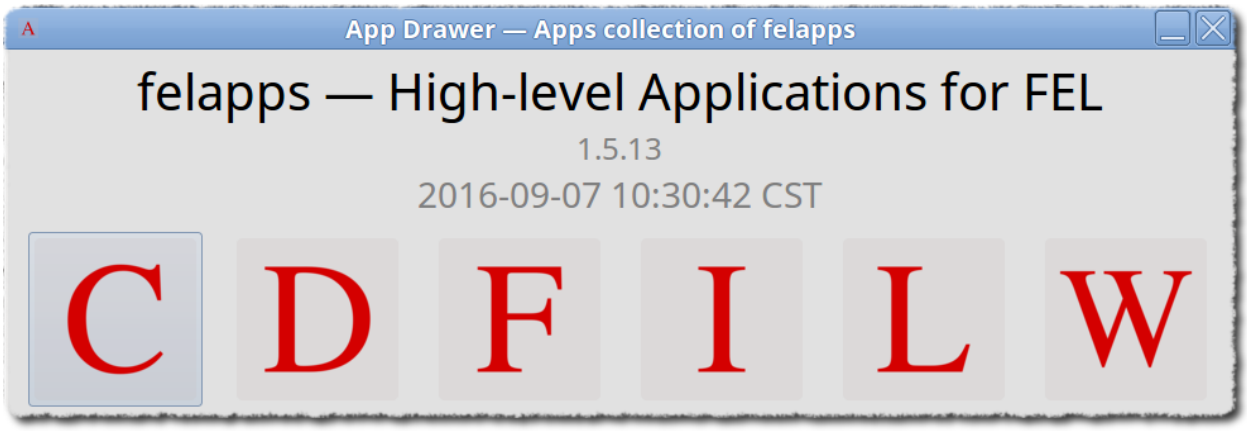
EXAMPLES AND DEMONSTRATIONS

Here goes through the main features that `felapps` provides, among which `latticeviewer` is an app based on `beamline` package.

Warning: The operating system is Linux (64 bit), distributions that have been tested: *Scientific Linux 6.4*, *CentOS 6.4*, *Linux Mint 17.x, 18* (*Ubuntu 15.x, 16.x*), other distros should work provided proper configurations. Part of features should work in Windows OS, but not has been fully tested.

In the following examples, the invocation of the apps could be one of these approaches:

1. Type <app name> in the terminal;
2. Browse <app name> from `FELApps` system menu;
3. Invoke `runfelapps` or `appdrawer`, click <app name> from the app's panel (see the following image).



Example 1

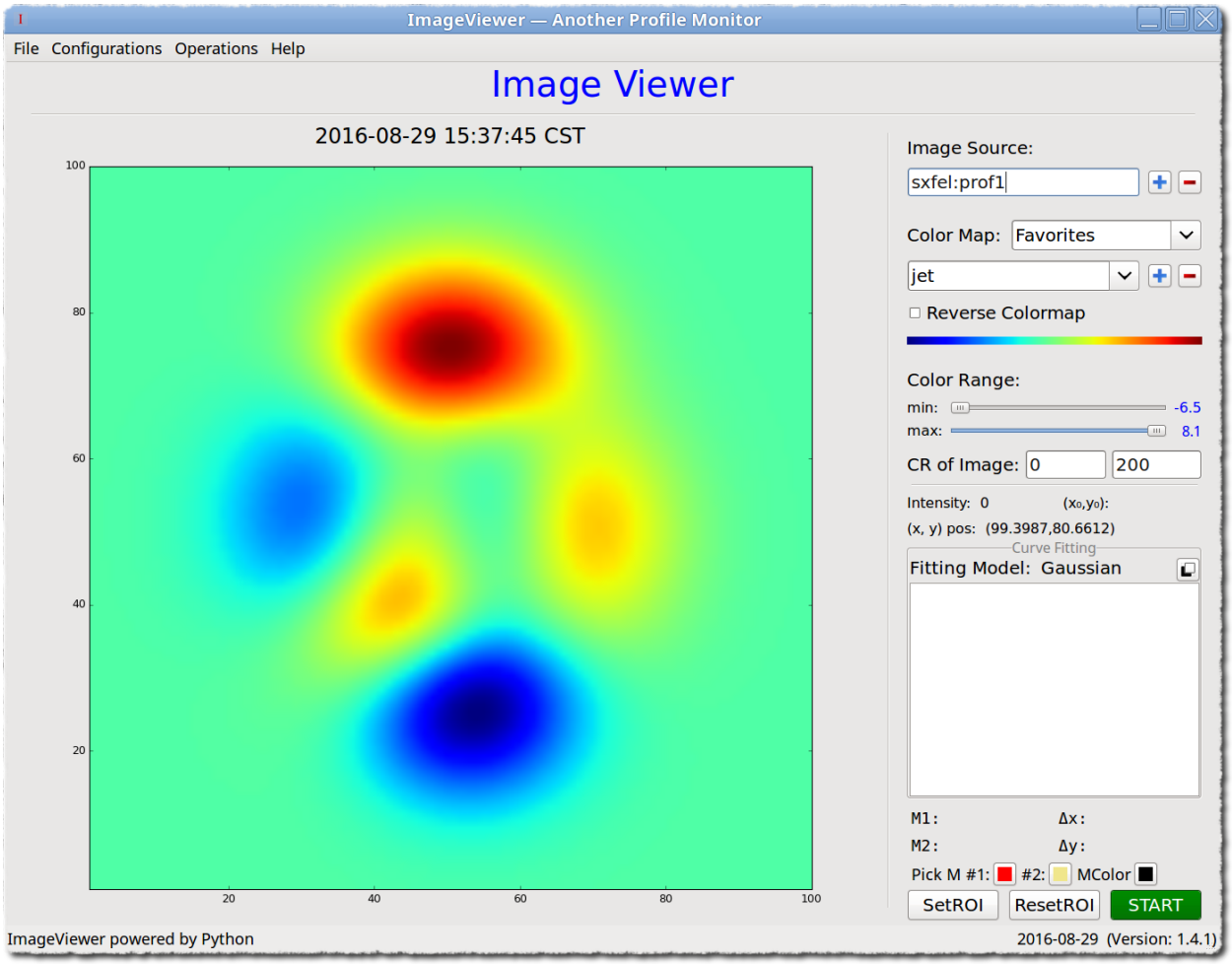
`imageviewer` – Image figure DAQ app.

Start DAQ

If it is the first time to start `imageviewer`, windows may be poped up to ask for the configuration file [1]. After loading the configuration file, the GUI style, e.g. background colour, font could be tunned, as well as

[1] Configuration file that `imageviewer` requires is a `.xml` formated file, a sample file could be downloaded `here`.

other options, e.g. the data/image format that to be saved, the DAQ rate, image data source, EPICS control configurations, etc.



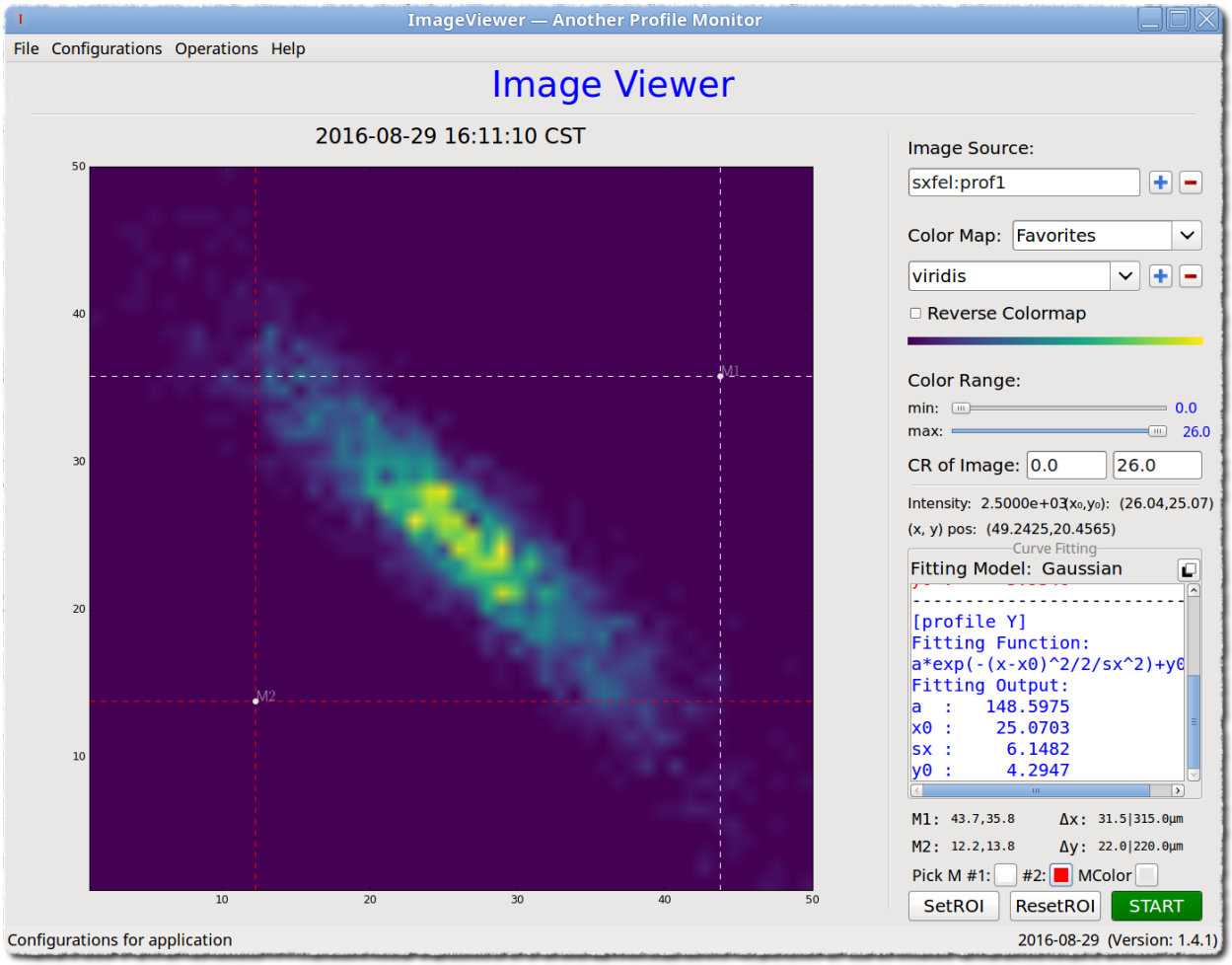
Some basic operations:

- Put the EPICS PV string name in the `Image Source:` box, press ENTER to retrieve the data (1D array), the `+/-` button is used to add/remove input PV string name to/from the internal favorite PV name array;
- Change the color map style of the obtained image, from `Color Map:` control field, select the grouped style from the combo list, add/remove current style to `Favorites` group;
- Reverse the color map, and change the color range, all these color options are of the goal to discriminate the interested signal from the big picture, i.e. to increase visibility;
- **To the figure plot region, there are operations:**
  - Place up to two (2) markers with selected color;
  - Track the `(x,y)` coordinate when mouse is moving;
  - Show the figure intensity and Gaussian fit indicated radius sizes in x and y;
- Choose and reset Region-of-Interest, i.e. ROI.

Other operations that could be reached through menu:

- Save figure as some formated image [2], shortcut: `CTRL+S`;
- Save figure as some formated data, shortcut: `CTRL+D`;
- Quit app: shortcut: `CTRL+W` [3];
- Save configurations: `SHIFT+CTRL+S`, load configurations: `SHIFT+CTRL+L`;
- Show profile along x-axis: `ALT+X`;
- Show profile along y-axis: `ALT+Y`;
- Show intensity monitor window: `SHIFT+CTRL+V`, the figure could also be saved;
- Save fetched image data automatically: `ALT+S` [4];

The following image is captured when `imageviewer` successfully fetch image data from PV named `sxfel:prof1` [5], which is a `50 x 50` array after reshaping operation, push `START` will trig the DAQ procedure with the configured rate `Preferences > Control > Monitor Frequency` [6].



Presently, `imageviewer` has been integrated Gaussian fitting model to show the x/y profiles of the image,

[2] The default image format could be set from menu: `Configurations > Preferences > Image Name Extension` or `SHIFT+CTRL+I` to open `Preferences` window, same rule applies to save figure data.

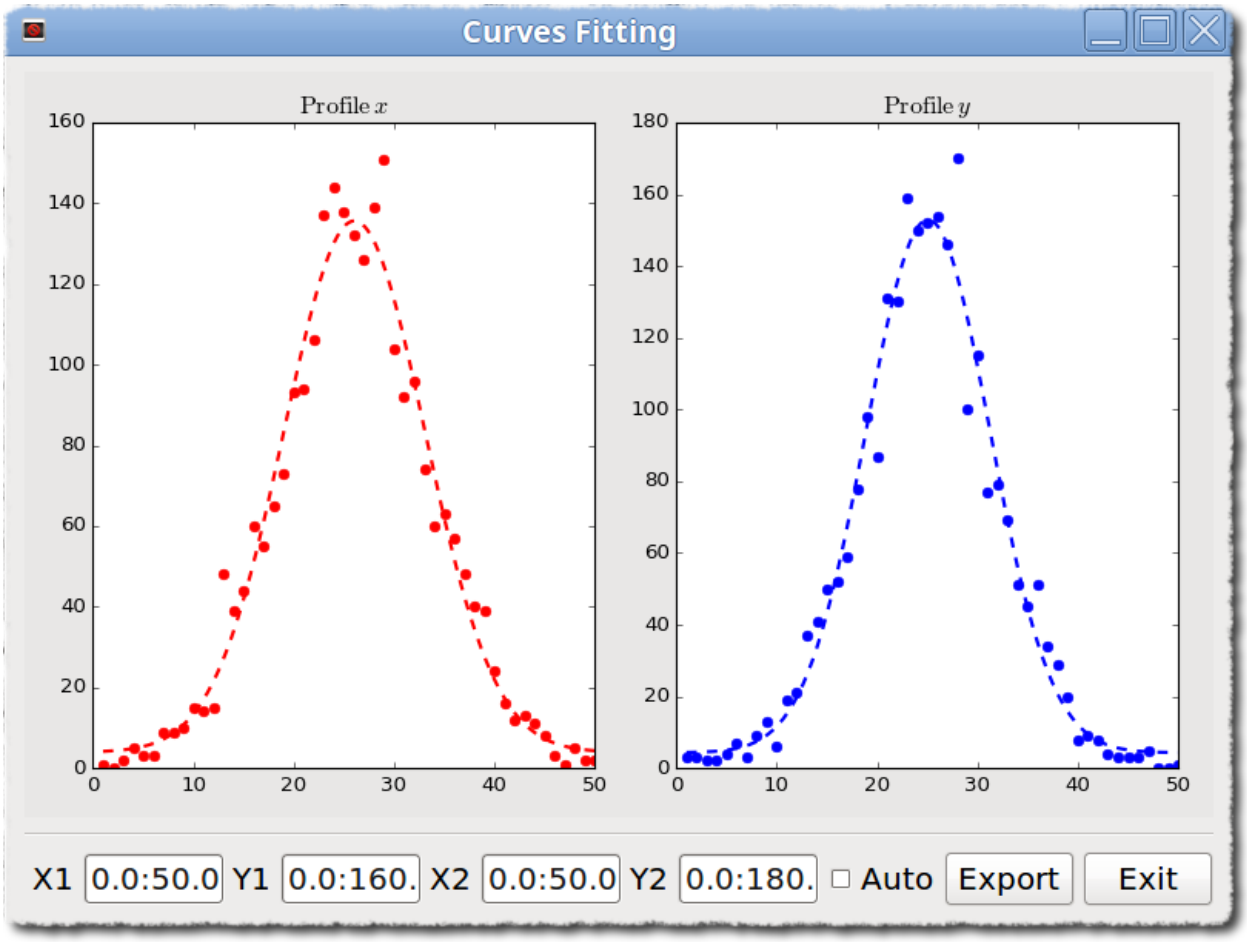
[3] `CTRL+W` is a common shortcut to close the app for all apps in `felapps`.

[4] Auto saved data files could be used as the input data files of app: `dataworkshop`, which serves as a simple data post-processor.

[5] `sxfel:prof1` is a waveform record of an soft-IOC that is used to simulate the EPICS control environment.

[6] It is suggested that increasing this value gradually, e.g. by step of 0.5, e.g. `1 Hz` for large image, and `5 Hz` for small image.

click the pop up button right aside of `... Gaussian ...` in the `Curve Fitting` panel of the main window, e.g. the following image shows the profiles.



The raw data and fitting data could be saved by `Export` button.

Note: Gaussian fitting model should be frequently used in the searching for radius size, central position of the light/beam spot, if other models should be applied, the `imageviewer` app needs to be extended.

Warning: Attention should be paid to the following configuration items:

1. `Image Width` and `Image Height`, which is the dimension (2D array) to reshape;
2. `CA Library Path`, `CA Max Array Bytes` and `Auto CA Address`, which is the control specific parameters, proper tunning is required so as to fetch the image data from soft-IOC.

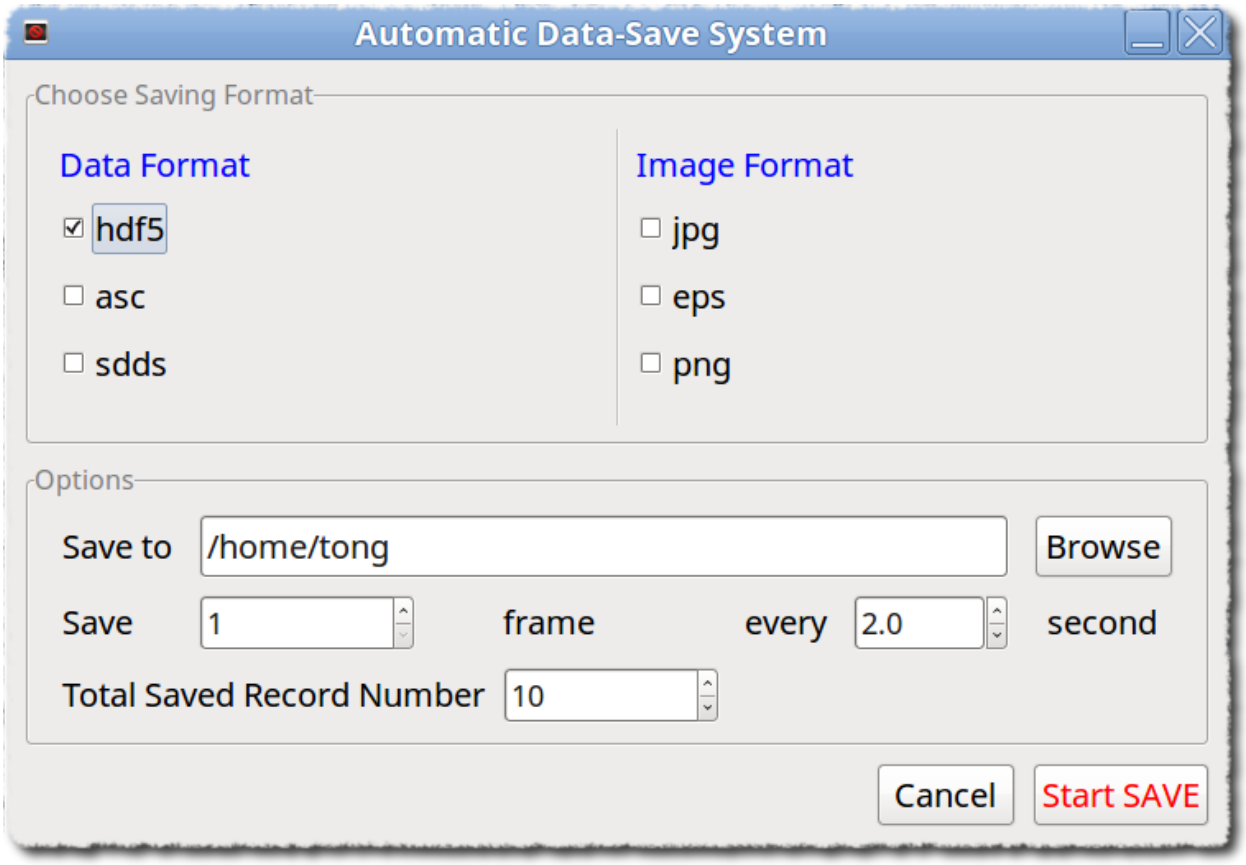
Auto Save

Here take the `Auto save` operation as example. `Auto save` is one of the operation modules in the `Operations` menu, future extended operation modules should be inserted into this menu. The shortcut for `Auto save` is `ALT+S`. `Auto save` is designed for the following scenario:

- The facility (e.g. FEL) finally has been properly configured;

- DAQ procedure should be activated to get bundle of data, e.g. for some kind of analysis.

Pop up `Auto save` window, in which, tick the interested data and image format to be saved [7], as well as the full path to save files and the save frequency and save counter, etc. Push `Start SAVE` to trig the auto save procedure, the saving status could be found in the highlighted status bar in `imageviewer` app.



Warning: When utilizing `Auto save` operation, remember `imageviewer` should be in the DAQ mode, i.e. `START` button should be pushed.

Data Example

The data files generated by `imageviewer` can be classified into two categories, i.e. *figure* and *data*, while *figure* could be saved as `png`, `jpeg`, `svg`, `tiff`, `ps`, `eps`, `pdf`, etc.; *data* could be saved as `txt` and `hdf5`, of which only `hdf5` could be self-explained, it is recommanded to use `hdf5`.

`hdf5` files could be generated in the following scenarios:

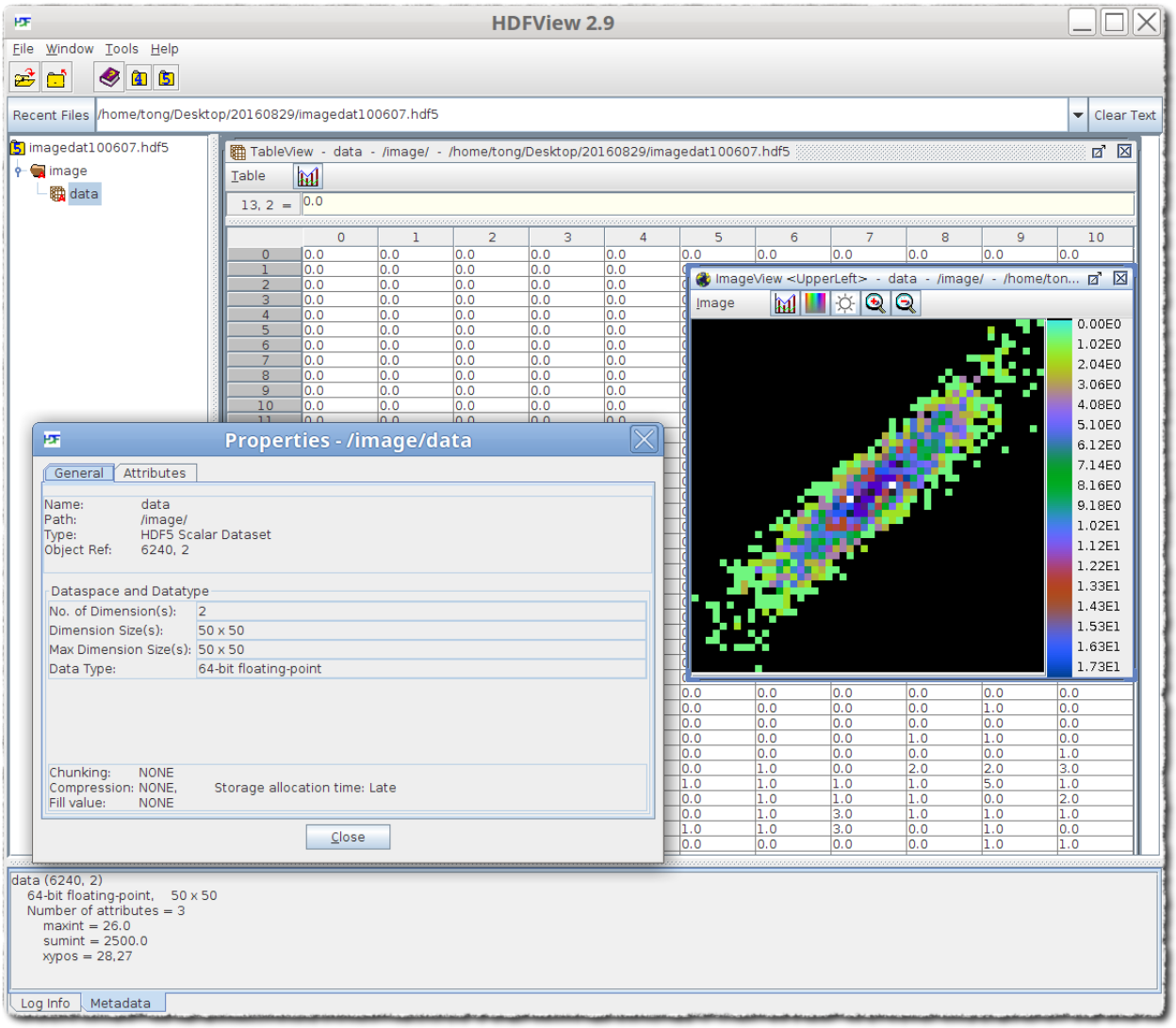
1. Image Data

Click `File > Save Image Data`, or by shortcut `CTRL+D`, raw data of the figure with trivial properties will be saved into a `hdf5` file, by default, the full path filename will be `save root directory` + `<folder name like`

[7] Since saving data, especially `hdf5` is always more efficient that saving other format, it is suggested that only tick `hdf5` options when cares more about efficient, especially for image with large pixles.

`'20160830'` + `save filename head` + `<time stamp (hour,min,sec) like '100607'>` [8].

Any `hdf5` file could be opened by a program that distributed by HDF5 official group, named `hdfview` [9]. Also, in the CLI, command `h5ls` could be used for inspection, below are the illustrations for the two approaches:



[8] All these meta information could be set and get in the `Preferences > Image` panel.

[9] https://www.hdfgroup.org/products/java/hdfview/

```
[tong@walkstation 20160829]$ h5ls -lvr imagedat100607.hdf5
Opened "imagedat100607.hdf5" with sec2 driver.
/                        Group
    Location:  1:96
    Links:     1
/image                   Group
    Attribute: app scalar
        Type:      variable-length null-terminated ASCII string
        Data:  "imageviewer"
    Attribute: timestamp scalar
        Type:      variable-length null-terminated ASCII string
        Data:  "2016-08-30 10:06:07 CST"
    Location:  1:800
    Links:     1
/image/data              Dataset {50/50, 50/50}
    Attribute: maxint scalar
        Type:      native double
        Data:  26
    Attribute: sumint scalar
        Type:      native double
        Data:  2500
    Attribute: xypos {2}
        Type:      native long
        Data:  28, 27
    Location:  1:6240
    Links:     1
    Storage:   20000 logical bytes, 20000 allocated bytes, 100.00% utilization
    Type:      native double
```

2. Curve Fitting Data

When curve fitting is enabled, in the pop up fitting panel, `Export` button will generate the `hdf5` data, saving path should be informed through the pop up dialog. Use `h5ls` to inspect the exported file:

```
$ h5ls -lvr fitdata104332.hdf5
Opened "fitdata104332.hdf5" with sec2 driver.
/                        Group
    Location:  1:96
    Links:     1
/data                    Group
    Attribute: timestamp scalar
        Type:      variable-length null-terminated ASCII string
        Data:  "2016-08-30 10:43:32 CST"
    Location:  1:800
    Links:     1
/data/fit                Group
    Attribute: x0 scalar
        Type:      native double
        Data:  26.041
    Attribute: xstd scalar
        Type:      native double
        Data:  6.99682
    Attribute: y0 scalar
        Type:      native double
        Data:  26.041
    Attribute: ystd scalar
        Type:      native double
        Data:  6.99682
    Location:  1:10608
    Links:     1
/data/fit/x              Dataset {200/200}
    Location:  1:14432
    Links:     1
```

```
    Storage:   1600 logical bytes, 1600 allocated bytes, 100.00% utilization
    Type:      native double
/data/fit/xdata          Dataset {200/200}
    Location:  1:14976
    Links:     1
    Storage:   1600 logical bytes, 1600 allocated bytes, 100.00% utilization
    Type:      native double
/data/fit/y              Dataset {200/200}
    Location:  1:11624
    Links:     1
    Storage:   1600 logical bytes, 1600 allocated bytes, 100.00% utilization
    Type:      native double
/data/fit/ydata          Dataset {200/200}
    Location:  1:14704
    Links:     1
    Storage:   1600 logical bytes, 1600 allocated bytes, 100.00% utilization
    Type:      native double
/data/raw                Group
    Location:  1:1960
    Links:     1
/data/raw/x              Dataset {50/50}
    Location:  1:7712
    Links:     1
    Storage:   400 logical bytes, 400 allocated bytes, 100.00% utilization
    Type:      native long
/data/raw/xdata          Dataset {50/50}
    Location:  1:10336
    Links:     1
    Storage:   400 logical bytes, 400 allocated bytes, 100.00% utilization
    Type:      native double
/data/raw/y              Dataset {50/50}
    Location:  1:6240
    Links:     1
    Storage:   400 logical bytes, 400 allocated bytes, 100.00% utilization
    Type:      native long
/data/raw/ydata          Dataset {50/50}
    Location:  1:7984
    Links:     1
    Storage:   400 logical bytes, 400 allocated bytes, 100.00% utilization
    Type:      native double
```

The grouped data set: `raw` and `fit`, other metadata like `timestamp`, central point coordinate (`x0`, `y0`), radius size (`xstd`, `ystd`) could be retrieved according to the string names.

3. Auto Saved Data

Data generated by `Auto save` operation could be `hdf5` (recommanded the exclusive choice), is similar as `Image Data`, besides `timestamp` property, but with only one property: `app`, which value is `imageviewer`, indicating that the file is generated by app named `imageviewer`. These `hdf5` data files could be imported into `dataworkshop` for post-processing, see *Example 2*.

Example 2

`dataworkshop` – Data post-processing app.

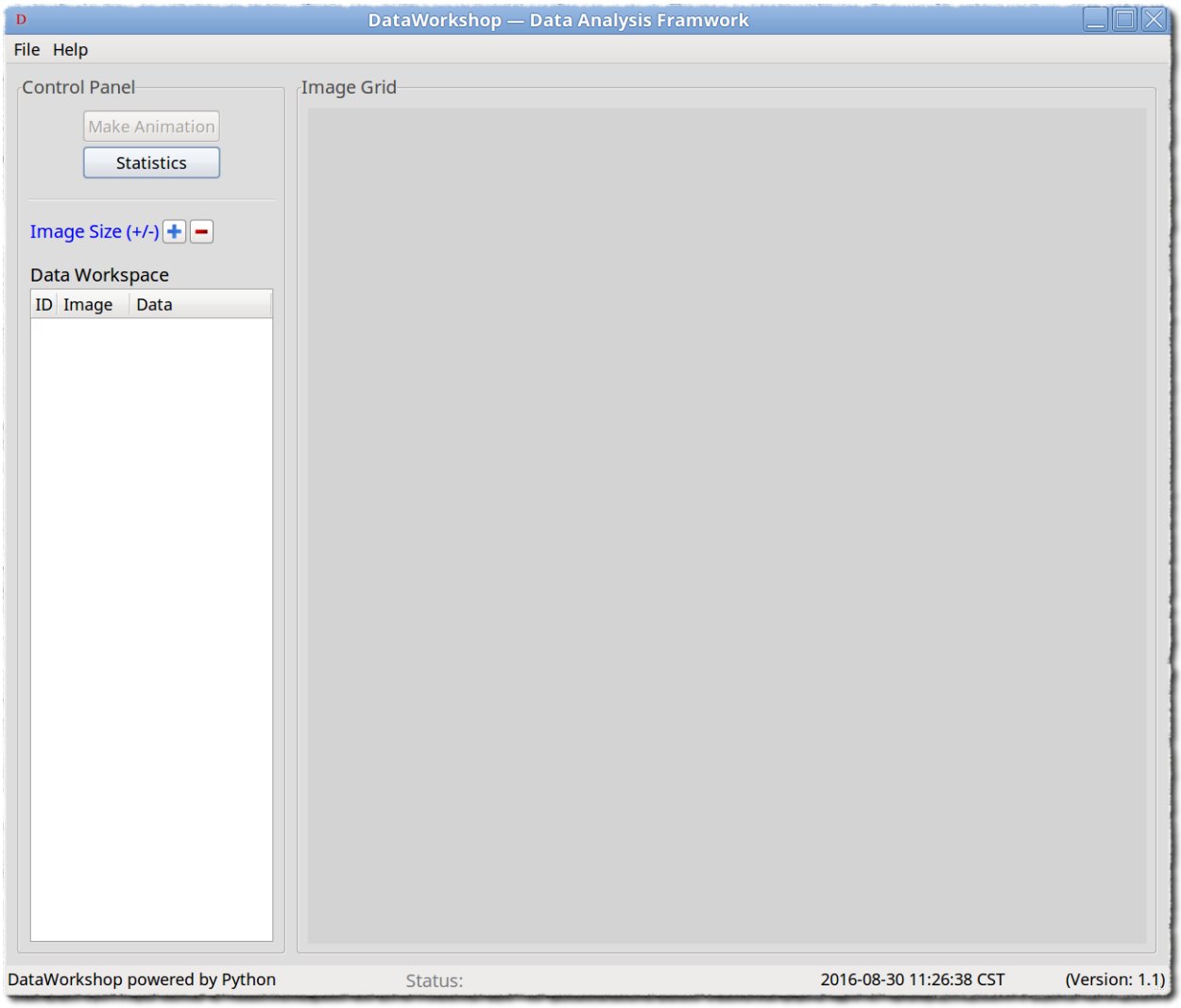
Introduction

Initially, `dataworkshop` is created for the data processing tasks, since not all of the tasks, especially those demanding heavy computing resources are not required to be processed simultaneously, so we first get all the data that may need, then filter/process by another app, that is `dataworkshop`'s job.

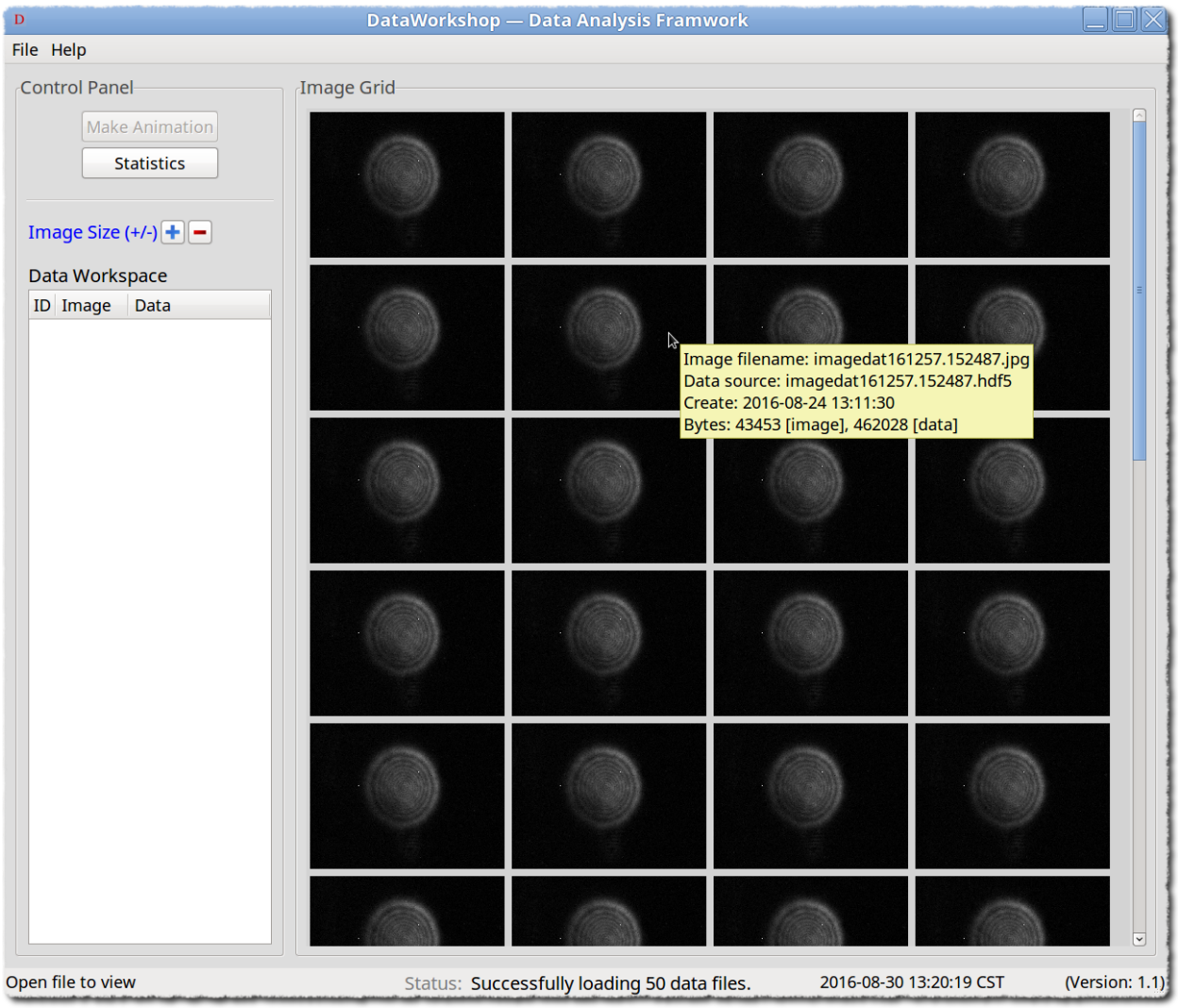
Usage Guide

Import Data Files

Open `dataworkshop` as the following image shows, `Image Grid` panel is the place where imported data file generated image should be put into, in the `Control Panel`, operations like `Make Animation` (not enabled), `Statistics` will take the data files that selected into `Data Workspace` as the input, to do specific post-processing procedures.



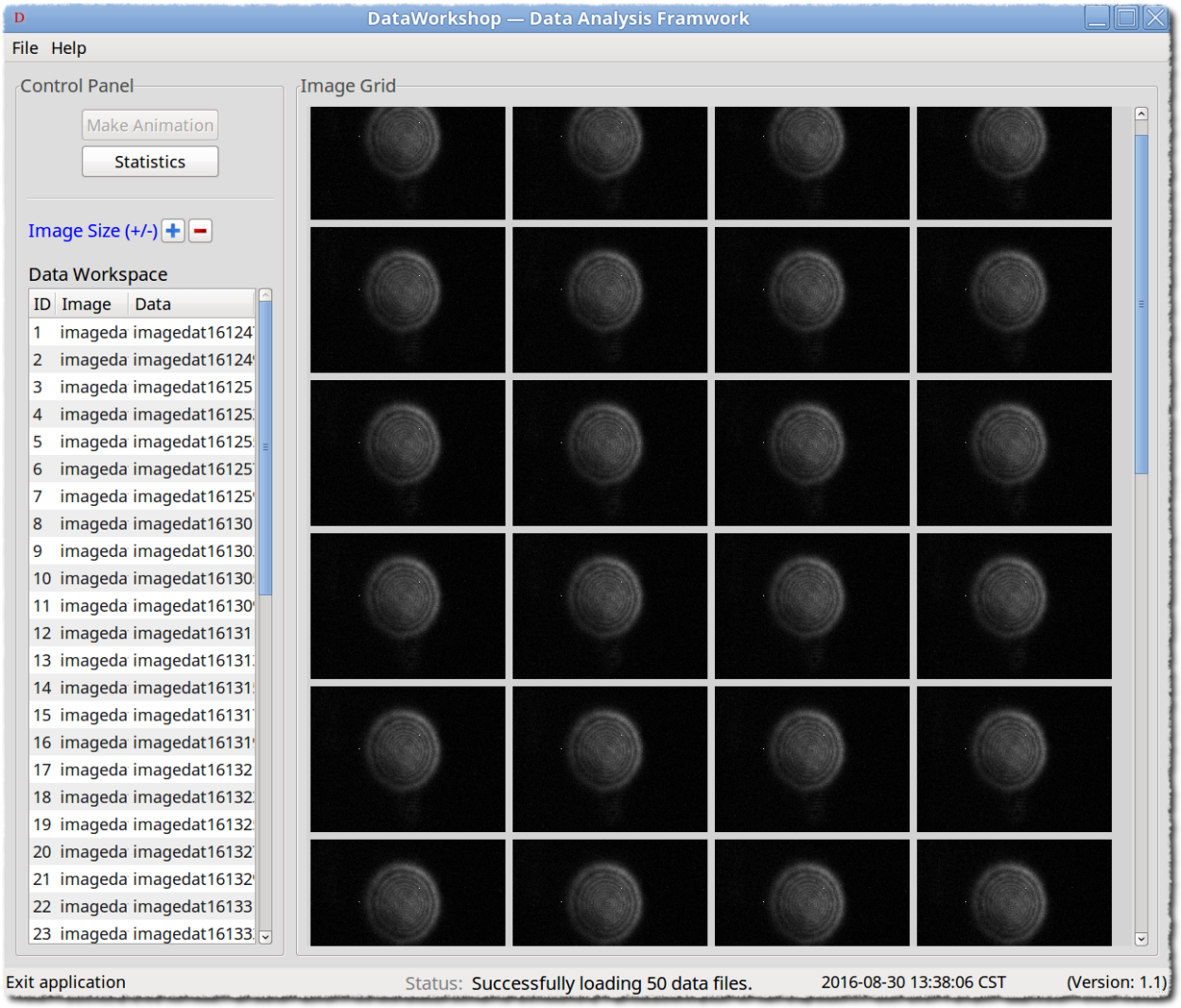
For example, click `File > Open files` or shortcut `CTRL+O` to import `hdf5` files that generated by `Auto save` operations of `imageviewer` app. In order to show the data files in a more informative way, the import/open operation will first transform the data files into images, then show in grid style, the image size could be adjusted by the `Image Size(+/-)` buttons, i.e. increase (+) or shrink (-); when the mouse is hovering over the image, information will show as a tooltip, including metadata like filename, timestamp, filesize, etc.



Note: `File > Add files (CTRL+A)` can be used to continue to add data files into the current `Image Grid`.

There are two approaches to add/remove data files into `Data Workspace`:

1. Right click the image files in `Image Grid`, choose `To Workspace` menu item to add the selected file into `Data Workspace`;
2. Right click the blank space of `Image Grid` panel, i.e. any space that is not occupied by the images, `Add all to workspace` and `Remove all to workspace` actions could be applied;
3. Once the file is already added into `Data Workspace`, then right click on the image should pop up `Out Workspace` action which would remove added file out of `Data Workspace`.

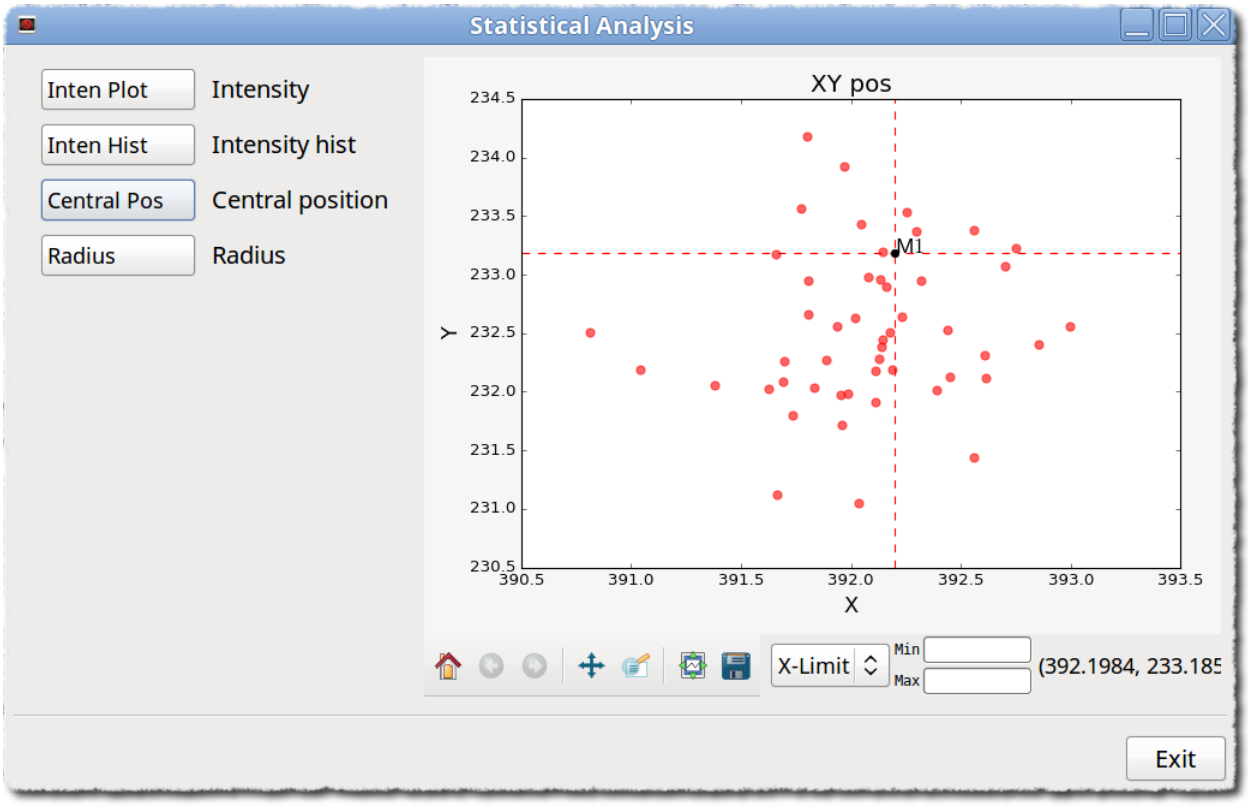


Note: When right clicking the image, another available action is `Pop up Analysis`, which will show the selected image in a more vivid way, details see *Example 6*.

Post-processing

All the operations to the data files should be implemented in the `Control Panel`, up to now, there is `Statistics` button available to do post-processing operations.

Push `Statistics` button to open the pop up window, in which, push `Radius`, will plot the distribution of the central positions of all the files that in `Data Workspace`, other mathods also could be developed to help the user to quickly understand the data, so as to make the next step decision.



Example 3

`cornalyzer` – Parameter correlation analysis app.

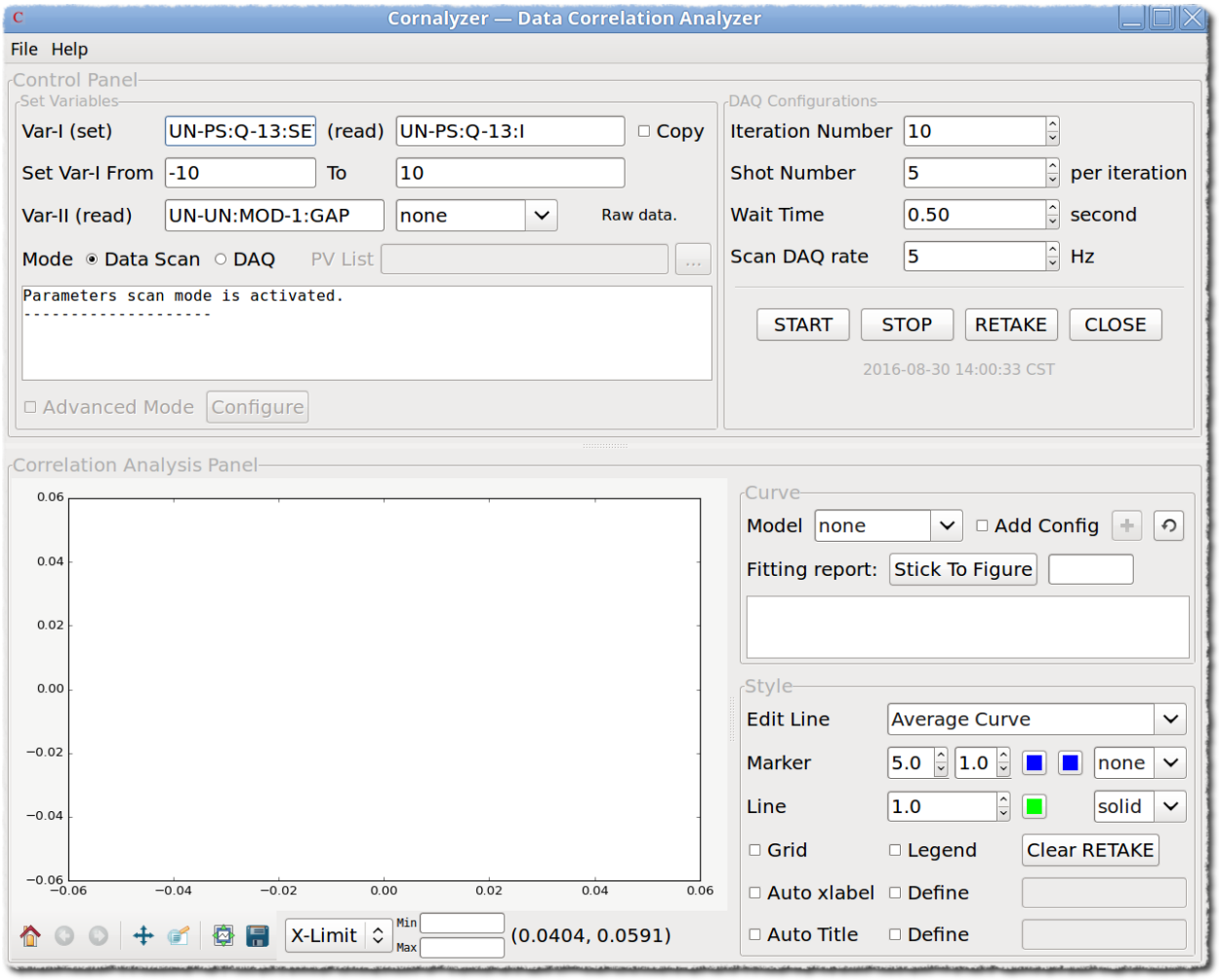
Introduction

If the user wanna analysis the correlation between parameters, `cornalyzer` could help. Presently, main features that `cornalyzer` provides are:

- **Two dimensional parameters correlation analysis;**
  - post-processor could be applied to the second dimension
- Data plot with x/y errorbars;
- **Curve fitting, support both polynomial and gaussian;**
  - additional fitting paramters
- **Plenty of operations on the figure:**
  - Line/marker/errorbar style
  - Labels, title, legend, text of fitted information
  - Data range control
- Re-take the selected variable points for correction by `RETAKE` button;
- Save the analysis result.

Usage Guide

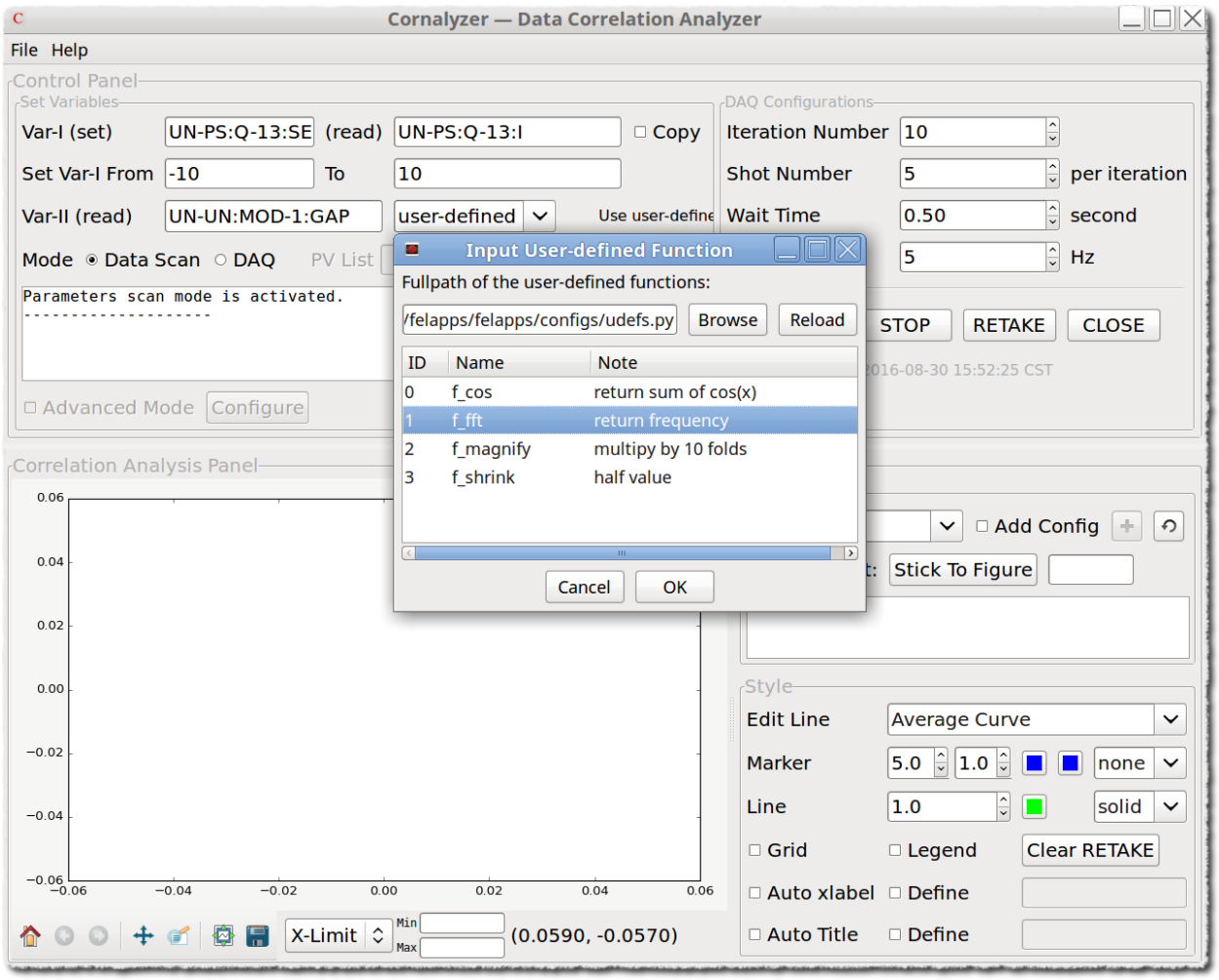
Open the `cornalyzer` app, like the following image shows, currently only `Data Scan` mode is supported.



Data Scan Guide

Input the EPICS PV string names in the `Set Variables` panel, `Var-I` should be treated as the first dimension of the analysis, i.e. the paramter to be altered, since the set and read of `Var-I` usually are controlled by two different PV strings, so two input boxes should be filled correctly [1]. The `Var-II` is the parameter that to be monitored as `Var-I` is tunning, post-processing operations could be applied to the raw data that read from the EPICS PV string of `Var-II`, click the combobox right next the input box of `Var-II`, choice could be one of `none`, `sum`, `max`, `min` and `user-defined`, the meaning could be found on the right side hint text.

[1] copy checkbox could be used to quickly copy `set` value into `read` field.



Click `user-defined` option will pop up a window, browse the Python module that contains user-defined functions to load [2], below is the example of user-defined Python module:

```
"""
user-defined functions to process A, where
A could be any type of:

    1. array, 1D or 2D,
    3. scalar, float, integer,

Usually, the raw value of A could be retrieved by caget(PV_string)

This script/module is potentially be used by 'cornalyzer',
i.e. correlation analyzer application, to apply post-processing
procedure on the scan dependent variable.

Created: 2016-07-14 14:59:11 PM CST
"""

import numpy as np

def f_fft(x):
    """ return frequency
    """
    f = abs(np.fft.fft(x))
    idx = np.where(f==f.max())[0][0]
    return idx

def f_magnify(x):
```

[2] The Python module file is a normal Python script that has funtion definitions, it is the user's liability to manage this file.

```
    """ multipy by 10 folds
    """
    a = 10
    return x * a

def f_shrink(x):
    """ half value
    """
    a = 0.5
    return x * a

def f_cos(x):
    """ return sum of cos(x)
    """
    return np.sum(np.cos(x))
```

Choose the function to be applied on `Var-II`, push `OK` to return to the main window.

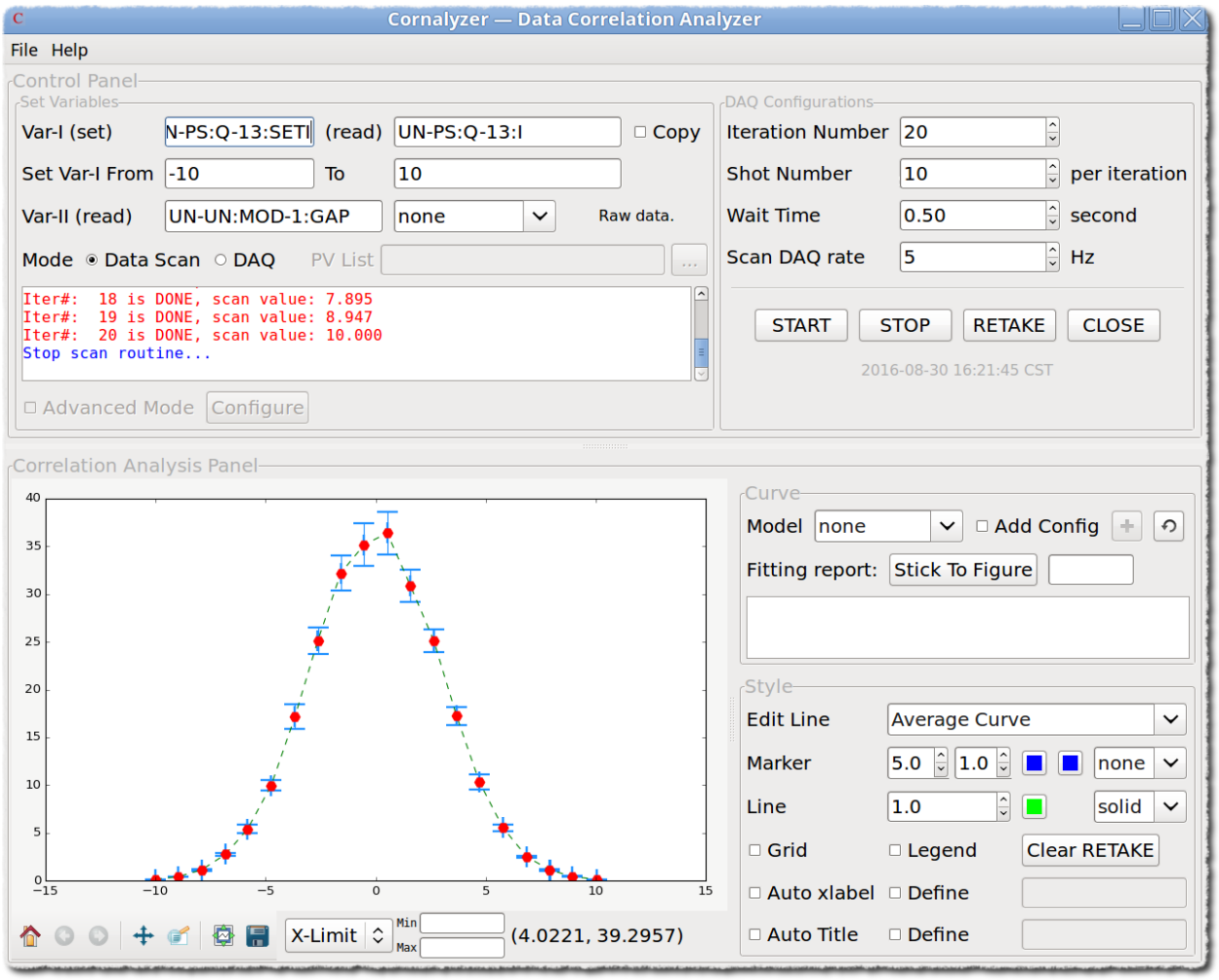
Note: Tips to write the user-defined functions:

1. Naming the function with meaningful strings;
2. Keep function as simple as possible;
3. Always write *doc string* (strings wrapped by `"""`), which will appear as the `Note` field to help understanding.

The meanings of other scan parameters could be get hint from the tool tips when the mouse hovers over the controls, e.g. `Iteration Number` is the total counter of `Var-I` scan array (equally spaced) that to be altered, `Shot Number` is the total counter of the fetched data for every iteration, and `Scan DAQ rate` is the DAQ frequency, e.g. scan configurations like: `Iteration = 20`, `Shot Number = 10`, `Wait Time = 0.5` and `Scan DAQ rate = 5 Hz` tell the app that alter the `Var-I` parameter from the min (`-10`) to max (`10`) with equal step, until the step counter reaches `20`, and for each variable update procedure, the global timer should wait `0.5` second [3], in each step, the DAQ system should be working to retrieve the data (`Var-II` with proper operation) with the speed of `0.2` second per shot, until get `10` shot in total, this is one iteration.

Click `START` to start the scan procedure, and `STOP` to stop any time, scan log will show in the textbox, beginning with `Iter#...` for each iteration.

[3] The data that used by example shown here is provided by a soft-IOC, with the scan frequency of 5 Hz, while for the real machine, e.g. Dalian Coherent Light Source (DCLS), tests show that 10-20 second may require so as to get valid signals.



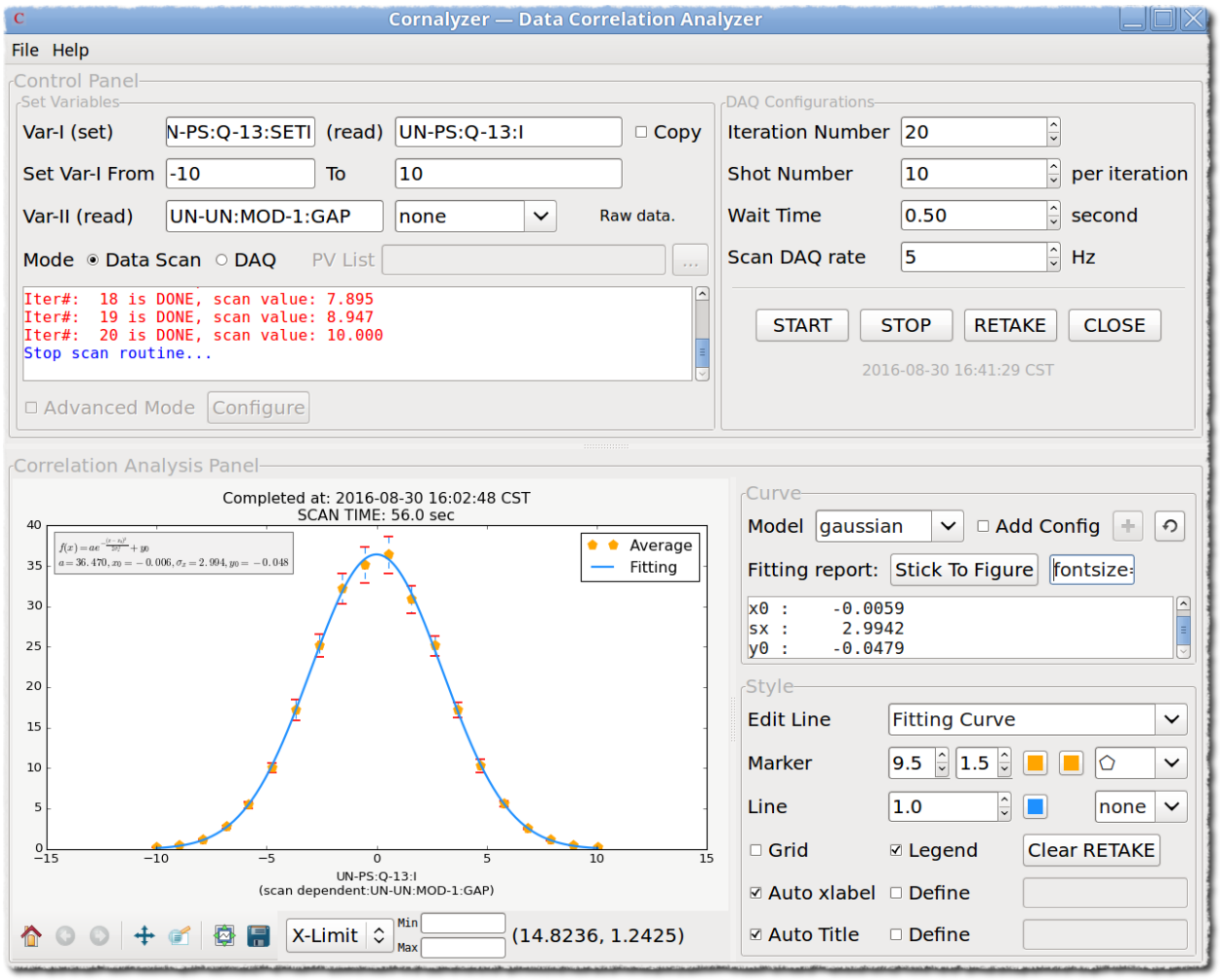
Curve Plot Tunning

`cornalyzer` is designed with the intention of generating publish-quality artwork for the data sharing and distributing.

Curve Fitting

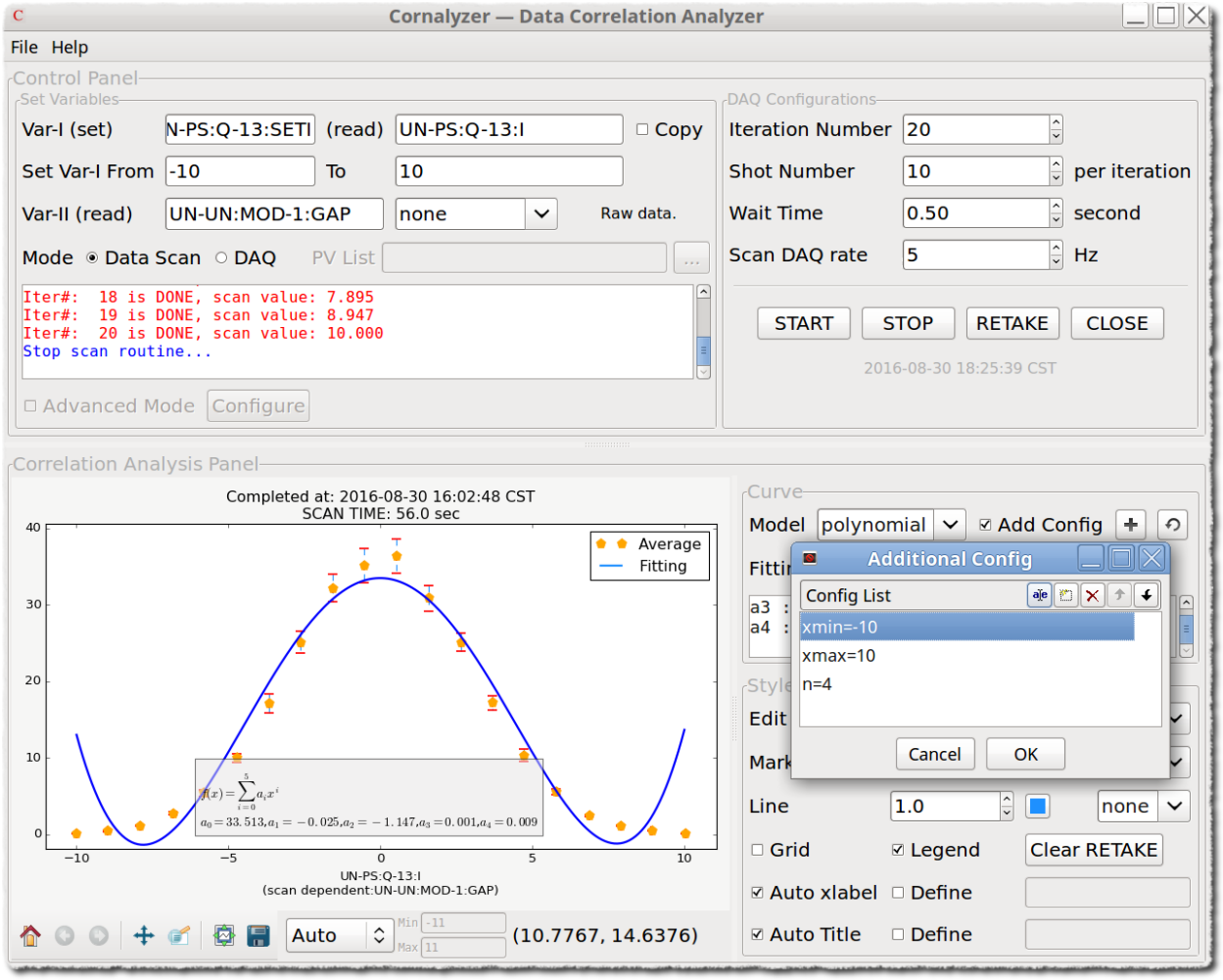
When the scan procedure is done, pop up dialog will show the warning that the scanned data could be saved by `CTRL+S`, while it is recommanded that trig saving action when proper fitting model be applied.

In the `Curve` panel, choose `Model` to apply proper fitting model, e.g. `gaussian`, `polynomial`, the option `none` means not fitting the curve. In this example `gaussian` should be selected, blue gaussian shaped line (fitted average every iteration line) will be drawn onto the figure.



Note: Additional parameters are also provided to make the curve fitting module more flexible, follow the steps to do this:

1. Check `Add Config`, click + button to add parameters, just like the following image shows;
2. Valid parameters: `n`, the highest order for the polynomial fitting model; `xmin` and `xmax`, to fix the fitting range along x-axis.



Todo

`user-defined` option should be added into the `Model` combobox, such that to handle more curve fitting scenarios.

Curve Style

To adjust the style of the curves is of significance to make high-quality artworks, fortunately, Python package — `matplotlib` is dedicated to accomplish such mission.

There are three options in `Edit Line` combobox, choose any one of them to apply proper style configurations below, e.g. line style, line color, line thinkness, marker style, marker color, maker size, etc., as well as putting legend, title, labels onto figure, moreover the fitting functions with fitting parameters could be annotated, which supports mouse dragging to reposition [4].

If `Auto xlabel` is checked, the xlabel will be assigned with the PV strings of `Var-I` and `Var-II`, and if the user wanna assign with other string, just check the `Define` option, and write the user-defined xlabel in the

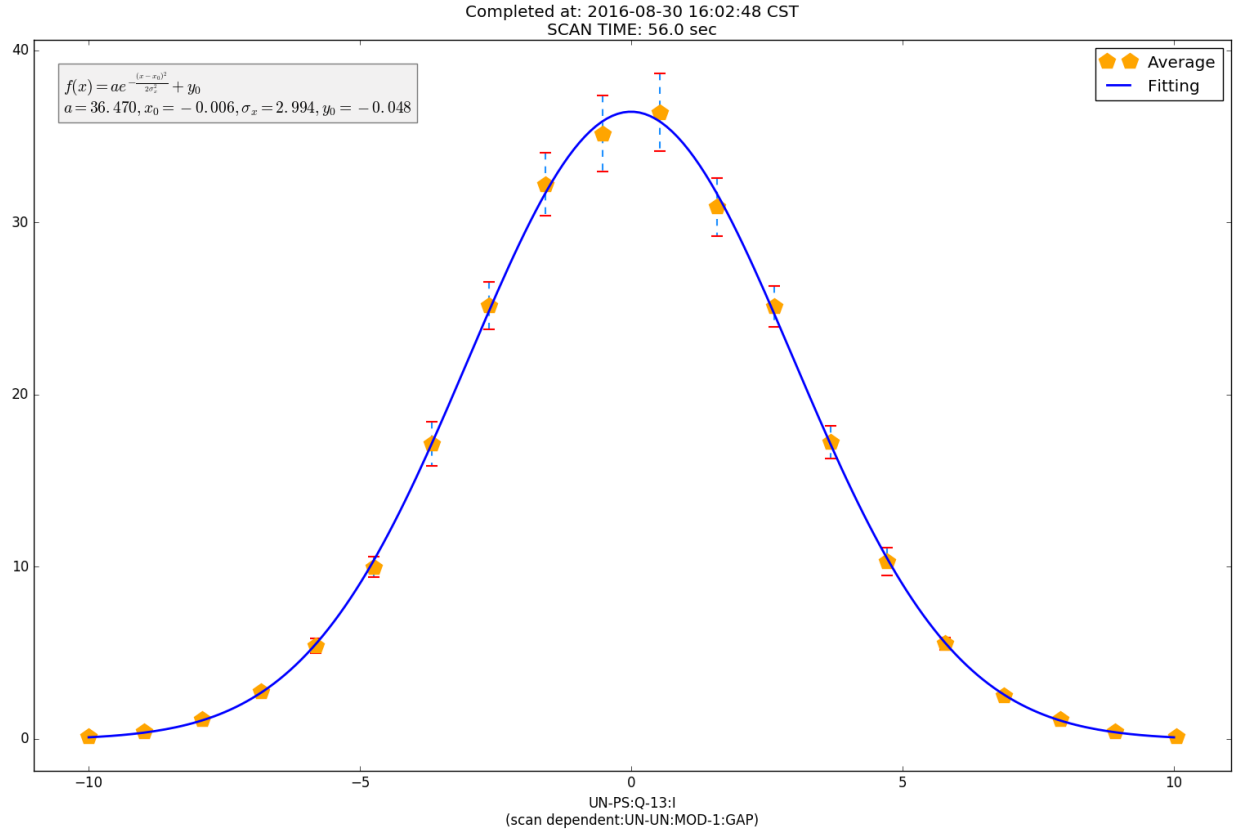
[4] Push `Stick To Figure` button will draw the fitting function onto figure (the grey box in the above image), control the position could be by mouse dragging or input proper parameters, e.g. input `x=0;y=0` in the textbox right aside of `Stick To Figure` button will move the grey box to `(0,0)`, another valid parameter is `fontsize`, which could be used to control the text font size, details could be reached when the mouse hovers over the textbox.

right aside textbox, `Auto xlabel` may be rechecked to show the new xlabel; the same rule applies to the `Auto Title` options [5].

Scan Plot

Main features of the scan plot region:

1. Save figure;
2. x/y data limit tunning;
3. Operations that toolbar provides: *zoom in/out*, *panning*, *reset*, etc.



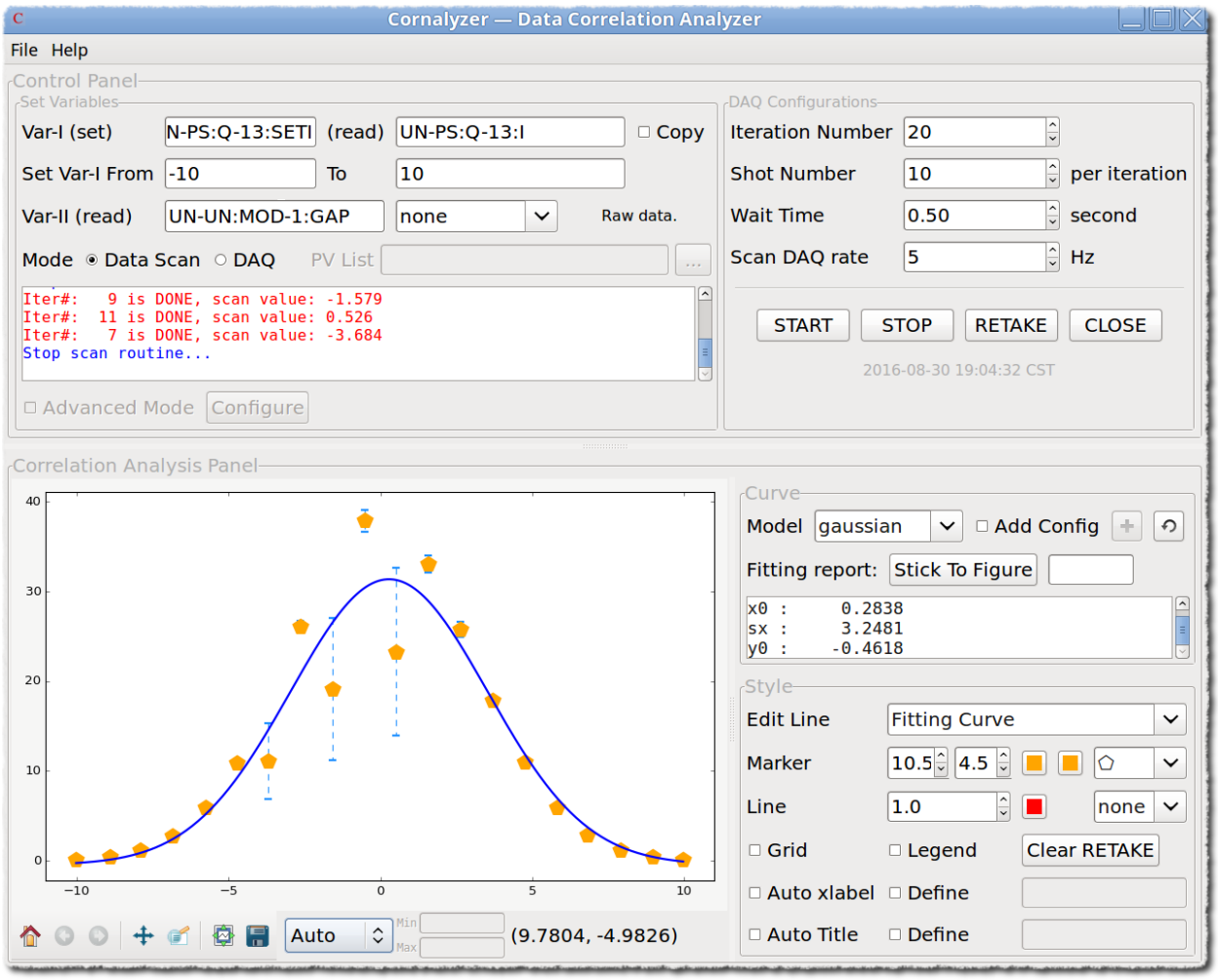
Bonus Feature

Sometimes, the user may demand that some iterations should be processed again, that means the data point should be retaken, no matter what reasons should be relied on, there should be some feature to accomplish such task, then the `RETAKE` button is created.

> **Warning:** Only `RETAKE` when you have every reason that the data point(s) to be replaced is(are) wrong or flawed, e.g. then the machine was not that stable, believed the large jitter was caused by some malfunction of some power sources, etc., In a word, let the SCIENTIFIC DATA be more convinced.

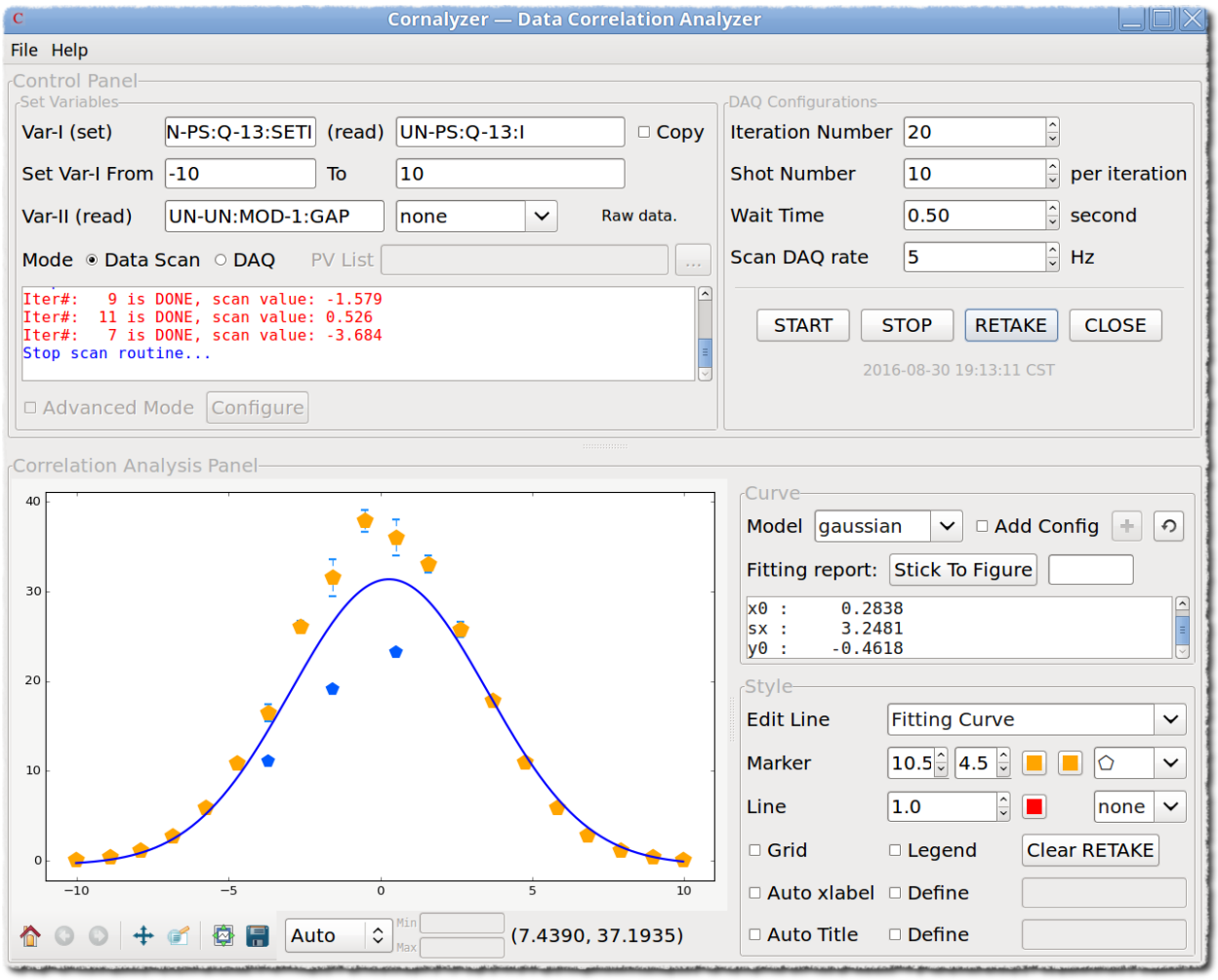
[5] When the `Define` checkbox right aside of `Auto Title` is checked, in the user-defined title string, macro `$TITLE` could be used to substitute the default title string that `Auto Title` uses.

In order to demonstrate the RETAKE feature, the data generator has been changed little, e.g. add more noise, just like the following image shows, three evident points with relatively large errorbar could be treated as wrong points, and need to be fixed [6]. To RETAKE, first select the point(s) to be retaken by clicking mouse onto the point(s), the selected point(s) should be highlighted, then push RETAKE button to start retaken procedure.

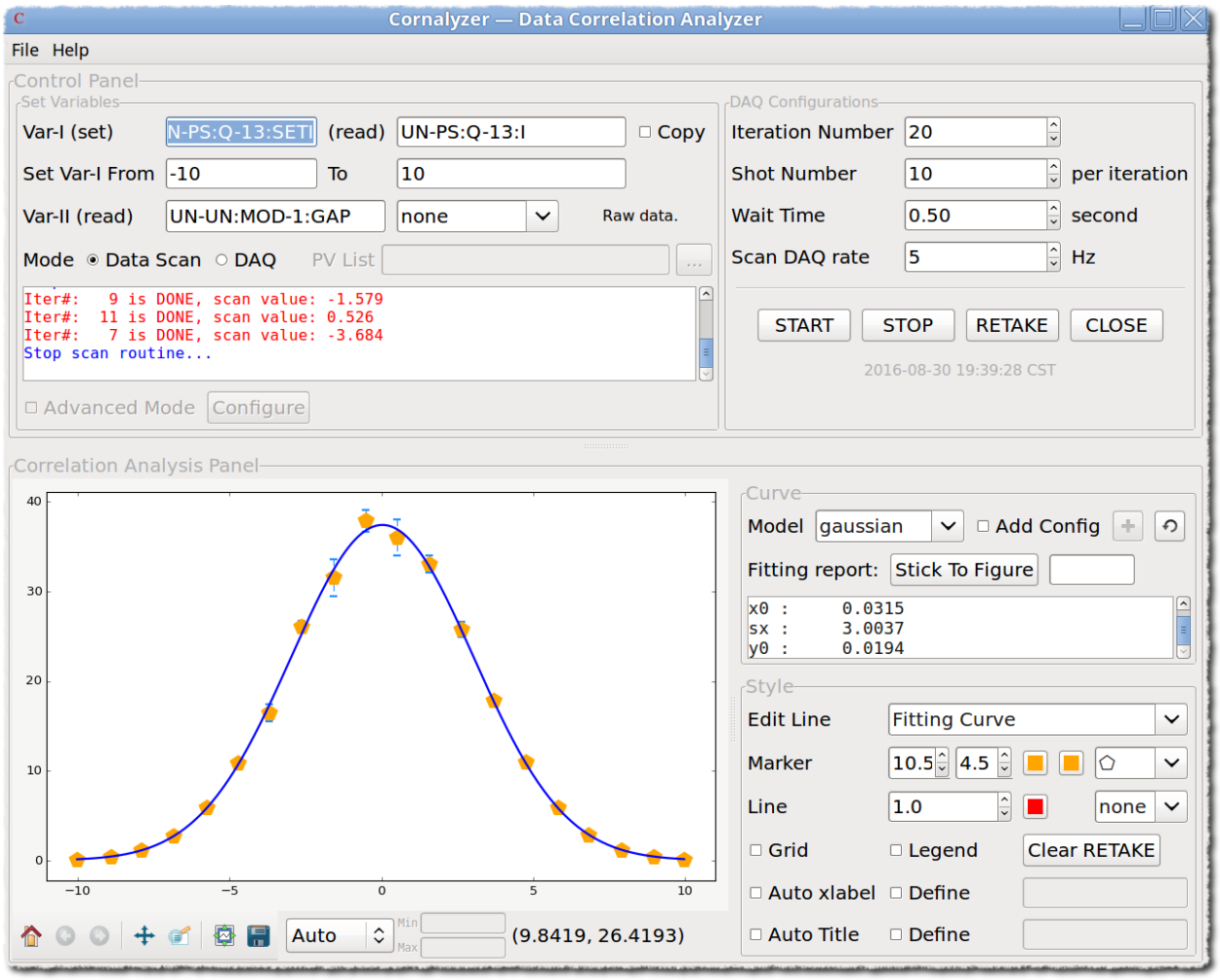


The wrong points have been pushed back.

[6] Just increase the noise level in the data generating process (soft-IOC side) to magnify this effect; decrease noise level before RETAKE is performed, to mimic that machine status is back to stable or acceptable.

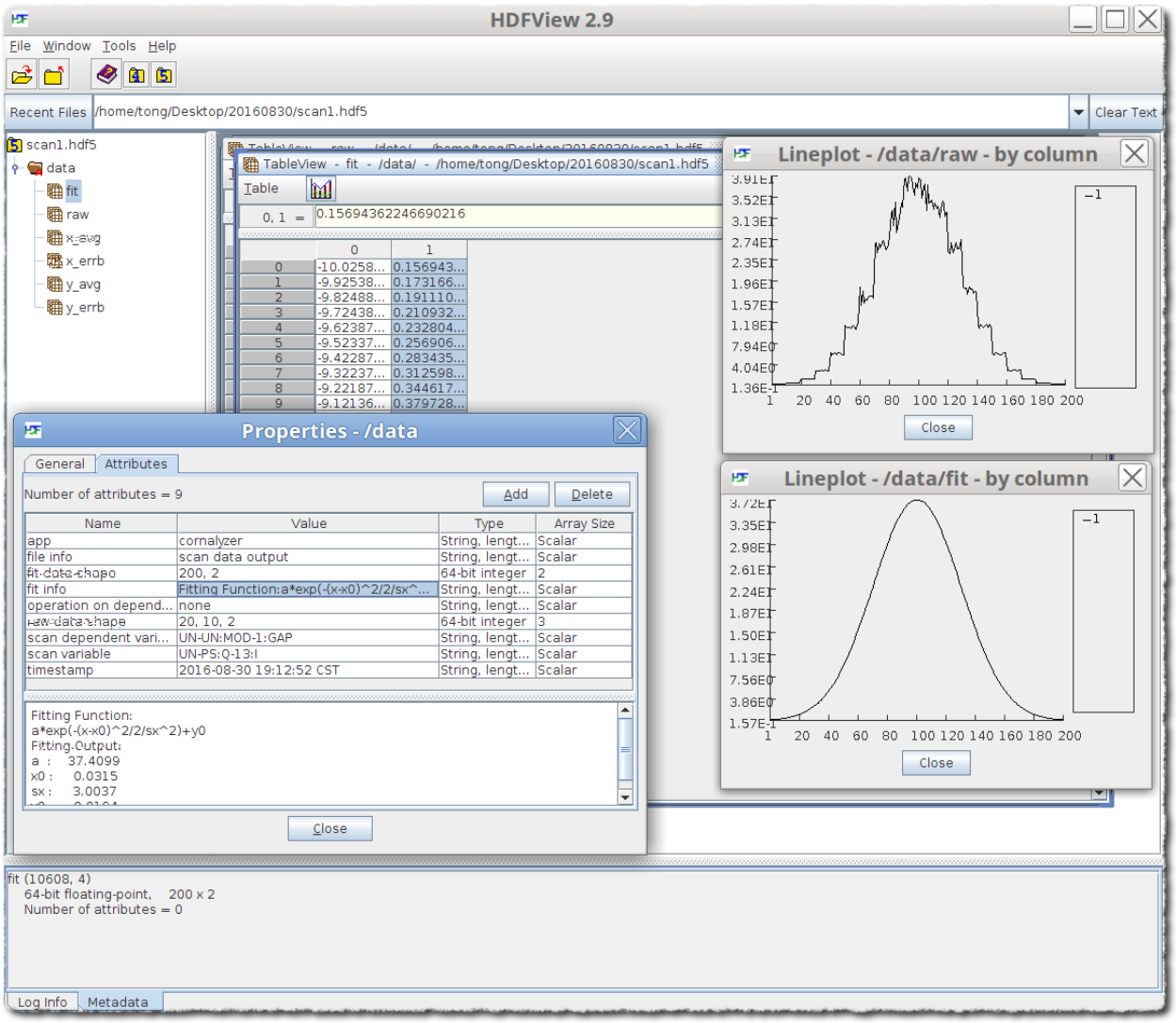


Apply fitting function by refresh button, right aside of +, to get final figure.



Scan Data

CTRL+S could save all the data into defined file, including *raw* and *fit* data, e.g. see below image:



Example 4

`felformula` – Basic FEL physics calculator app.

Introduction

Help users/operators or other non-FEL physics specialists to understand the FEL facility performance under certain machine configurations should be significant, this app is created for such mission that not only providing the staright-forward FEL calculation results, but also showing the clues for the FEL specialists to tunning the machine.

The main features of `felformula` are:

- FEL parameters calculations based on analytical formulae;
- Parameters scan studies, valid parameters:
  - Beam energy, E_b, unit: *MeV*

  - Energy spread, σ_γ
  - Transverse emittance, ϵ_n, unit: m
  - Peak current, I_p, unit: A
  - Undulator period, λ_u, unit: m
  - FEL wavelength, λ_s, unit: m
  - Average beta function, β, unit: m
  - Bunch charge, Q, unit: C
- Interactive scan results figure plot;
- Data saving/exporting and loading/importing.

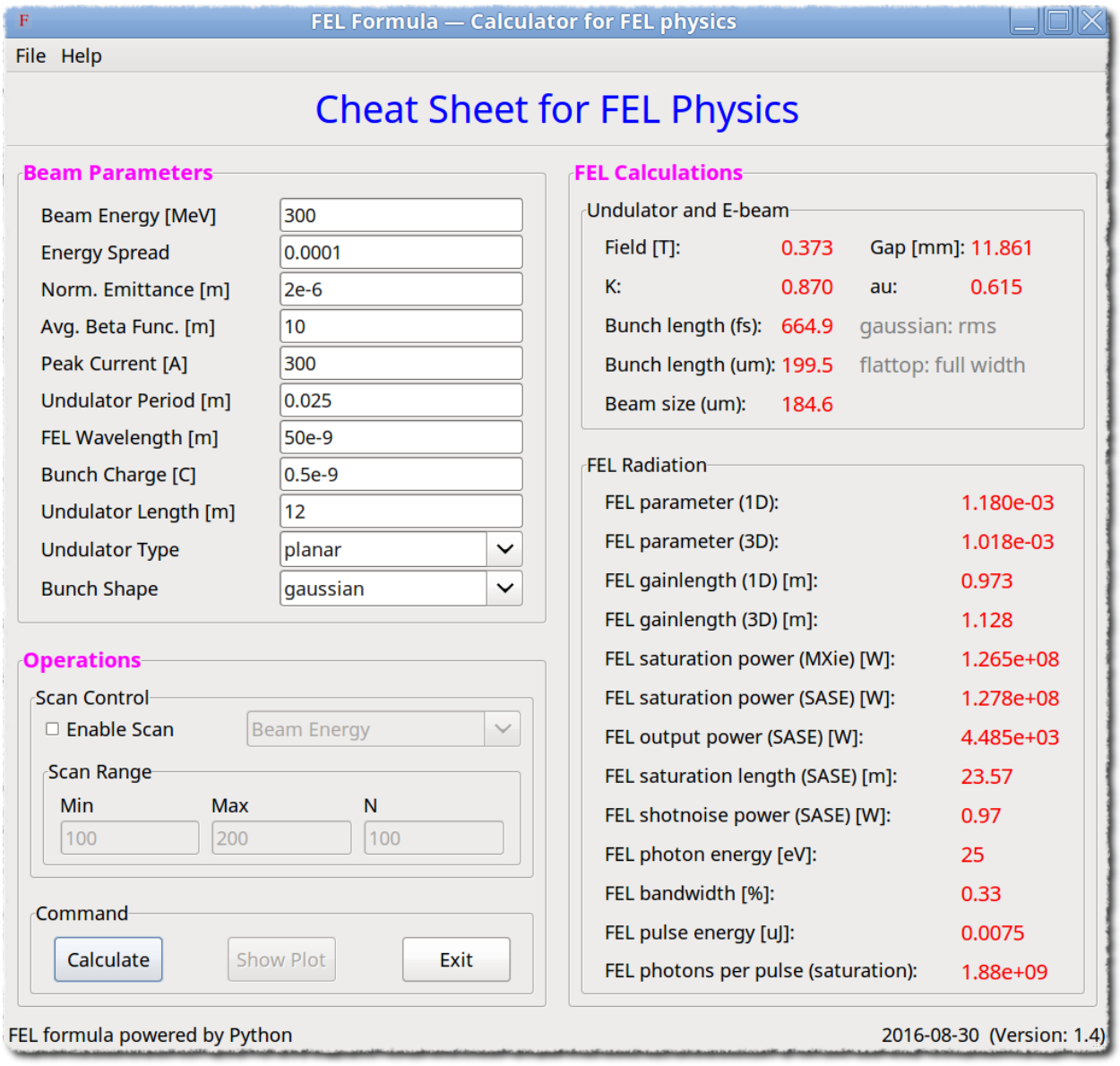
Todo

Comprehensive FEL physics studies could be extended to `felformula`, e.g. integrate numerical simulations into `Advanced` menu (should be created first into the menubar), into which modules could be built to implement various FEL simulation modes, like SASE, HGHG, EEHG, PEHG, and mixtures of them, etc., as well as the algorithms like Twiss-matching, multi-parameters optimization, post-processing for data analyais, etc. Note that every single module should be built with clean and clear APIs, so as to communicate with other modules or apps.

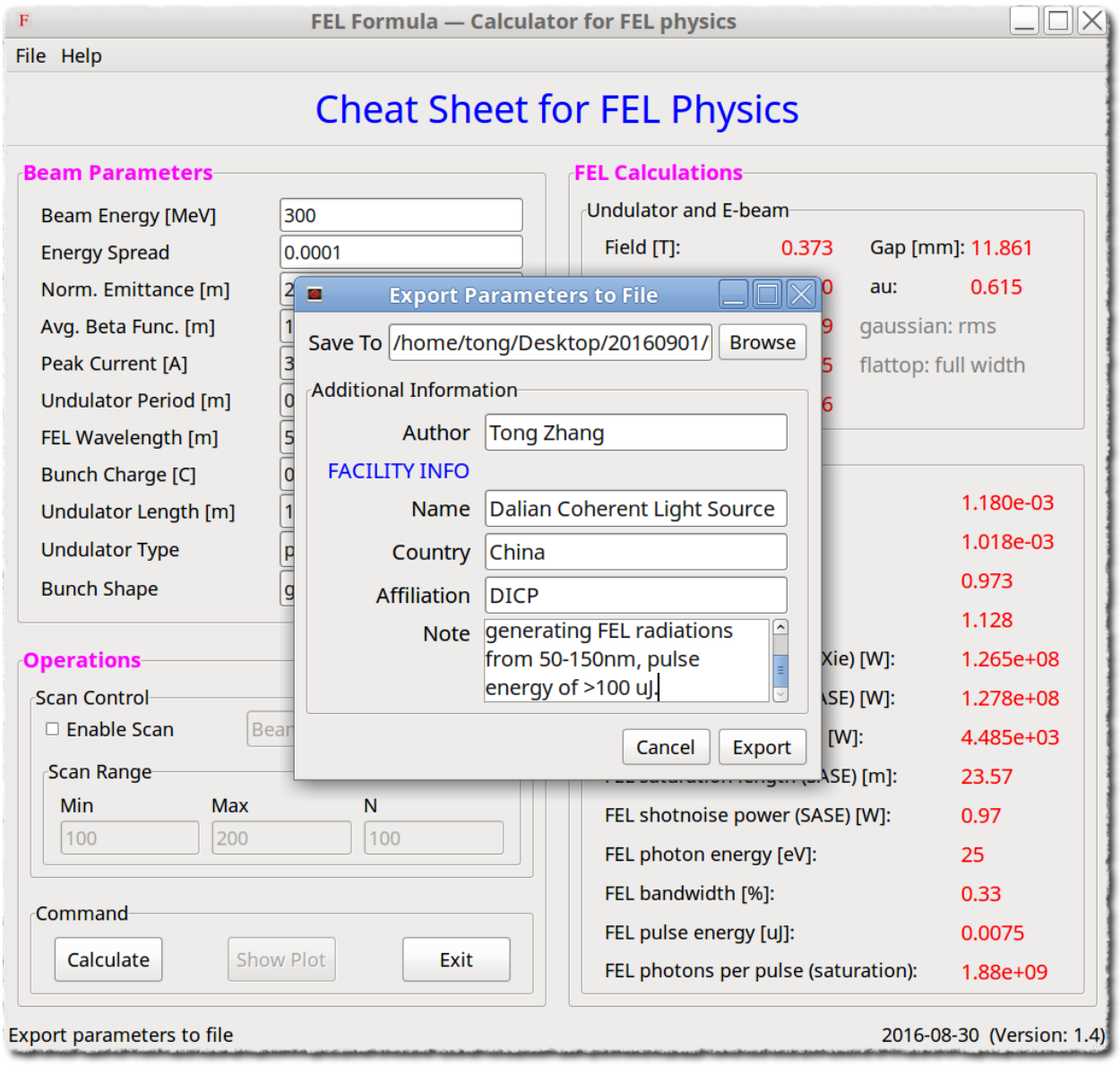
Usage Guide

FEL Calculation

Open `felformula` as the previous examples demonstrated. Fill the `Beam Parameters` panel to setup the main parameters of an FEL facility, e.g. Dalian Coherent Light Source, the physics meaning of every input parameter should be explained by the name itself. Push `Calculate` button in the `Operations > Command` panel to calculate/update the output parameters that listed in `FEL Calculations` panel, where all the available parameters are grouped into `Undulator and E-beam` and `FEL radiation` categories, note that these calculated resutls are only for reference.



All the data could be exported by `File > Export (SHIFT+CTRL+E)`, additional meta information could be appended into saved file, see the following image.



While the exported data file could also be imported by `File > Import (SHIFT+CTRL+I)`, here is the exported file example,

```
[00-info]
author = Tong Zhang
created_time = 2016-09-01 07:20:38 CST
note = note = FEL user facility based on HGHG principle, generating FEL radiations from 50-150nm, pulse energy of >100 uJ.

[01-facility]
affiliation = DICP
country = China
name = Dalian Coherent Light Source

[02-electron_beam]
average_beta_function(m) = 10
bunch_charge(C) = 0.5e-9
bunch_length_t(fs) = 664.9
bunch_length_z(um) = 199.5
bunch_shape = gaussian
central_energy(MeV) = 300
energy_spread = 0.0001
normalized_emittance(m) = 2e-6
peak_current(A) = 300
```

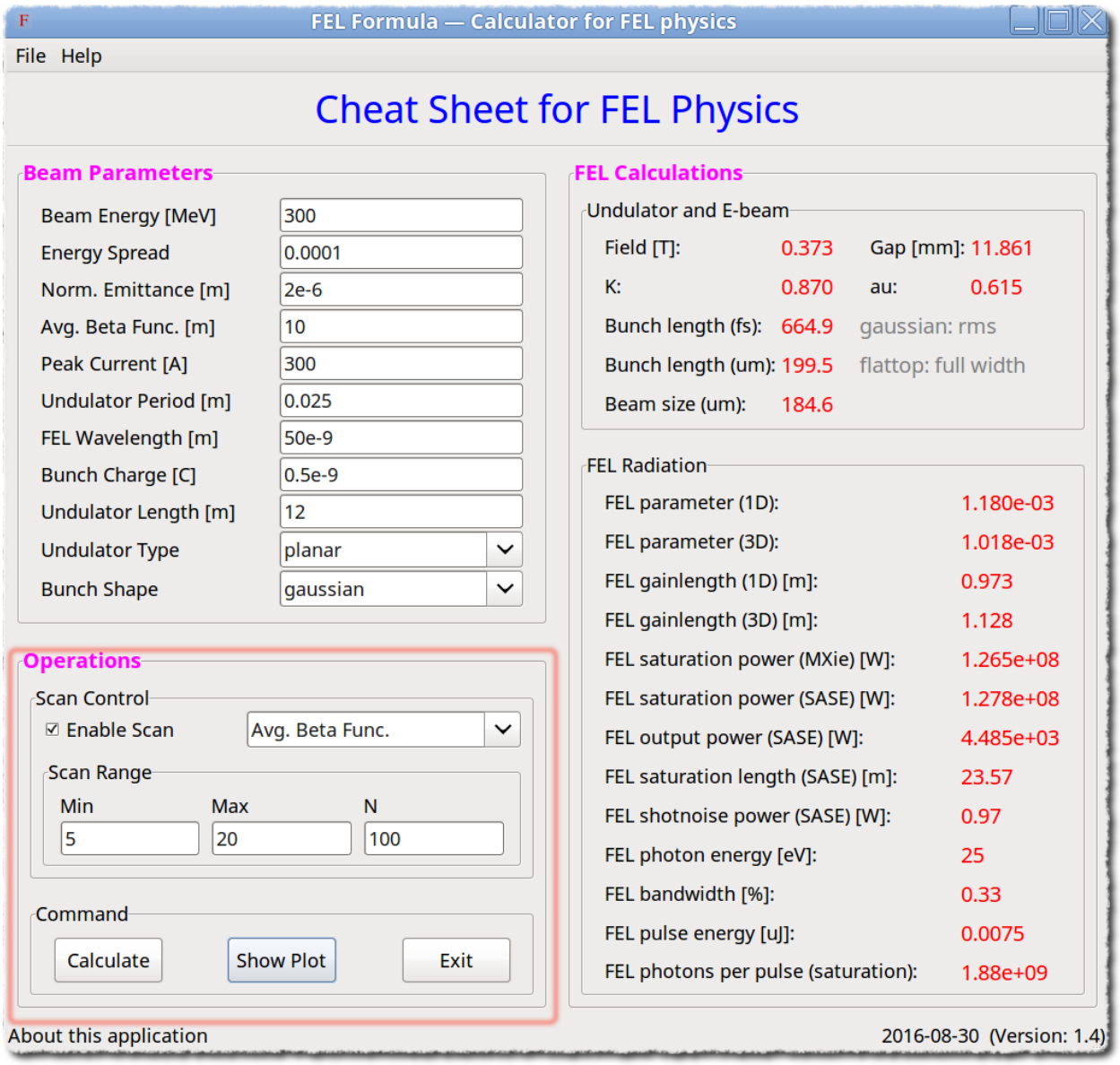
```
transverse_beam_size(m) = 184.6

[03-undulator]
K = 0.870
au = 0.615
gap(mm) = 11.861
peak_field(T) = 0.373
period_length(m) = 0.025
total_length(m) = 12
type = planar

[04-FEL_radiation]
FEL_parameter_1D = 1.180e-03
FEL_parameter_3D = 1.018e-03
bandwidth(%) = 0.33
gainlength_1D(m) = 0.973
gainlength_3D(m) = 1.128
output_power(W) = 4.485e+03
photon_energy(eV) = 25
photon_per_pulse = 1.88e+09
pulse_energy(uJ) = 0.0075
saturation_length_SASE(m) = 23.57
saturation_power_MXie(W) = 1.265e+08
saturation_power_SASE(W) = 1.278e+08
shotnoise_power_SASE(W) = 0.97
wavelength(m) = 50e-9
```

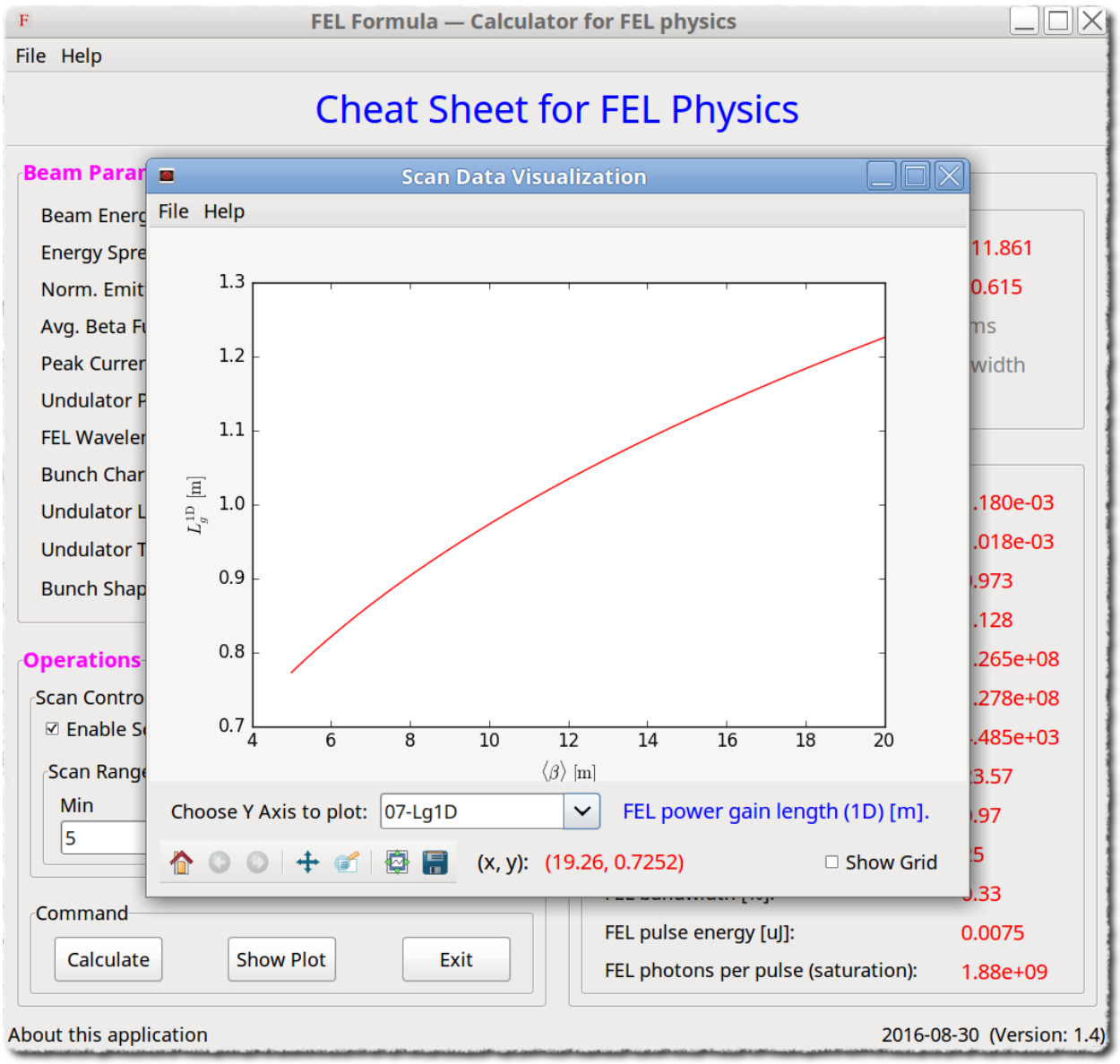
Parameters Scan

Parameters scan feature is supported by checking the `Enable Scan` box, and selecting the parameter to be altered, e.g. average beta function.

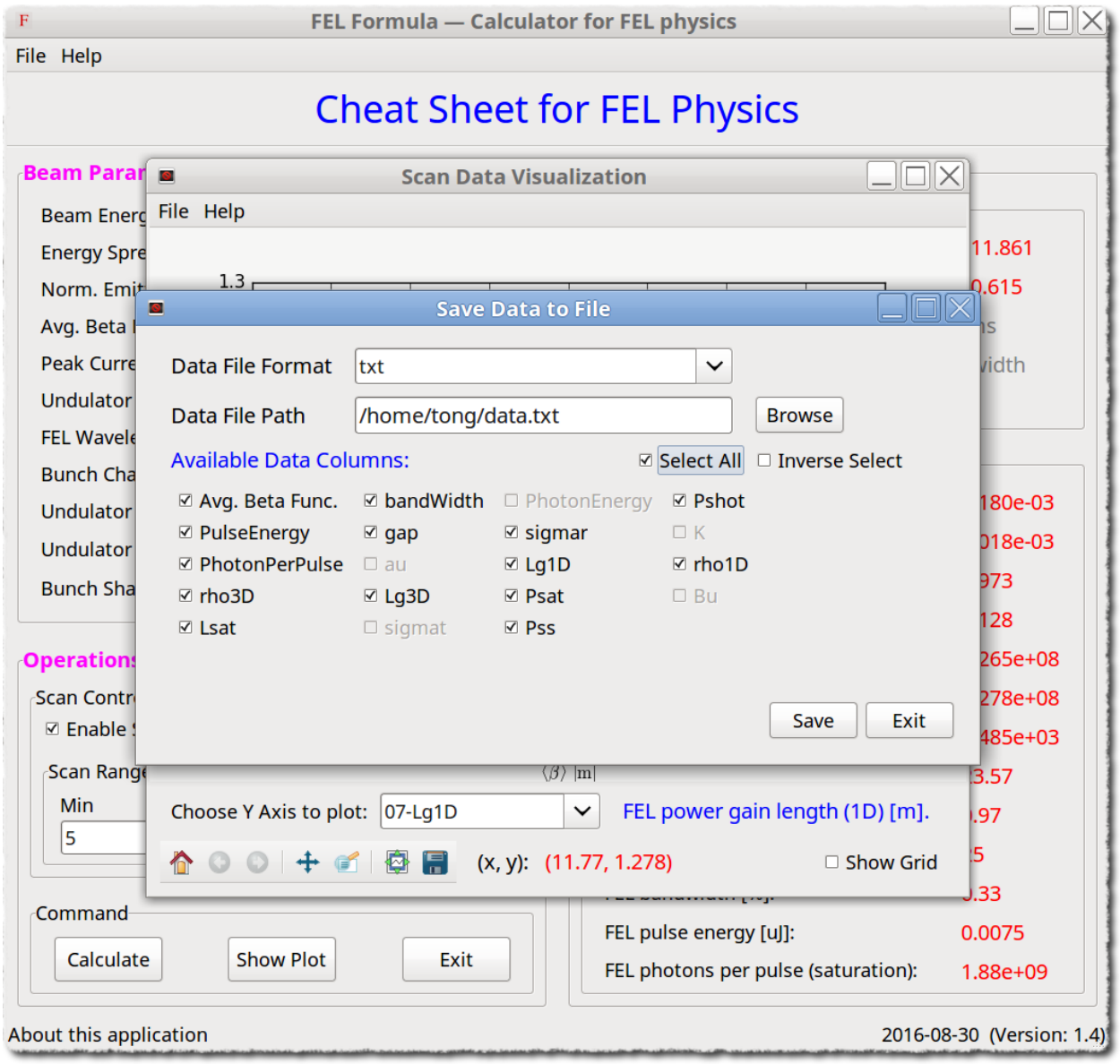


Click `Calculate` first, then `Show Plot` to open the figure plot window, where the data for Y-axis could be seleted from the following table:

Choice	Symbol	Literal explanations
01-au	a_u	Normalized undulator parameter
02-K	K	Undulator parameter
03-Bu	B_u [T]	Undulator magnetic peak field [T]
04-gap	Gap [mm]	Permanent undulator gap [mm]
05-rho1D	ρ^{1D}	FEL parameter or Pierce parameter (1D)
06-rho3D	ρ^{3D}	FEL parameter or Pierce parameter (3D)
07-Lg1D	L_g^{1D} [m]	FEL power gain length (1D) [m]
08-Lg3D	L_g^{3D} [m]	FEL power gain length (3D) [m]
09-Psat	P_{sat} [W]	FEL saturation power (M.Xie formulae) [W]
10-Pshot	P_{shot} [W]	FEL initial shotnoise power [W]
11-Pss	P_{ss} [W]	FEL saturation power (SASE) [W]
12-Lsat	L_{sat} [m]	FEL saturation length (SASE) [m]
13-sigmar	σ_r [μm]	Transverse e-beam radius size (rms) [micro m]
14-sigmat	σ_t [fs]	Temporal bunch length (rms) [fs]
15-bandWidth	$\Delta\lambda/\lambda[\%]$	FEL bandwidth [%]
16-PhotonEnergy	E_p [eV]	FEL photon energy [eV]
17-PulseEnergy	W [μJ]	FEL pulse energy [micro J]
18-PhotonPerPulse	Photon #/pulse	FEL photon number per pulse



The figure will be automatically updated when selected valid parameters.



The data could also be saved by `SHIFT+CTRL+S`, or `CTRL+S` to save the figure. Select the data columns that to be saved, the grey ones are invalid, example file see `here`.

Example 5

`latticeviewer` – Accelerator online modeling app.

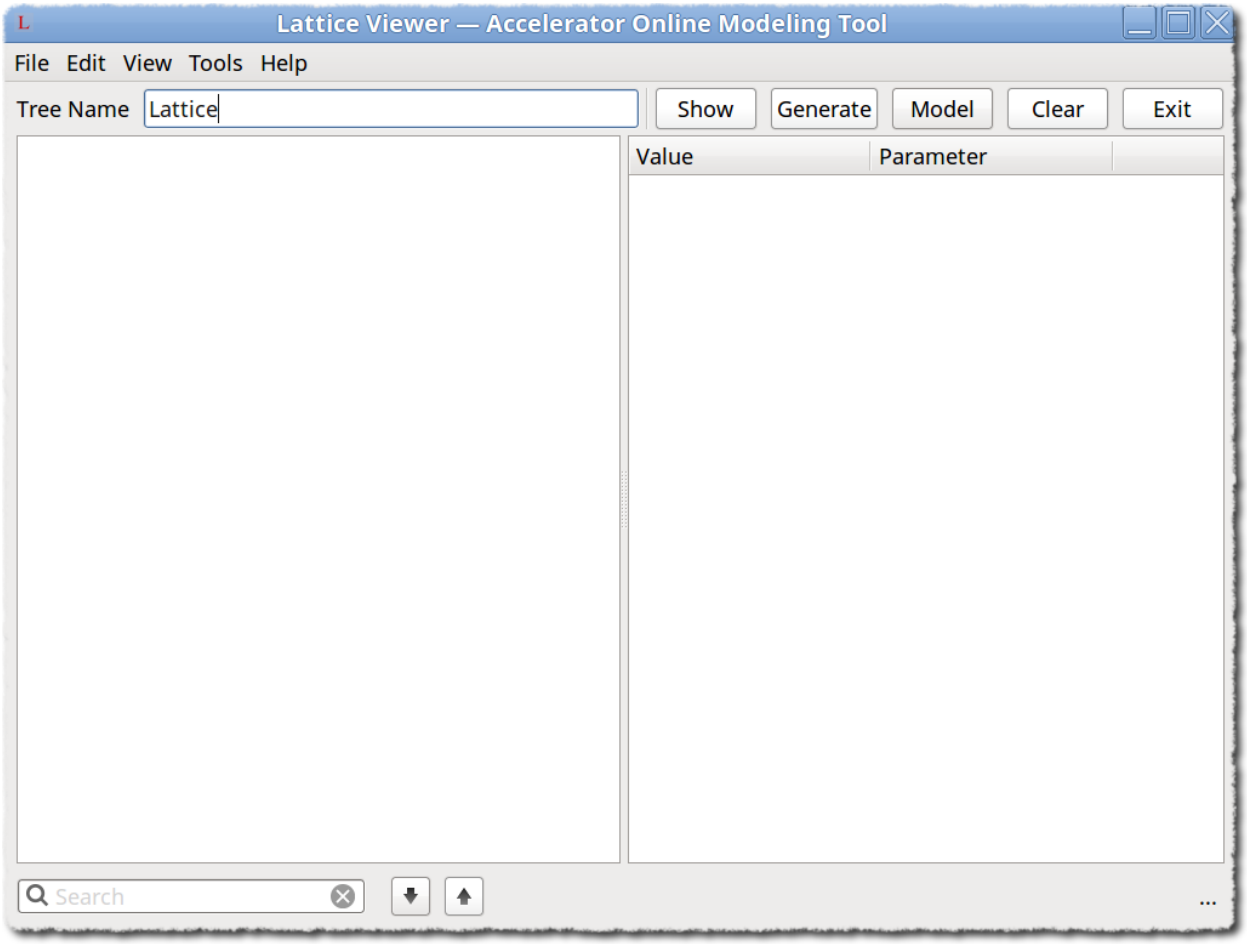
Note: `latticeviewer` is an app based on package `beamline`, and is wrapped by `felapps` to present the GUI to the users, either in the main panel of `runfelapps` or `appdrawer`, or the script named `latticeviewer` or `lv`.

Introduction

This app is built upon the console environment of accelerator online-model, which provides the basic toolkits to translate the real accelerator machine (as well as free-electron laser facility) into computer-recognized objects, such that manipulation to the real machine should be operated on the software level, so intuitively, this app should be the base on which physics-related procedures/extensions could be built. In this example, the basic online-model procedure will be presented, contrasted to the CLI approach demonstrated here.

Usage Guide

Open app by `lv` or `latticeviewer`, just like the following image shows:



The acceptable file should be either valid `lte` file (which is lattice file that could be used by `Elegant`) or `JSON` file (which is converted from valid `lte` file).

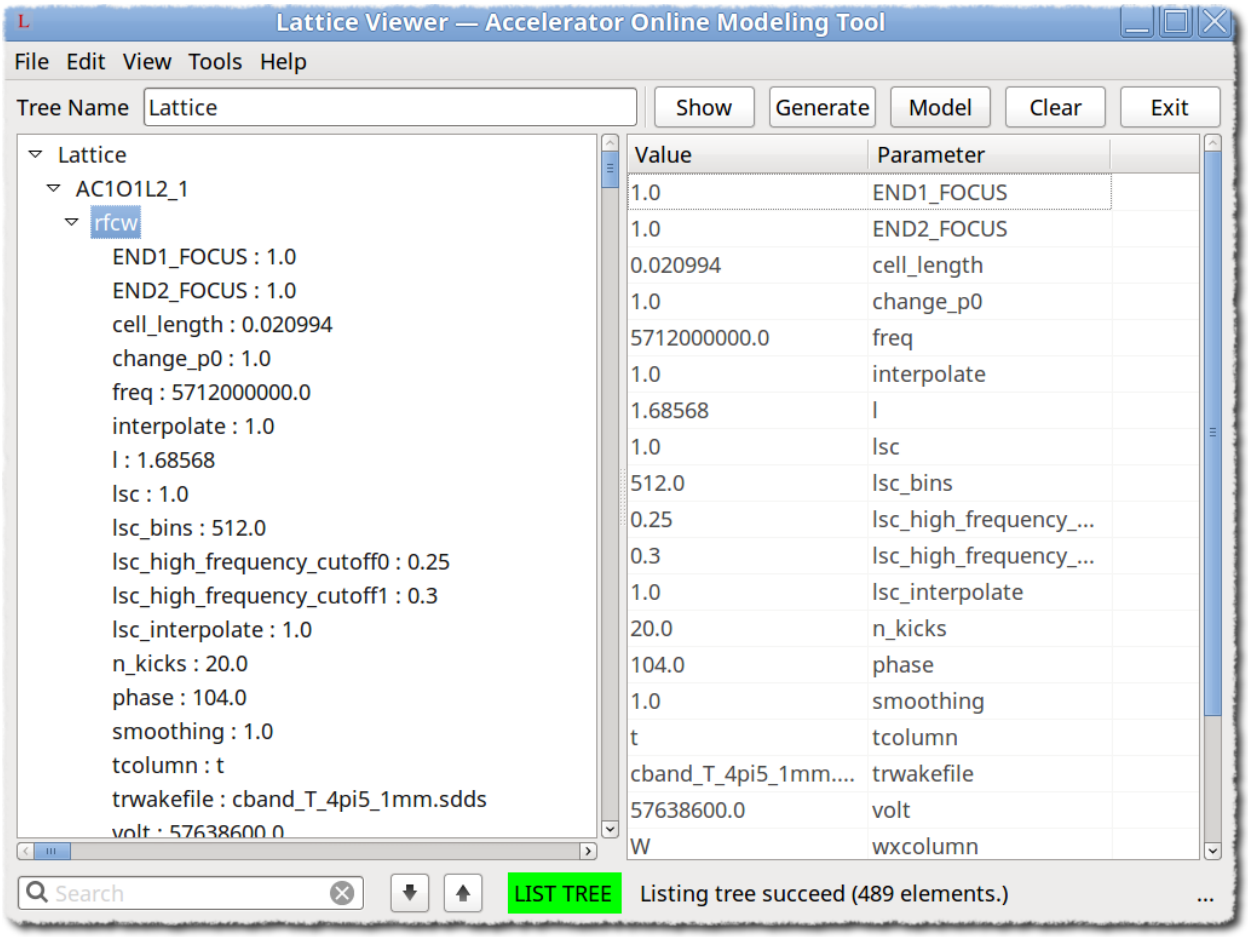
Note: `lte` file could be converted to `JSON` file and vice versa, see commands' doc: `lte2json` and `json2lte`.

Open File

Click `File > Open (CTRL+O)` to import the lattice file (`lte` or `JSON`), here take the lattice of Shanghai soft x-ray free-electron laser (SXFEL) as example, `here` is the `lte` file, which covers injector, LINAC, linac-to-undulator

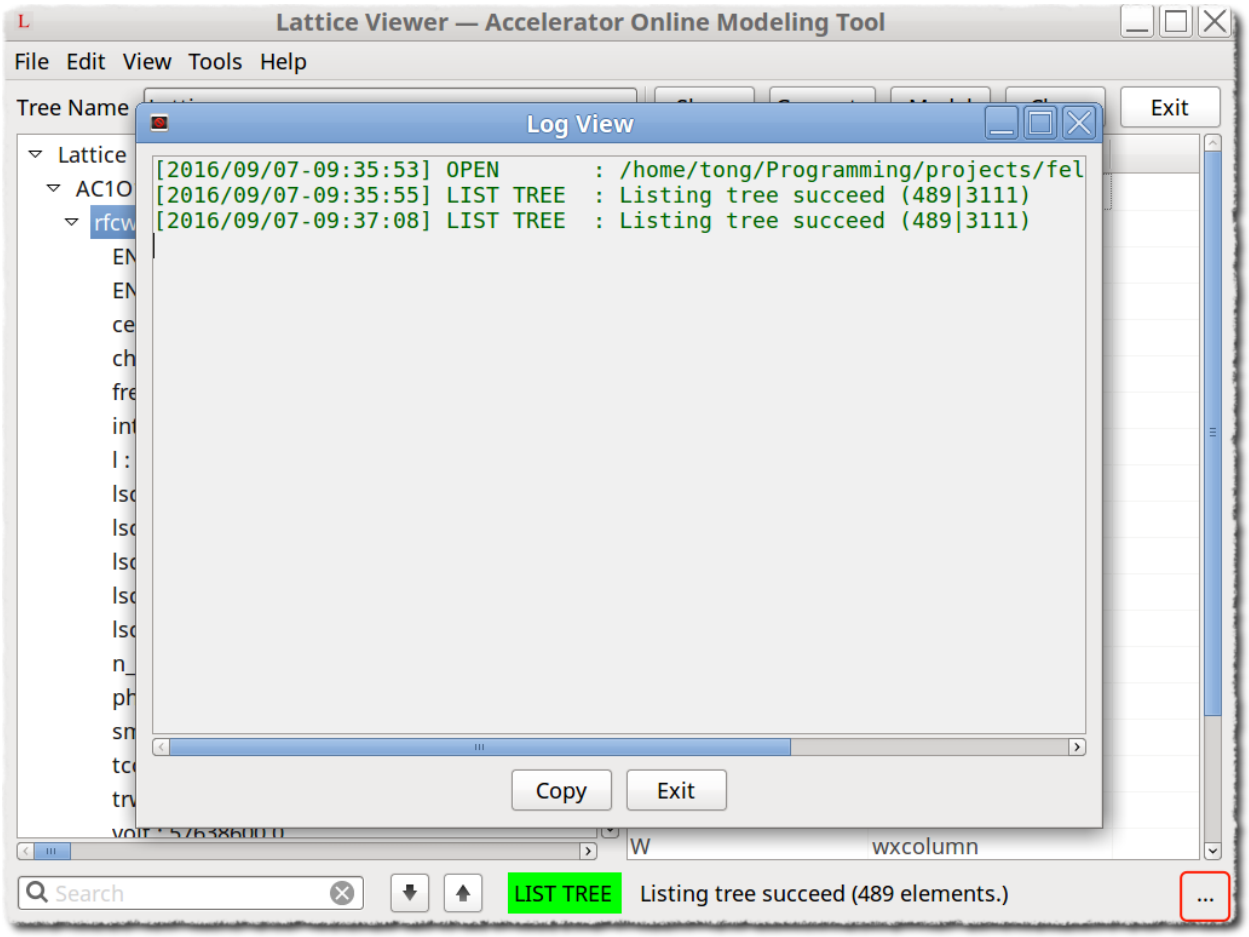
sections, while possible `ele` file and other required files that could be used to accomplish `Elegant` tracking could be download `here` and `injector data` if needed.

After the file is opened, the lattice would be parsed into tree list as shown in the left panel of `latticeviewer`, with the tree rootname of `Lattice` (which could be changed in `Tree Name` box [1]). All elements could be reached either by clicking the items or by searching the keywords in the left bottom search box. When element is selected, e.g. `rfcw` is selected, the right panel will list all the properties of selected item, just like the following image shows.



In the right bottom, push red squared button, the brief operation log will pop up, from which one can grab basic information about the loaded lattice file, e.g. here indicates that 489 elements and 3111 properties have been parsed, if operation fails, the green boxed `LIST TREE` status should turn red.

[1] When the tree name is changed, `Reopen (CTRL+R)` is needed.



Currently, the operations of the properties listing panel only support `Highlight` and `Edit`, it is envisioned that more functionality should be extended.

Operations

`latticeviewer` is designed to be an online-modeling app, other high-level operations should be based on the fundamental lattice parsing feature, below is the main operations:

Show

If lattice file is not opened by `Open` menu, push `Show` button should give guideline to locate the valid lattice file to show.

Generate

This button if the first step of online-modeling, heavy tasks about the element/beamline modeling will be done if `Generate` is pushed, dialog will pop up if some other operation trigs this operation.

Model

This button is required by the online-modeling, another approach implemented on the GUI app [2].

Clear

Clear the lattice tree.

[2] The CLI approach could be found in `beamline`'s doc.

Exit

Quit the app.

Menu operations

- Save, `CTRL+S`;
- Save AS, `SHIFT+CTRL+S`;
- Choose beamline, `SHIFT+CTRL+B`;
- Lattice visualization, `SHIFT+CTRL+V`.
- View lattice in three format: `raw string`, `JSON string`, `dict`.

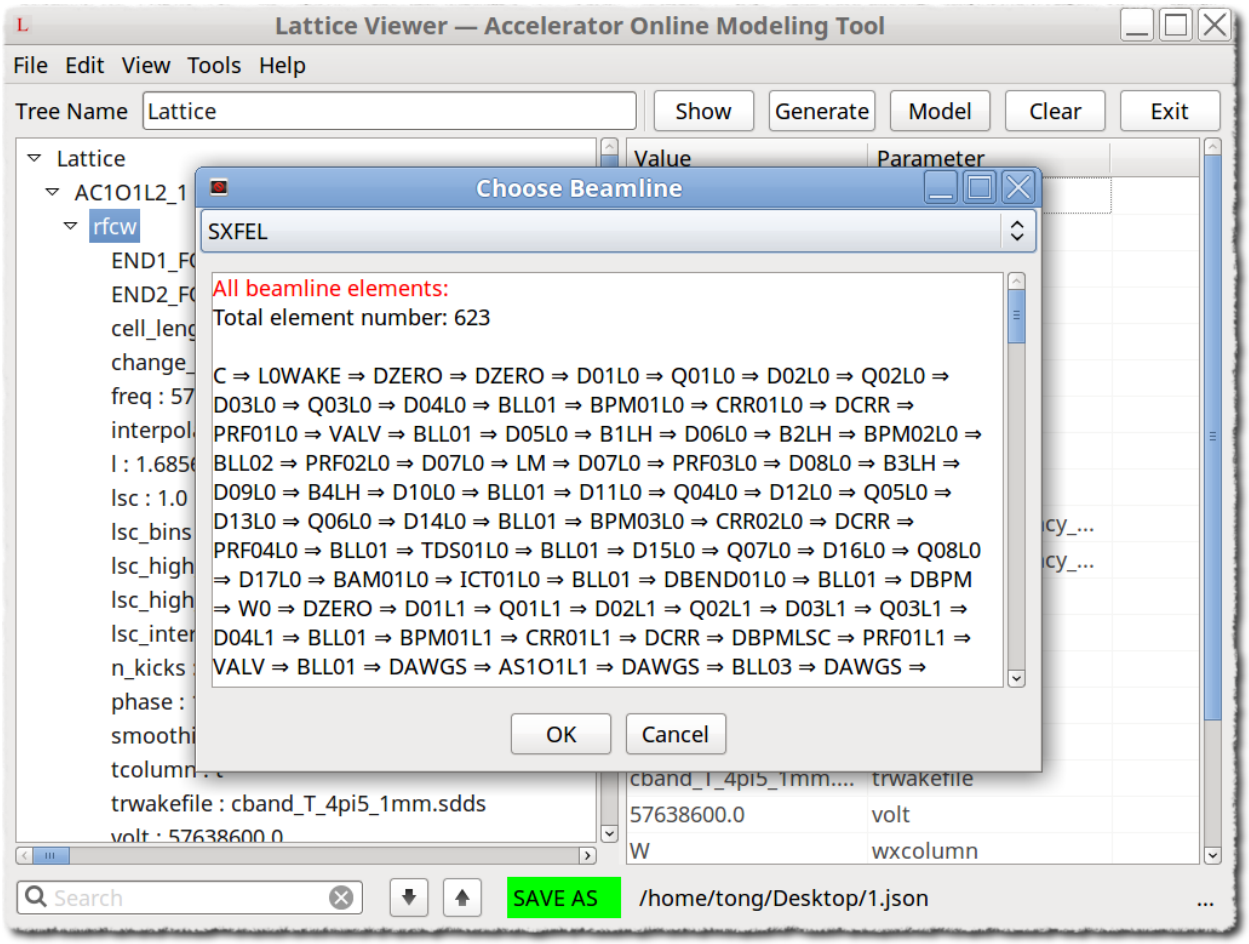
Examples

Save AS

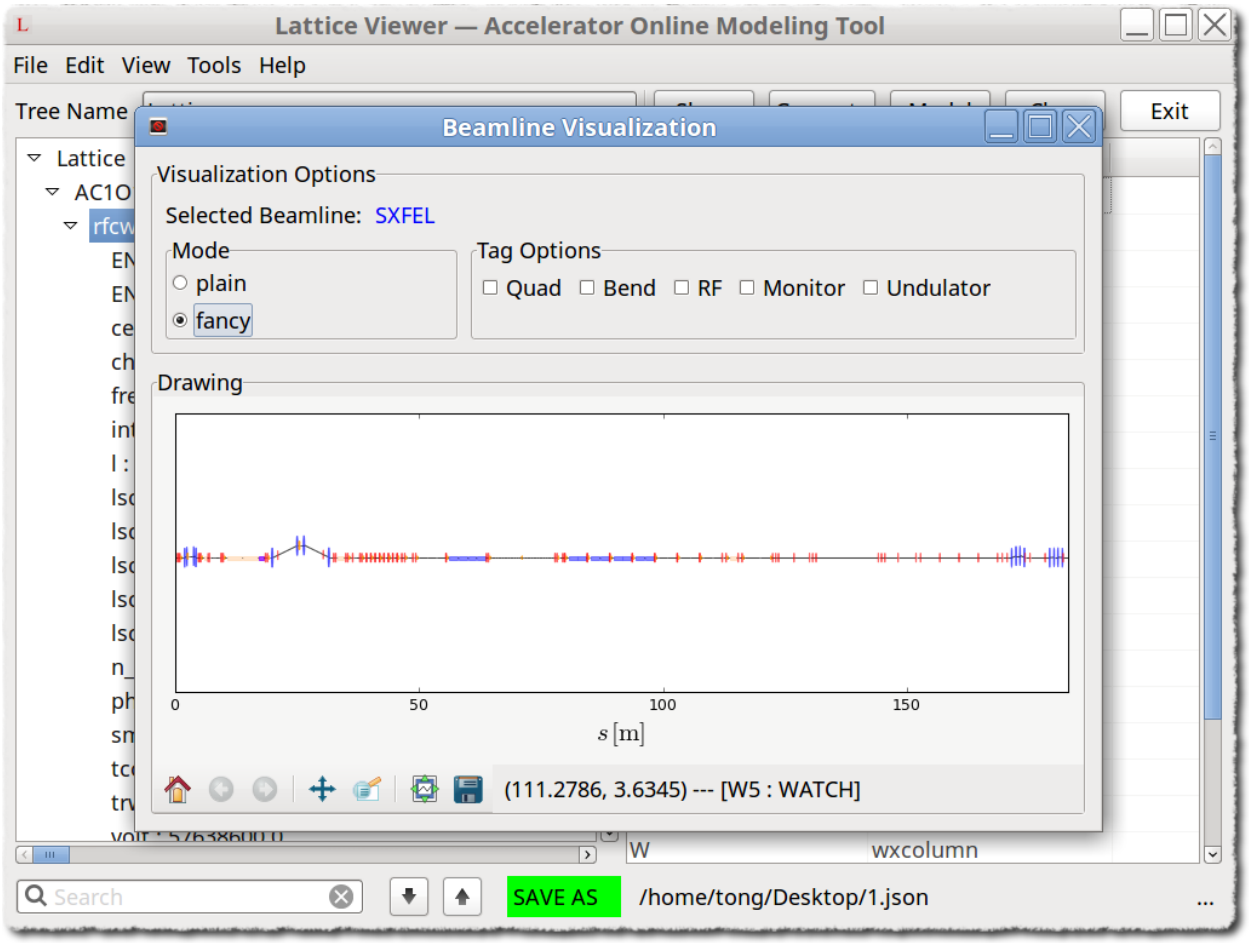
Push `Generate`, and `SHIFT+CTRL+S` to save lattice as `JSON` format.

Model Selected Beamline

Push `SHIFT+CTRL+B` to select the beamline name that to be modeled, e.g. `SXFEL` here is the beamline defined in `lte` file that including elements from LINAC and LTU. Click `OK` to confirm.



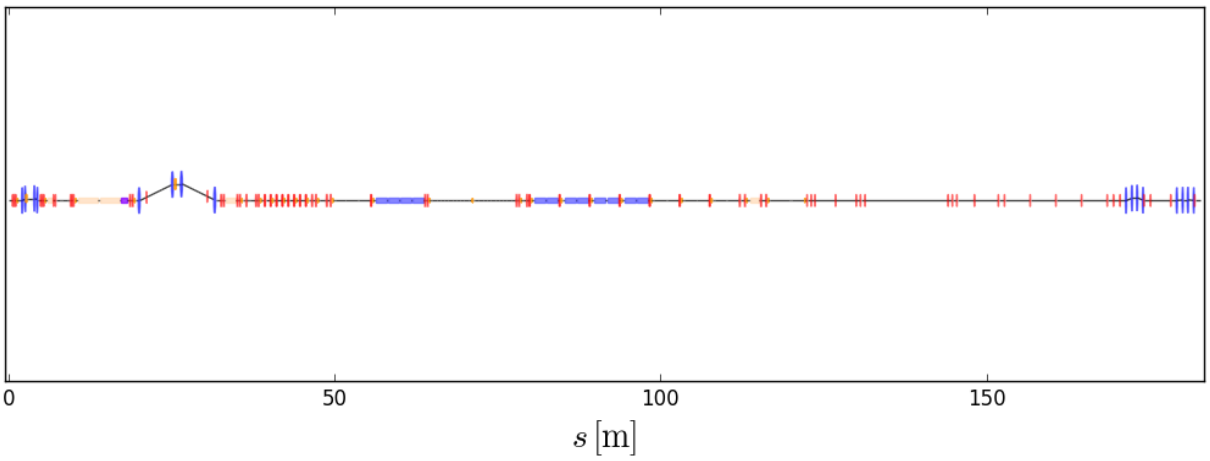
Click `Model` button to finish the modeling procedure, then push `SHIFT+CTRL+V` to open `Tools > Visualization` window.



In this window, the artist view of the selected beamline is firstly shown in the `Drawing` panel, which support features like:

- Operations provides by the bottom toolbar, e.g. *panning*, *zooming*, *save as*, etc.
- Element name tooltips, i.e. show the element name and type in the right of toolbar when mouse move onto the element, as well as the `s` position along the beamline.

Also, type name annotation feature is supported, by checking in `Tag Options` panel.



Todo

Note that `latticeviewer` is just a GUI app that only facilitate part of online-modeling operations, all the infrastructure features that make this possible are built into CLI, thus more features that would be implemented in the GUI framework should require much more effect. The ideal development cycle is to built high-level operations to the lattice into `Tools` menu, after all the algorithms have been well tested in the CLI framework.

Example 6

`wxmpv` – General matplot viewer app.

Introduction

`wxmpv` is the brief of `wx[Python] matplot[lib] viewer` or `wx[Python] mat[lab] plot viewer`, is designed with the intention of to provide the universal interface to manipulate the data files in the plot analysis stage, basic features should be included, such as:

- data loading/saving
- figure plot
- figure style tunning
- curve fitting
- etc.

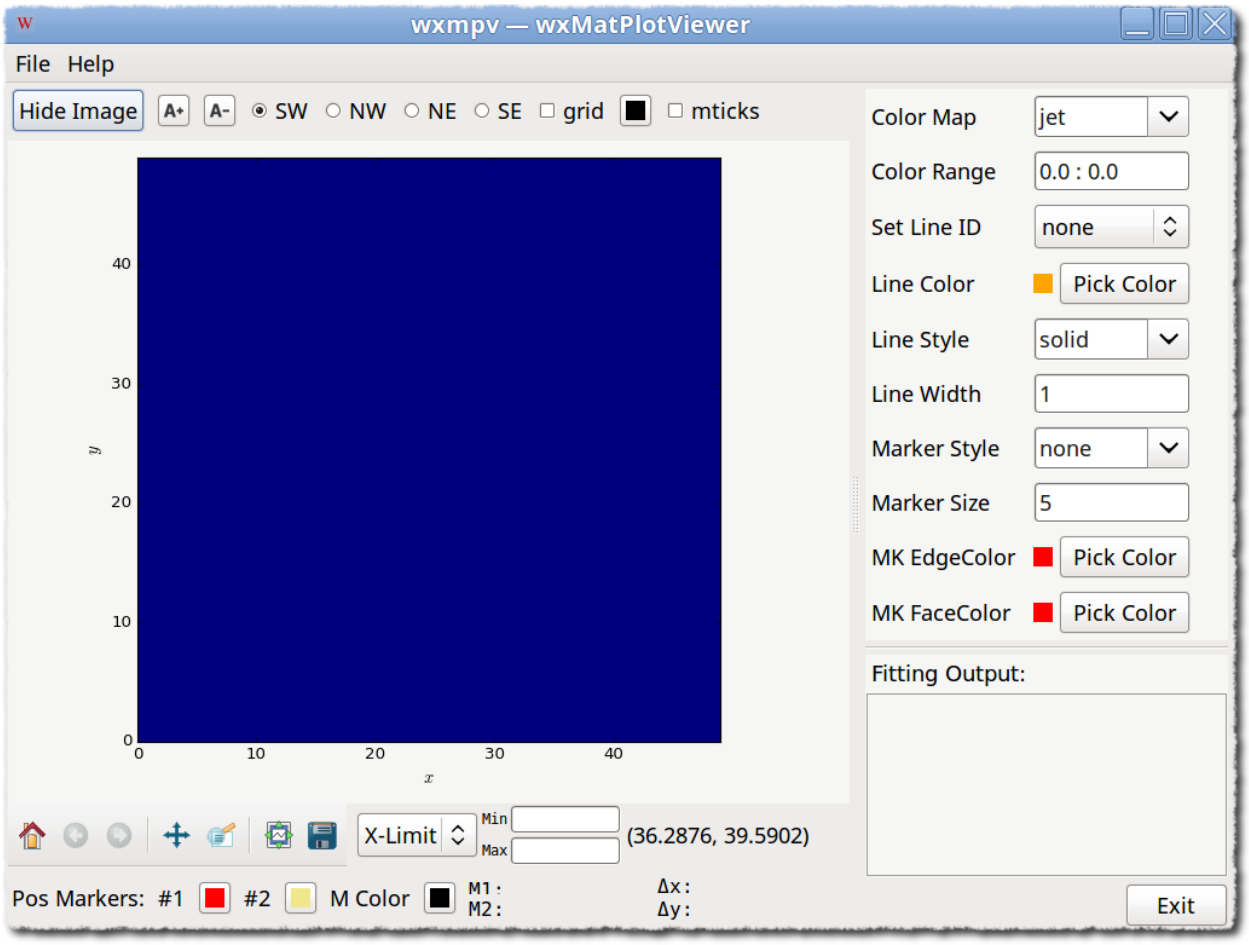
Currently, `wxmpv` could be used to operate these two (2) kinds of `hdf5` data that generated from `imageviewer` [1]:

- data saved by `CTRL+D` in `imageviewer`
- data generated from `Autosave` operation of `imageviewer`

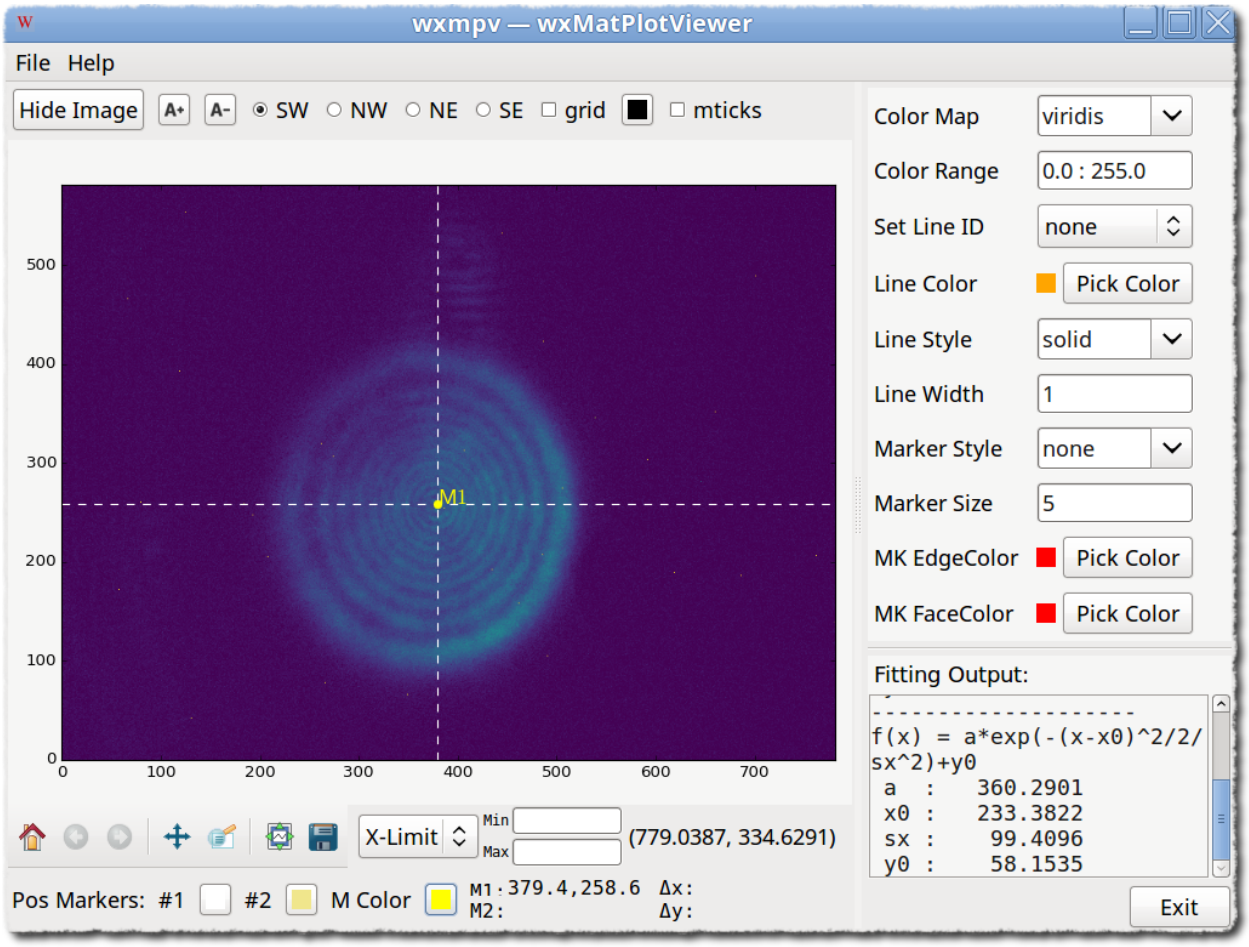
[1] `hdf5` data format should be first set in `Configurations` menu, `Image Data Name Extension` and choose `hdf5` in `Auto save` window.

Usage Guide

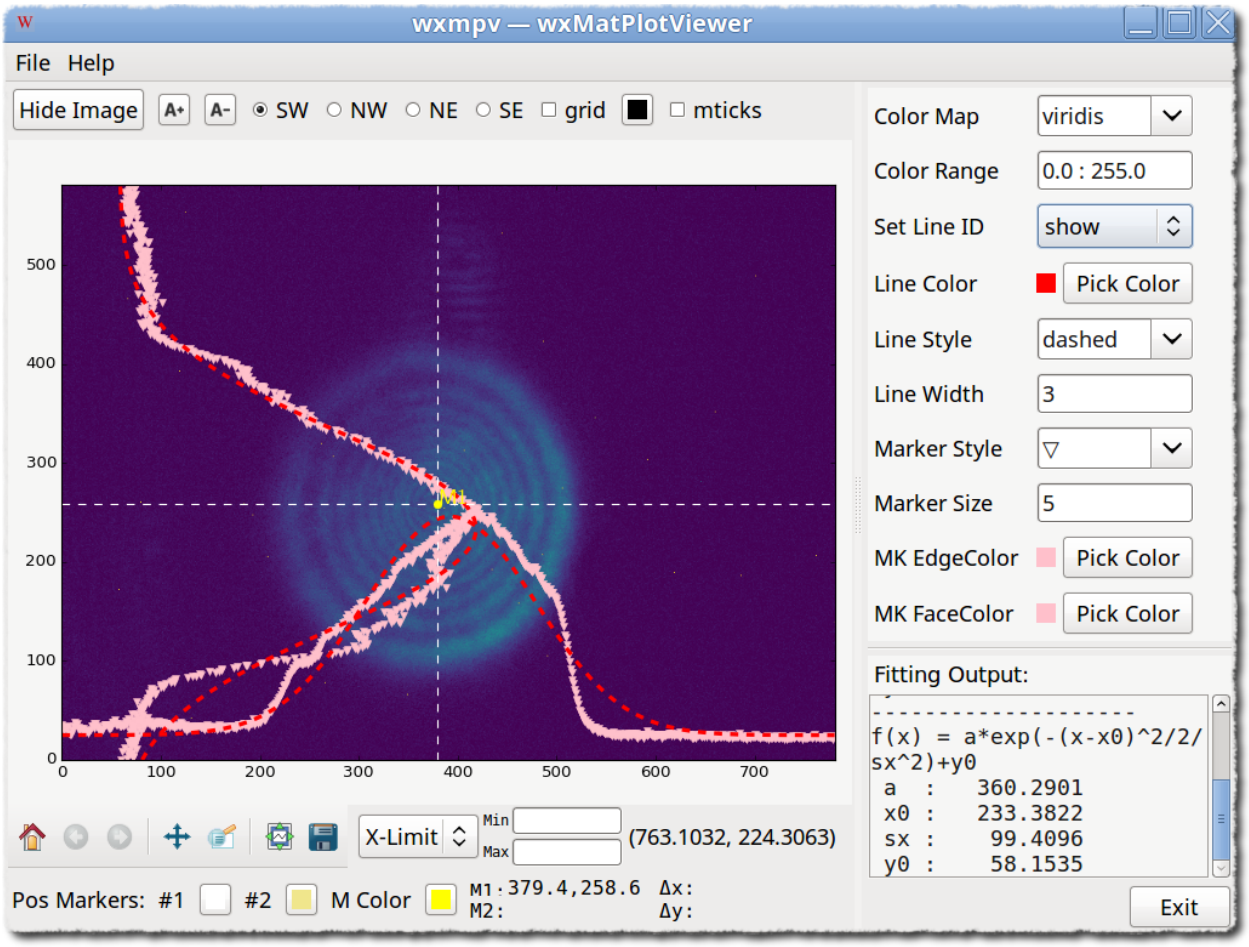
Open `wxmpv`, just like the following image shows, there are mainly two operation sections, the top and bottom region of figure and the style configuration panel in the right, as well as the curve fitting panel.

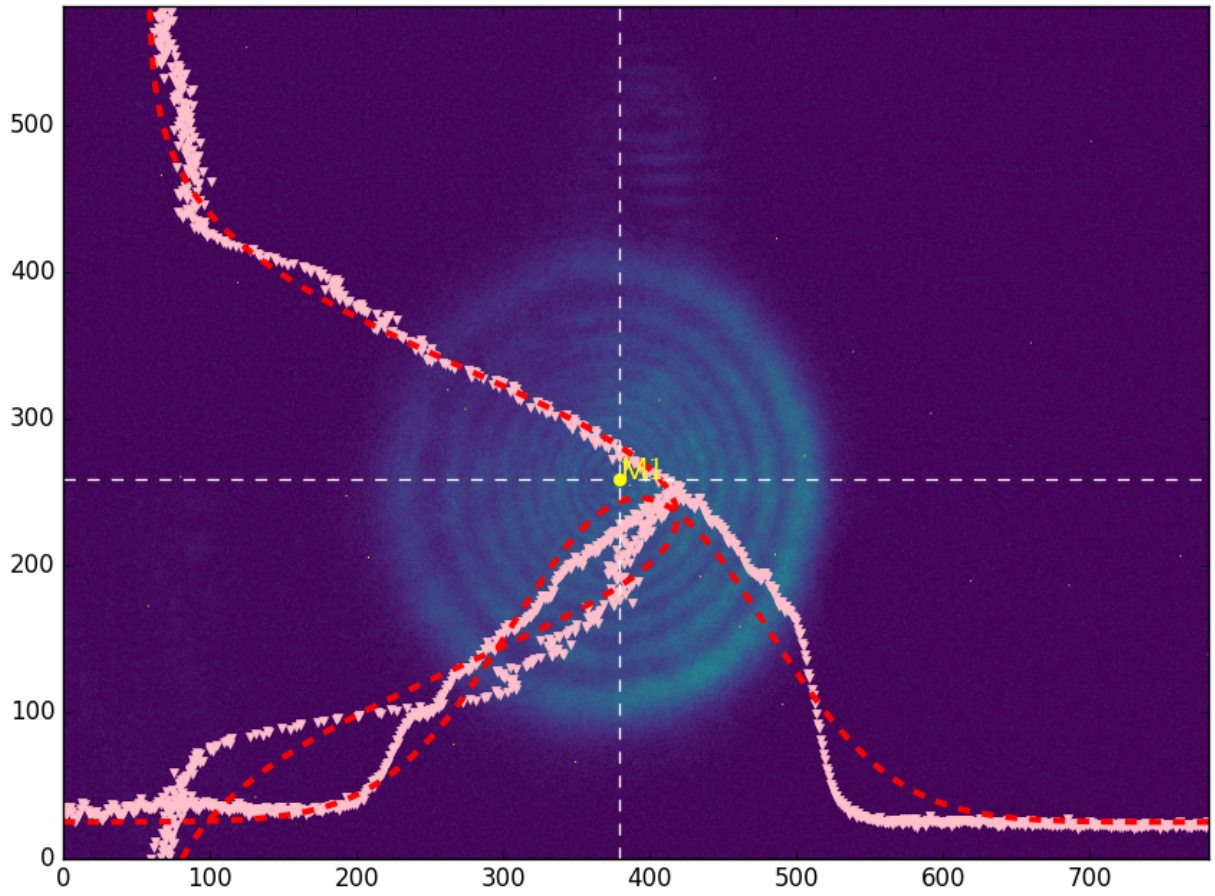


Click `File > Open (CTRL+O)` to open supported `hdf5` data file, both image and data curve should be plotted onto the figure, style adjustment to the image and curve is possible.



For example, when tunning the curve styles, push `Hide Image` to hide the image temporarily, and click `Set Line ID` to select the line to adjust the style, valid options: line color, line width, line style, marker color, marker size, marker style, etc.; while the image could be adjusted with different type of color maps and color ranges, to meet the users' demanding.


wxmpv — wxMatPlotViewer
File Help
Hide Image
A+
A-
SW
NW
NE
SE
grid
mticks
Color Map
viridis
Color Range
0.0 : 255.0
Set Line ID
show
Line Color
Pick Color
Line Style
dashed
Line Width
3
Marker Style
Marker Size
5
MK EdgeColor
Pick Color
MK FaceColor
Pick Color
Fitting Output:
f(x) = a*exp(-(x-x0)^2/2/sx^2)+y0
a : 360.2901
x0 : 233.3822
sx : 99.4096
y0 : 58.1535
X-Limit
Min
Max
(763.1032, 224.3063)
Pos Markers: #1 #2 M Color
M1: 379.4,258.6 Δx:
M2: Δy:
Exit



The figure could be saved as `eps`, `ps`, `png`, etc. Other options could be tried by the users.

Todo

`wxmpv` should be extended to support more data plotting functionality, so as to serve as the general-purpose data plotting and manipulation app.

Shortcut: `CTRL+S` would save all the data into `hdf5` format, sample file could be downloaded `here`.

CHAPTER

FOUR

API

felapps package

Python package created for the commissioning of free-electron laser facilities, providing general-purpose graphical user interface applications to fulfill various tasks, such as image DAQ, data post-processing, data correlation analysis and FEL physics calculations, etc. Friendly user interative approaches are subtly designed, as well as the straight-forward distribution manner.

`felapps` is the beginning phase to the world of python-powered software ecosystem that could be served as the infrastructure for the future versatile large-scale scientific facilities.

> **Warning:** Before installing `felapps`, another python package `beamline` is required, so install that package first, see the documentation.

Brief guide to the users:

1. type `runfelapps` or `appdrawer` in the terminal to open the main app portal of `felapps`;
2. in [i]python terminal, first `import felapps`, then call `felapps.imageviewer.run()` to open `imageviewer` app, the same rule applies to other apps:
   - `dataworkshop`
   - `cornalyzer`
   - `felformula`
   - `wxmpv`

Version 1.5.13

Author Tong Zhang (zhangtong@sinap.ac.cn)

Subpackages

felapps.apps package

This subpackage is designed for the interfaces of ready-to-use GUI applications that built from other subpackages, such as `utils`, `physics`, etc.

These are the applications available:

- appdrawer
- cornalyzer

- dataworkshop
- felformula
- imageviewer
- latticeviewer [1]
- wxmpv

Subpackages

felapps.apps.appdrawer package

Submodules

felapps.apps.appdrawer.appdrawer module

main launcher panel for all available apps from felapps

Author: Tong Zhang Created: Oct. 8, 2015

class **AppDrawerFrame**(*parent*, *appversion='1.0'*, *icon_set=None*, ***kwargs*)
Bases: `wx._windows.Frame`

createMenubar()

createPanel()

createStatusbar()

createToolbar()

exitApp()

initUI()

onExit(*event*)

postInit()

preInit()

class **AppDrawerPanel**(*parent*, *version*, *icon_set*, **args*, ***kwargs*)
Bases: `wx._windows.Panel`

createPanel()

onClickAppC(*event*)

onClickAppD(*event*)

onClickAppF(*event*)

onClickAppI(*event*)

onClickAppL(*event*)

onClickAppM(*event*)

onClickAppW(*event*)

onTickTime(*event*)

[1] Implement in package `beamline`, only integrate into `appdrawer`.

class `InspectApp`(*redirect=False*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)
Bases: `wx._core.App`, `wx.lib.mixins.inspection.InspectionMixin`

`OnInit`()

`run`(*maximize=True*, *logon=False*, *debug=True*, ***kws*)
function to make appdrawer app run.

Module contents

felapps.apps.cornalyzer package

Submodules

felapps.apps.cornalyzer.cornalyzer module

cornalyzer: correlation analyzer, study the relation between parameters

Author: Tong Zhang Created: May. 27, 2015

class `InspectApp`(*redirect=False*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)
Bases: `wx._core.App`, `wx.lib.mixins.inspection.InspectionMixin`

`OnInit`()

`run`(*maximize=True*, *logon=False*, *debug=True*)
function to call cornalyzer

Module contents

felapps.apps.dataworkshop package

Submodules

felapps.apps.dataworkshop.dataworkshop module

DataWorkshop: application to handle data, e.g. generated from imageviewer

Author: Tong Zhang Created: Sep. 23rd, 2015

class `InspectApp`(*redirect=False*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)
Bases: `wx._core.App`, `wx.lib.mixins.inspection.InspectionMixin`

`OnInit`()

`run`(*maximize=True*, *logon=False*, *debug=True*)
function to make dataworkshop app run.

Module contents

felapps.apps.felformula package

Submodules

felapps.apps.felformula.felformula module

FelFormula: calculation cheat sheet for free-electron laser physics

Author: Tong Zhang Created: May. 28, 2015 (re-organized)

class **InspectApp**(*redirect=False*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)
Bases: wx._core.App, wx.lib.mixins.inspection.InspectionMixin

OnInit()

run(*maximize=True*, *logon=False*, *debug=False*)
function to call felformula GUI

Module contents

felapps.apps.imageviewer package

Submodules

felapps.apps.imageviewer.imageviewer module

ImageViewer: universal application for image viewing and data post-processing.

Author: Tong Zhang Created: Feb. 3rd, 2015

class **InspectApp**(*redirect=False*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)
Bases: wx._core.App, wx.lib.mixins.inspection.InspectionMixin

OnInit()

run(*maximize=True*, *logon=False*, *debug=False*)
function to make imageviewer app run.

Module contents

felapps.apps.wxmpv package

Submodules

felapps.apps.wxmpv.wxmpv module

General matplotlib plot panel developed by wxPython (UI) and Python

Author : Tong Zhang Created: 2016-07-05 21:20:05 PM CST

class **InspectApp**(*redirect=False*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)
Bases: wx._core.App, wx.lib.mixins.inspection.InspectionMixin

OnInit()

class **MainFrame**(*parent*, *appversion='0.2'*)
Bases: felapps.utils.analysisframe.AnalysisFrame

on_about(*event*)

on_exit(*event*)

on_info(*event*)

on_open(*event*)

on_save(*event*)

save_data(*fname*, *data*)

run(*logon=False*, *debug=False*)
function to make app run.

Module contents

felapps.physics package

This subpackage is designed for the purpose of including physics-related modules, both accelerator physics and FEL physics.

Submodules

felapps.physics.felbase module

Python module for the basis of free-electron laser calculation.

class FELcalc(*_beamEnergy=6000.0*, *_relativeEnergySpread=0.0001*, *_unduPeriodLength=0.015*, *_avgBetaFunc=20.0*, *_radWavelength=1e-10*, *_normEmittance=4e-07*, *_peakCurrent=3500.0*, *_bunchCharge=5e-10*, *_undulatorLength=10.0*, *_bunchShape='gaussian'*, *_undulatorType='planar'*)
Bases: felapps.physics.felbase.PhysicalConstants

Analytical calculation for Free-electron Laser physics

Usage: res = FELcalc(p1, p2, p3, p4, p5, p6, p7, p8, p9, p10)

Parameters

- **p1** – beamEnergy, [MeV]
- **p2** – relative energy spread
- **p3** – undulator period length, [m]
- **p4** – avgerage beta function, [m]
- **p5** – radiation wavelength, [m]
- **p6** – normalized transverse emittance, [m]
- **p7** – peak current, [A]
- **p8** – bunchCharge [C]
- **p9** – undulatorLength [m]
- **p10** – bunchShape, 'gaussian' or 'flattop'

Return res dict, keys: "au", "bu", "gap", "sigmar", "rho1D", "rho3D", "Lg1D", "Lg3D", "Psat", "Pshot", "Pss"

findSatFactor(*nl*, *l3*, *xlamd*, *factor0=20*)
Calculator saturation length in the unit of 3D power gainlength

Parameters

- **factor0** – initial saturation factor, saturation length over power gain length
- **nl** – electron count within one unit of FEL wavelength
- **l3** – power gain length (3D)
- **xlamd** – undulator period

onFELAnalyse()
Apply M. Xie formulae for FEL analytical estimation

class HalbachPerm(*_a=3.33*, *_b=-5.47*, *_c=1.8*, *_lambdau=20*, *_Bu=1.0*)
Bases: object

Input parameters:

Parameters

- **_a** – first Halbach parameter
- **_b** – second Halbach parameter
- **_lambdau** – undulator period length, [mm]

Parma _c third Halbach parameter

Parma _Bu undulator magnetic field, [T]

findGap(*gap0=10*)
Solve undulator gap value

Parameters **gap0** – initial gap value vector, [mm]

Returns gap value in [mm]

class PhysicalConstants
Bases: object

Physical constants

Parameters

- **c0** – c_0, velocity of light in vacuum
- **epsilon0** – ϵ_0, permittivity in vacuum
- **mu0** – μ_0, permeability in vacuum
- **e0** – e, electron charge, [C]
- **m0** – m_e, electron mass, [kg]
- **h0** – h, Plank constant
- **currentA** – I_A, Alven current, [A]

c0 = 299792458.0

currentA = 17045

e0 = 1.60218e-19

epsilon0 = 8.85418781762039e-12

h0 = 6.62607e-34

m0 = 9.10938e-31

mu0 = 1.2566370614359173e-06

test1()

test2()

test3()

felapps.physics.felcalc module

Author: Tong Zhang Create Time: 15:22, Jan. 15, 2015 first version is written in MATLAB on Dec. 25th, 2014.

class DataSaveFrame(*parent*, *dataparent*, *title*, ***kwargs*)
Bases: wx._windows.Frame

InitUI()

onCheckCols1(*event*)

onCheckCols2(*event*)

onChooseFmt(*event*)

onChoosePth(*event*)

onExit(*event*)

onSave(*event*)

class ExportFrame(*parent*, *title*, ***kwargs*)
Bases: wx._windows.Frame

InitUI()

onCancel(*event*)

onChoose(*event*)

onExport(*event*)

class InfoFrame(*parent*, *title*, ***kwargs*)
Bases: wx._windows.Frame

InitUI()

class LogFrame(*parent*, *title*, ***kwargs*)
Bases: wx._windows.Frame

InitUI()

onShowCLog(*event*)

class MainFrame(*parent*, *size=(800, 600)*, *appversion='1.0'*, ***kwargs*)
Bases: wx._windows.Frame

collectAllParams()

createMenu()

createPanel()

createStatusbar()

initUI()

onAbout(*event*)

onCalc(*event*)

onChangelog(*event*)

onEnableScan(*event*)

onExit(*event*)

onExport(*event*)
Export input parameters and output results into file (for next import or archive)

onExportJ(*event*)
Export input parameters and output results as a json string to file

onImport(*event*)
Import parameters from external configuration file

onInfo(*event*)

onMenuHL(*event*)

onPlot(*event*)

onSetInitalScanRange(*event*)

onTextEnter(*event*)

updateUI(*paramsdict*)
update UI with import parameters from .conf file

class PlotFrame(*parent*, *title*, ***kwargs*)
Bases: `wx._windows.Frame`

InitUI()

createMenu()

createPanel()

createToolbar()

onAboutPlot(*event*)

onExitPlot(*event*)

onGridCheck(*event*)

onMotion(*event*)

onPlotChoice(*event*)

onSaveData(*event*)

onSavePlot(*event*)

repaint()

setColor(*rgbtuple=None*)

run()

felapps.utils package

This subpackage is designed for providing essential modules, classes, functions that `felapps` needs to accomplish various different tasks, basically all the modules are grouped by the type of the task.

Submodules

felapps.utils.analysisframe module

Subclass of PlotFrame, which is generated by wxFormBuilder.

class **AnalysisFrame**(*parent*, *datasrc*)
Bases: felapps.utils.myui.PlotFrame

cmap_cbOnCombobox(*event*)

crange_tcOnTextEnter(*event*)

direct_xyaxis(*x_direction*, *y_direction*)

exitApp()

exit_btnOnButtonClick(*event*)

get_data(*datasrc=None*)

grid_ckbOnCheckBox(*event*)

imhide_tgbtnOnToggleButton(*event*)

lc_btnOnButtonClick(*event*)

lineid_choiceOnChoice(*event*)

ll_originOnRadioButton(*event*)
set origin lower left

lr_originOnRadioButton(*event*)
set origin lower right

ls_cbOnCombobox(*event*)

lw_tcOnTextEnter(*event*)

mec_btnOnButtonClick(*event*)

mfc_btnOnButtonClick(*event*)

mk_cbOnCombobox(*event*)

ms_tcOnTextEnter(*event*)

mticks_ckbOnCheckBox(*event*)

onDecFontSize(*event*)

onExit(*event*)

onIncFontSize(*event*)

onPickGridc(*event*)

onPickMK1c(*event*)

onPickMK2c(*event*)

onPickPcc(*event*)

pick_color()

post_init()

set_fit_output()

set_staticbmp_color(*obj*, *color*)

ul_originOnRadioButton(*event*)
set origin upper left

ur_originOnRadioButton(*event*)
set origin upper right

felapps.utils.datascanapp module

New App for data correlation analysis

Tong Zhang 2016-07-12 10:11:05 AM CST

class MainFrame(*parent*, *appversion='0.1'*)
Bases: felapps.utils.datascanframe.DataScanFrame

about_mitemOnMenuSelection(*event*)

close_btnOnButtonClick(*event*)

exit_app()

exit_mitemOnMenuSelection(*event*)

on_exit(*event*)

save_mitemOnMenuSelection(*event*)

felapps.utils.datascanframe module

Subclass of DataScanFrame, which is generated by wxFormBuilder.

class DataScanFrame(*parent*)
Bases: felapps.utils.datascanframe_ui.DataScanFrame

about_mitemOnMenuSelection(*event*)

adv_mode_ckbOnCheckBox(*event*)

auto_title_ckbOnCheckBox(*event*)

auto_xlabel_ckbOnCheckBox(*event*)

clr_retake_btnOnButtonClick(*event*)

daq_flag_rbOnRadioButton(*event*)

daq_pv_list_btnOnButtonClick(*event*)

daqrate_scOnSpinCtrl(*event*)

ds_flag_rbOnRadioButton(*event*)

exit_mitemOnMenuSelection(*event*)

fit_config_btnOnButtonClick(*event*)

fit_config_ckbOnCheckBox(*event*)

fit_curve(*model='gaussian'*)

fit_model_cbOnCombobox(*event*)

fit_refresh_btnOnButtonClick(*event*)
refresh fit

fit_report(*fm*)
fill fit_report_tc with curve fitting report

fit_to_fig_btnOnButtonClick(*event*)
stick fit result onto scan figure

grid_ckbOnCheckBox(*event*)

iternum_scOnSpinCtrl(*event*)

lc_btnOnButtonClick(*event*)

legend_ckbOnCheckBox(*event*)

lineid_cbOnCombobox(*event*)

ls_cbOnCombobox(*event*)

lw_scOnSpinCtrl(*event*)

mec_btnOnButtonClick(*event*)

mew_scOnSpinCtrl(*event*)

mfc_btnOnButtonClick(*event*)

mks_scOnSpinCtrl(*event*)

mkstyle_cbOnCombobox(*event*)

retake_btnOnButtonClick(*event*)

save_mitemOnMenuSelection(*event*)

scan_ctrl_timerOnTimer(*event*)
scan procedure control timer

scan_daq_timerOnTimer(*event*)
timer to control every scan iteration

set_retake_flag()
set retake process flag

set_scan_params()
set up scan parameters, var1(x) and var2(y)

shotperiter_scOnSpinCtrl(*event*)

show_scandiag()
show dialog when scan is done

start_btnOnButtonClick(*event*)

start_scan_daq_timer(*ms*, *scanidx*)

stick_pos_tcOnTextEnter(*event*)

stop_btnOnButtonClick(*event*)

tick_timerOnTimer(*event*)

unset_retake_flag()

update_scanfigure()
update figure plot

user_title_ckbOnCheckBox(*event*)

user_xlabel_ckbOnCheckBox(*event*)

var1_from_tcOnTextEnter(*event*)

var1_pv_flag_ckbOnCheckBox(*event*)

var1_pv_tcOnTextEnter(*event*)

var1_to_tcOnTextEnter(*event*)

var2_op_comboxOnCombobox(*event*)

var2_pv_tcOnTextEnter(*event*)

waittime_scOnSpinCtrl(*event*)

class MyEditConfigFrame(*parent*, *string_list=None*, *label=None*)
Bases: felapps.utils.uiutils.EditListFrame

on_ok(*event*)

class MyEditFrame(*parent*, *string_list=None*, *label=None*)
Bases: felapps.utils.uiutils.EditListFrame

on_ok(*event*)

class MyEditListFrame(*parent*, *string_list=None*, *label=None*)
Bases: felapps.utils.uiutils.EditListFrame

on_ok(*event*)

felapps.utils.datascanframe_ui module

class DataScanFrame(*parent*)
Bases: wx._windows.Frame

about_mitemOnMenuSelection(*event*)

adv_mode_btnOnButtonClick(*event*)

adv_mode_ckbOnCheckBox(*event*)

auto_title_ckbOnCheckBox(*event*)

auto_xlabel_ckbOnCheckBox(*event*)

close_btnOnButtonClick(*event*)

clr_retake_btnOnButtonClick(*event*)

daq_flag_rbOnRadioButton(*event*)

daq_pv_list_btnOnButtonClick(*event*)

daqrate_scOnSpinCtrl(*event*)

ds_flag_rbOnRadioButton(*event*)

exit_mitemOnMenuSelection(*event*)

fit_config_btnOnButtonClick(*event*)

fit_config_ckbOnCheckBox(*event*)

fit_model_cbOnCombobox(*event*)

fit_refresh_btnOnButtonClick(*event*)

fit_to_fig_btnOnButtonClick(*event*)

grid_ckbOnCheckBox(*event*)

iternum_scOnSpinCtrl(*event*)

legend_ckbOnCheckBox(*event*)

lineid_cbOnCombobox(*event*)

ls_cbOnCombobox(*event*)

m_splitterOnIdle(*event*)

m_splitter_downOnIdle(*event*)

mkstyle_cbOnCombobox(*event*)

retake_btnOnButtonClick(*event*)

save_mitemOnMenuSelection(*event*)

shotperiter_scOnSpinCtrl(*event*)

start_btnOnButtonClick(*event*)

stick_pos_tcOnTextEnter(*event*)

stop_btnOnButtonClick(*event*)

user_title_ckbOnCheckBox(*event*)

user_xlabel_ckbOnCheckBox(*event*)

var1_from_tcOnTextEnter(*event*)

var1_pv_flag_ckbOnCheckBox(*event*)

var1_pv_tcOnTextEnter(*event*)

var1_to_tcOnTextEnter(*event*)

var2_op_comboxOnCombobox(*event*)

var2_pv_tcOnTextEnter(*event*)

class FuncListFrame(*parent*)

Bases: wx._windows.Frame

cancel_btnOnButtonClick(*event*)

ok_btnOnButtonClick(*event*)

path_btnOnButtonClick(*event*)

reload_btnOnButtonClick(*event*)

udefs_lcOnListItemSelected(*event*)

felapps.utils.datautils module

python modules for data processing utilities: dataworkshop: main GUI framework for data post-processing

Author: Tong Zhang Created: Sep. 23rd, 2015

class **DataImportThread**(*parent*, *target*, *datafiles*)
Bases: threading.Thread

run()

stop()

class **DataWorkshop**(*parent*, *config='config.xml'*, *size=(1000, 750)*, *appversion='1.0'*, ***kwargs*)
Bases: wx._windows.Frame

createMenubar()

createPanel()

createStatusbar()

createToolbar()

exitApp()

initUI()

onAbout(*event*)

onAdd(*event*)
Add datafiles

onAnalysis(*event*)

onAnimate(*event*)

onConfigApps(*event*)

onConfigLoad(*event*)

onConfigSave(*event*)

onExit(*event*)

onMenuHL(*event*)

onOpen(*event*)
select data files to be visulized in imagegrid panel Tasks:

- clear two attributes: image_list and fdata_list
- clear image grid panel to be ready for new data and image files

fdata_list: put loaded data files via open operation image_list: put generated image files from fdata_list

onSave(*event*)

onScaleDec(*event*)

onScaleInc(*event*)

onStatistics(*event*)

onStopWorker(*file_num*)

onTickTime(*event*)

postInit()

preInit()

updateImageGrid(*image_file_list*, *fdir*, *ftype*)

vizData(*datafiles*)
read data file and show images on right panel :param datafiles: list of data filenames, ext: hdf5 | dat | asc

class ImageGrid(*parent*, *figsize*, *dpi*, *bgcolor*, ***kwargs*)
Bases: felapps.utils.pltutils.ImagePanel

doPlot()

onGetData()

onMotion(*event*)

onPress(*event*)

repaint()

class PlotPanel(*parent*, ***kwargs*)
Bases: felapps.utils.funutils.AnalysisPlotPanel

clear()

doHist()

doScatter()

doXY2plot()

doXYplot()

on_motion(*event*)

refresh()

class StatPanel(*parent*, *datafiles*, ***kwargs*)
Bases: wx._windows.Frame

InitUI()

createPanel()

gaussian_fit_all()

onCentralPosStat(*event*)

onExit(*event*)

onIntensityStat(*event*)

onIntensityStatHist(*event*)

onRadiusStat(*event*)

postInit()

felapps.utils.felutils module

Author: Tong Zhang Created Time: 20:14, Jan. 28, 2015

FEL utilities: 1 Three functional sections for HGHG: 1.1: ModdsPanel 1.2: MatchPanel 1.3: RadisPanel

class ChiConfigPanel(*parent*, ***kwargs*)
Bases: wx._windows.Frame

InitUI()

OnHelp(*event*)

OnInter(*event*)

OnUpdateParams(*event*)

createMenu()

createPanel()

createStatusbar()

class EbeamConfigPanel(*parent*, ***kwargs*)
Bases: wx._windows.Frame

InitUI()

OnCalc(*event*)

OnDAQ(*event*)

OnHelp(*event*)

OnSingleShot(*event*)

OnTextEnter(*event*)

OnUpdate(*event*)

OnUpdateParams(*event*)

createMenu()

createPanel()

createStatusbar()

class LaserConfigPanel(*parent*, ***kwargs*)
Bases: wx._windows.Frame

InitUI()

OnDAQ(*event*)

OnLaserHelp(*event*)

OnLaserLoad(*event*)

OnSingleShot(*event*)

OnTextEnter(*event*)

OnUpdate(*event*)

OnUpdateParams(*event*)

createMenu()

createPanel()

createStatusbar()

class MatchPanel(*parent*, ***kwargs*)
Bases: wx._windows.Frame

InitUI()

createMenu()

createPanel()

createStatusbar()

onAbout(*event*)

onExit(*event*)

onLoad(*event*)

onSave(*event*)

class ModConfigPanel(*parent*, ***kwargs*)
Bases: wx._windows.Frame

InitUI()

OnHelp(*event*)

OnInter(*event*)

OnUpdateParams(*event*)

createMenu()

createPanel()

createStatusbar()

class ModdsPanel(*parent*, ***kwargs*)
Bases: wx._windows.Frame

InitUI()

OnConfigChicane(*event*)

OnConfigEbeam(*event*)

OnConfigLaser(*event*)

OnConfigModulator(*event*)

OnUpdatePS(*psid='mod'*, *newval=0*)

createMenu()

createPanel()

createStatusbar()

onAbout(*event*)

onExit(*event*)

onLoad(*event*)

onSave(*event*)

class RadisPanel(*parent*, ***kwargs*)
Bases: wx._windows.Frame

InitUI()

createMenu()

createPanel()

createStatusbar()

onAbout(*event*)

onExit(*event*)

onLoad(*event*)

onSave(*event*)

main(*ClassName=<class 'felapps.utils.felutils.ModdsPanel'>*)

felapps.utils.funclistframe module

class FuncListFrame(*parent*, *fullpath='.'*)
Bases: felapps.utils.datascanframe_ui.FuncListFrame

cancel_btnOnButtonClick(*event*)

ok_btnOnButtonClick(*event*)

path_btnOnButtonClick(*event*)

reload_btnOnButtonClick(*event*)

udefs_lcOnListItemSelected(*event*)

felapps.utils.funutils module

Author: Tong Zhang Created Time: 11:09, Jan. 29, 2015

utilities/functions for convenience

class AnalysisPlotPanel(*parent*, *data=None*, ***kwargs*)
Bases: felapps.utils.uiutils.MyPlotPanel

clear()

direct_xyaxis(*x_direction*, *y_direction*)

draw_hvlines(*x0*, *y0*)

get_clim()

get_data()
return data: image, raw data, fit data

get_fit_report(*xoy='x'*)
return fitting report if success, else return None

hide_image(*hide_flag*)

on_press(*event*)

on_release(*event*)

set_clim(*cr*)

set_color(*rgb_tuple*)
set figure and canvas with the same color. :param rgb_tuple: rgb color tuple,

e.g. (255, 255, 255) for white color

set_colormap(*cmap*)

set_figure_data(*data*, *fit=True*)

set_fontsize(*fontsize*)

set_grids(*color*, *b=None*, *which='major'*)

set_line_id(*line='raw'*)
selected current editable line, 'raw': raw data 'fitted': fitted lines 'none': hide all lines 'show': show all lines

set_linecolor(*color*)

set_lines()

set_linestyle(*ls*)

set_linewidth(*lw*)

set_marker(*mk*)

set_markersize(*ms*)

set_markflags(*mk1=False*, *mk2=False*)

set_mec(*c*)

set_mfc(*c*)

set_mkc1(*color*)

set_mkc2(*color*)

set_mticks(*flag='off'*)

set_origin(*ll=False*, *ul=False*, *ur=False*, *lr=False*)

set_pcc(*color*)

set_ticks(*flag='half'*)

update_deltxy()

class ExportData(*data_raw*, *data_fit*, *model_x*, *model_y*, *fname*)
Bases: object

onProcess()

onSave()

class FitModels(*model='gaussian'*, *params=None*, ***kws*)
Bases: object

calc_p0()
return p0 from input x, y

fit()

fit_report()

get_data()
return raw data

get_fit_result()

get_fitfunc(*p0=None*)

method

mode

model

set_data(*data=None*, *x=None*, *y=None*)
set raw data to fit

set_params(***p0*)
p0: initial parameters dict

class FloatSlider(*parent*, *id=-1*, *value=0*, *minValue=0*, *maxValue=10*, *increment=0.1*, *size=wx.Size(-1, -1)*, *style=4*, **args*, ***kws*)
Bases: wx._controls.Slider

GetInc()

GetMax()

GetMin()

GetRange()

GetValue()

SetInc(*inc*)

SetMax(*maxval*)

SetMin(*minval*)

SetRange(*minval*, *maxval*)

SetValue(*value*)

class ImageDataFactor(*z*)
Bases: object

getData()

getInt()

setData(*z*)

class MyButton(*parent*, *font=None*, *fontsize=10*, *fontcolor='black'*, **args*, ***kws*)
Bases: wx._controls.Button

setFont(*font*)

setFontColor(*fontcolor*)

setFontFaceName(*facename*)

setFontSize(*fontsize*)

class MyCheckBox(*parent*, *font=None*, *fontsize=10*, *fontcolor='black'*, **args*, ***kws*)
Bases: wx._controls.CheckBox

setFont(*font*)

setFontColor(*fontcolor*)

setFontFaceName(*facename*)

setFontSize(*fontsize*)

class MyComboBox(*parent*, *font=None*, *fontsize=12*, *fontcolor='black'*, *fontweight=90*, **args*, ***kws*)
Bases: wx._controls.ComboBox

setFont(*font*)

setFontColor(*fontcolor*)

setFontFaceName(*facename*)

setFontSize(*fontsize*)

class **MyListCtrl**(*parent*, *ID*, *pos=wx.Point(-1, -1)*, *size=wx.Size(-1, -1)*, *style=0*)
Bases: wx._controls.ListCtrl, wx.lib.mixins.listctrl.ListCtrlAutoWidthMixin

class **MySpinCtrl**(*parent*, *font=None*, *fontsize=10*, *fontcolor='black'*, *fontweight=90*, **args*, ***kws*)
Bases: wx._controls.SpinCtrl

font: wx.Font()

setFont(*font*)

setFontColor(*fontcolor*)

setFontFaceName(*facename*)

setFontSize(*fontsize*)

class **MyStaticText**(*parent*, *font=None*, *fontsize=10*, *fontcolor='black'*, *fontweight=90*, **args*, ***kws*)
Bases: wx._controls.StaticText

setFont(*font*)

setFontColor(*fontcolor*)

setFontFaceName(*facename*)

setFontSize(*fontsize*)

class **MyTextCtrl**(*parent*, *font=None*, *fontsize=10*, *fontcolor='black'*, **args*, ***kws*)
Bases: wx._controls.TextCtrl

setFont(*font*)

setFontColor(*fontcolor*)

setFontFaceName(*facename*)

setFontSize(*fontsize*)

class **ProgressBarFrame**(*parent*, *title*, *range=100*, **args*, ***kws*)
Bases: wx._windows.Frame

createProgressbar()

onTickTimer(*event*)

class **SaveData**(*data*, *fname*, *type*, *app='imageviewer'*)
Bases: object

onDataProcess()

onSaveASC()

onSaveHDF5()

onSaveSDDS()

class **ScanDataFactor**(*z*, *scannum*, *shotnum*, *ndim=2*)
Bases: object

getXavg()

getXerrbar()

getYavg()

getYerrbar()

setData(*z*)

show()

class ScanPlotPanel(*parent*, *data=None*, ***kws*)
Bases: felapps.utils.funutils.AnalysisPlotPanel

adjustErrbar(*err*, *x*, *y*, *x_error*, *y_error*)

clear_pick_pt()
clear picked points (which are picked for retaking)

get_edit_obj()

get_errorbar_line()

get_errorbar_mks()

get_fit_line()

get_fit_model()

get_fit_result()

get_mean_line()

get_pick_pt()
return picked points dict {k:v}, k: index, v: drawing artist

hide_fit_line()

on_motion(*event*)

on_pick(*event*)

on_press(*event*)

on_release(*event*)

repaint()

set_fit_line(*point_num=200*, ***kws*)
apply fitting model to average curve

set_fit_model(*model='gaussian'*, ***kws*)

set_grid()

set_legend(***kws*)

set_line_id(*line='Average Curve'*)
selected current editable line, 'Average Curve': curve of mean value of every iteration 'Errorbars' : errorbars, x and y 'Fitting Curve': fitting curve of average curve

set_linecolor(*color*)

set_linestyle(*ls*)

set_linewidth(*lw*)

set_marker(*mk*)

set_mec(*color*)

set_mew(*mew*)

set_mfc(*color*)

set_mks(*mks*)

set_text(*text*, ***kws*)
set fitting result output as a Text object :param text: fitting result text from FitModels class available keys of kws:, x, y, fontsize

set_title(***kws*)

set_xlabel(***kws*)

update_text(***kws*)
available keys of kws:, x, y, fontsize

aupmu(*gap*, *xlamd*, *a=3.44*, *b=-5.0*, *c=1.54*)
gap, xlamd: [mm]

color_to_hex(*c*)
convert matplotlib colors into hex string format, e.g. 1 color_to_hex('r') = '#FF0000' 2 color_to_hex('red') = '#FF0000' 3 color_to_hex('#FF0000') = '#FF0000'

createwxButton(*parent*, *label*, *fontname=17*, *fontsize=10*, *fontweight=90*, *fontcolor='black'*, *size=wx.Size(-1, -1)*)

createwxPanel(*parent*, *backgroundcolor=None*, *id=-1*)

createwxStaticBox(*parent*, *label=''*, *style=0*, *fontname=17*, *fontsize=10*, *fontweight=90*, *fontcolor='black'*)

createwxStaticText(*parent*, *label*, *size=wx.Size(-1, -1)*, *style=0*, *fontname=17*, *fontsize=10*, *fontweight=90*, *fontcolor='black'*)

createwxTextCtrl(*parent*, *value=''*, *style=0*, *fontname=17*, *fontsize=10*, *fontweight=90*, *fontcolor='black'*)

findObj(*objroot*, *objclass*)
Find all the objects that is instance of objclass belongs to objroot. obj

func_peaks(*x*, *y*)

func_sinc(*x*, *y*)

gaussian_fit(*x*, *xdata*, *mode='full'*)
return fit result and fitmodels :param x: data to fit, x col, numpy array :param xdata: data to fit, y col, numpy array

getFileToLoad(*parent*, *ext='*'*, *flag='single'*)

getFileToSave(*parent*, *ext='*'*)

getResPath(*filename*, *cwd='.'*, *resdir='../resources'*)
return absolute path for resources, e.g. images, data :param reshead : the relative path for resources dir :param filename: filename for images or data

get_file_info(*filepath*)
return file information :param filepath: file full path name

get_randstr(*length=1*)
return string of random picked up chars :param length: string length to return

get_range(*x*, *xmin*, *xmax*)
find array range, :param x: orignal numpy array, 1d :param xmin: x min of range :param xmax: x max of range return range_index and array

handleConfig(*config_name='imageviewer.xml'*)

handle configuration files issues: 1 load configuration files from app data 2 create user specific configuration files

Parameters **config_name** – configuration file name, by default is 'imageviewer.xml' (for app Image Viewer)

reutrn valid configuration file at default user location

hex2rgb(*hex_string*)
convert hexadecimal color into rgb form. :param hex_string: hex color string, e.g. white color: '#FFFFFF'

Example: >>> hex2rgb('#FFAABB') (255, 170, 187)

importCheck(*moduleName*)
check could import moduleName or not

invert_color_hex(*hex_color_str*)
invert hex colors, e.g. invert_color_hex('#FFFFFF') = '#000000'

pick_color()

r56chi(*gam0*, *ibfield*, *imagl=0.15*, *idril=0.285*)
return r56 of chicane, ibfield: [T]

readfld(*filename*, *ncar=121*)

rescaleImage(*image0*, *scaledFac*)
Jan. 28, 2105 rescale image, given the image full path on disk

rgb2hex(*rgb_tuple*)
convert color rgb into hex form. :param rgb_tuple: tuple of rgb color, e.g. white color: (255, 255, 255)

Example: >>> rgb2hex((255, 170, 187)) u'ffaabb'

setPath(*pathstr*)

set_staticbmp_color(*obj*, *color*)
obj: staticbitmap, bitmapbutton color: could be returned by pick_color()

felapps.utils.imageutils module

classes and functions for image processing data2Image: generate image from data file, hdf5, asc, dat

Author: Tong Zhang Created: Sep. 28, 2015

class **ImageGalleryPanel**(*parent*, **args*, ***kwargs*)
Bases: `wx.lib.scrolledpanel.ScrolledPanel`

Panel for image gallery show

genImage(*imagelist*, *fdir*, *ftype*)

get_workspace(*fmt='dvc'*)
_workspace_list element format: "image_file_full_path;;;data_file_fullpath"

initUI()

onClear()
clear all objs on panel

onLeftClick(*event*)
when left click image, select corresponding item on workspace panel

onPopAdd(*event*, *obj*)

onPopAddAll(*event*, *obj*)

onPopAnalysis(*event*, *obj*)

onPopRemove(*event*, *obj*)

onPopRemoveAll(*event*, *obj*)

onRightClick(*event*)

onRightPopup(*event*)

onScaleDec(*percent*, *fdata_list*, *image_list*)
shrink image by percent

onScaleImageSize(*fdata_list*, *image_list*)
just rescale image size to show on image grid :param fdata_list: data file list, fullpath :param image_list: image file list, fullpath

onScaleInc(*percent*, *fdata_list*, *image_list*)
enlarge image by percent

onUpdate(*image_list=None*, *fdir=None*, *ftype=None*)
update image grid panel :param image_list: image file list, jpg :parma fdir: dirname of data files :param ftype: ext of data format, which generate image_list

class ProgressBarFrame(*parent*, *title*, *range=100*)
Bases: `wx._windows.Frame`

createProgressbar()

onTickTimer(*event*)

class WorkspacePanel(*parent*, *data=None*)
Bases: `wx._windows.Panel`

clear()

set_data(*data*)

update(*data*)

data2Image(*filename*, *datatype='hdf5'*, *figtype='jpg'*, *wdir=None*, *width=None*, *height=None*, *whflag='h'*, *cmtype='hot'*, **args*, ***kwargs*)
generate image thumbnails from data file, by default from hdf5 array.

Parameters

- **filename** – data filename, f.
- **datatype** – data format, 'hdf5' or 'h5': image data could be extracted by fid = h5py.File(f); fid['image']['data']. 'asc' or 'dat': image data could be extracted by np.loadtxt(f).
- **figtype** – 'jpg', 'png' or others.
- **wdir** – working directory, to put generated jpg figures, if None, use cwd.
- **width** – image size in w, if None, take original values.
- **height** – image size in h, if None, take original values.
- **whflag** – 'w' (h prop with w) or 'h' (w prop with h) or 'None' (take size as w x h).
- **cmtype** – colormap type, hot by default.

main()

resizeImage(*imagepath*, *height*, *width=None*, *whflag='h'*, *quality=0*)
resize image read from file, return wx.Image obj :param imagepath: jpg image file path :param height: height in pixel after resizing :param width: width in pixel after resizing :param whflag: w.r.t. height ('h') or width ('w')

test_data2Image()
test function Hdf2Image

felapps.utils.matchutils module

python modules for beamline matching utilities: matchwizard: main GUI framework for beamline matching

Author: Tong Zhang Created: Sep. 23rd, 2015

class LatVisPanel(*parent*, *figsize*, *dpi*, *bgcolor*, ***kwargs*)
Bases: felapps.utils.pltutils.ImagePanelxy

clear()

refresh()

visBeamline(*zoomfac=1.5*)

class MatchWizard(*parent*, *config='config.xml'*, *size=(1000, 750)*, *appversion='1.0'*, ***kwargs*)
Bases: wx._windows.Frame

createMenubar()

createPanel()

createStatusbar()

exitApp()

initUI()

onAbout(*event*)

onCheckUserDefinedBL(*event*)

onChooseBL(*event*)

onConfigApps(*event*)

onConfigLoad(*event*)

onConfigSave(*event*)

onExit(*event*)

onMenuHL(*event*)

onOpen(*event*)

onSave(*event*)

onTickTime(*event*)

onVisLattice(*event*)

postInit()

preInit()

felapps.utils.miscutils module

class AppVersions

getVersion(*appName='imageviewer'*)

setVersion(*verNum*, *appName='imageviewer'*)

felapps.utils.myui module

class PlotFrame(*parent*)
Bases: wx._windows.Frame

cmap_cbOnCombobox(*event*)

crange_tcOnTextEnter(*event*)

exit_btnOnButtonClick(*event*)

grid_ckbOnCheckBox(*event*)

imhide_tgbtnOnToggleButton(*event*)

lc_btnOnButtonClick(*event*)

lineid_choiceOnChoice(*event*)

ll_originOnRadioButton(*event*)

lr_originOnRadioButton(*event*)

ls_cbOnCombobox(*event*)

lw_tcOnTextEnter(*event*)

m_splitterOnIdle(*event*)

mec_btnOnButtonClick(*event*)

mfc_btnOnButtonClick(*event*)

mk_cbOnCombobox(*event*)

ms_tcOnTextEnter(*event*)

mticks_ckbOnCheckBox(*event*)

ul_originOnRadioButton(*event*)

ur_originOnRadioButton(*event*)

felapps.utils.parseutils module

Author: Tong Zhang Created Time: 10:22, Sep. 17, 2015

class ConfigFile(*infilename='config.xml'*, **args*, ***kwargs*)
Bases: object

Class to resolve parameters parsing by applying xmlparser approach.

getConfigs()

parseConfigs()

updateConfigs(*params_dict*, *savetofile=None*)

class ParamParser(*inifilename='config.ini'*, **args*, ***kws*)
Bases: object

Class to resolve parameters parsing by applying ConfigParser approach.

createTemplate(*configfilename='config_sample.conf'*)

dumpDictToConfig(*newhierdict*, *configfilename*)

makeFlatDict()
return dict with key,value pairs

makeHierDict()
return dict with hierarch structure

readConfig()

saveConfig(*filetosave=None*)

setAllParams(*newhierdict*)

setOneParam(*section_name*, *option_name*, *newvalue*)

loadtest()

savetest()

felapps.utils.pltutils module

python modules for plot utilities: ImageViewer: main GUI framework for an universal image viewer

Author: Tong Zhang Created: Feb. 3rd, 2015

class AppConfigPanel(*parent*, ***kwargs*)
Bases: wx._windows.Frame

InitUI()

createMenu()

createNotebooks()

createStatusbar()

onApplyData(*event*)

onCancelData(*event*)

onUpdateData(*event*)

setParams()

class AutoSavePanel(*parent*, ***kwargs*)
Bases: wx._windows.Frame

InitUI()

createPanel()

onCancel(*event*)

onChoosePath(*event*)

onStart(*event*)

postInit()

class ChooseROIFrame(*parent*, *imgsrcptn*, ***kwargs*)
Bases: wx._windows.Frame

InitUI()

createMenu()

createPanel(*imgsrcptn*)

createStatusbar()

onCancel(*event*)

onGetROI(*event*)

onMotion(*event*)

onPress(*event*)

onRelease(*event*)

class ChooseROIPanel(*parent*, *figsize*, *dpi*, ***kwargs*)
Bases: wx._windows.Panel

doPlot(*imgsrcptn*)

onSize(*event*)

setColor(*rgbtuple=None*)
Set figure and canvas colours to be the same.

class ConfigNoteBook(*parent*, *style=64*, **args*, ***kws*)
Bases: wx._controls.Notebook

MakePages()

class ControlConfigPanel(*parent*, ***kwargs*)
Bases: wx._windows.Panel

createPanel()

initConfig()

initUI()

onCheckAuto(*event*)

onChooseLib(*event*)

onUpdateParams1(*event*)

onUpdateParams2(*event*)

class DebugPanel(*parent*, ***kwargs*)
Bases: wx._windows.Frame

InitUI()

createPanel()

class FitPlotFrame(*parent*, *model_x*, *model_y*, ***kwargs*)
Bases: wx._windows.Frame

onAutoScale(*event*)

onExit(*event*)

onExport(*event*)

onSetRange(*event*)

set_datalim(*nid*, *range_list*)

class FitPlotPanel(*parent*, *figsize*, *dpi*, *bgcolor*, *fitmodel_x*, *fitmodel_y*, ***kwargs*)
Bases: felapps.utils.pltutils.ImagePanelxy

doPlot()

onGetData()

refresh_draw()

repaint(*new_fitmodel_x*, *new_fitmodel_y*)

update_figure()

update_fit_modules(*new_fitmodel_x*, *new_fitmodel_y*)

class HistPlotConfigPanel(*parent*, ***kwargs*)
Bases: wx._windows.Panel

createPanel()

initConfig()

initUI()

class ImageColorMap(*parent*, *figsize*, *dpi*, *bgcolor*, ***kwargs*)
Bases: wx._windows.Panel

doPlot()

fitCanvas()

onGetData()

onSetCr(*crange*)

onSetcm(*cmap*)

onSize(*event*)

repaint()

setColor(*rgbtuple=None*)
Set figure and canvas colours to be the same.

class ImageConfigFile(*infilename='config.xml'*, **args*, ***kwargs*)
Bases: felapps.utils.parseutils.ConfigFile

parseConfigs()

class ImageConfigPanel(*parent*, ***kwargs*)
Bases: wx._windows.Panel

createPanel()

initConfig()

initUI()

onChooseDirpath(*event*)

onUpdateParams(*event*)

class **ImagePanel**(*parent*, *figsize*, *dpi*, *bgcolor*, *heightratio=0.4*, *func='peaks'*, ***kwargs*)
Bases: wx._windows.Panel

doPlot()

draw_hvlines(*x0*, *y0*)

fitCanvas()

onConfigPlot()

onGetData()

onMotion(*event*)

onPress(*event*)

onRelease(*event*)

onSetCr(*crange*)

onSetHist()

onSetcm(*cmap*)

onSize(*event*)

refresh()

repaint()

setColor(*rgbtuple=None*)
Set figure and canvas colours to be the same.

setHratio(*hratio*)

set_markflags(*mk1=False*, *mk2=False*)

set_mkc1(*color*)

set_mkc2(*color*)

set_pcc(*color*)

update_deltxy()

class **ImagePanelxy**(*parent*, *figsize*, *dpi*, *bgcolor*, ***kwargs*)
Bases: wx._windows.Panel

doPlot()

fitCanvas()

onConfigPlot()

onGetData()

onMotion(*event*)

onPress(*event*)

onRelease(*event*)

onSize(*event*)

onUpdatePlot()

repaint()

setColor(*rgbtuple=None*)
Set figure and canvas colours to be the same.

class ImageViewer(*parent*, *config='config.xml'*, *size=(800, 600)*, *appversion='1.0'*, ***kwargs*)
Bases: wx._windows.Frame

InitUI()

createMenubar()

createPanel()

createStatusbar()
self.statusbar = self.CreateStatusBar(2) self.statusbar.SetStatusWidths([-4, -1]) self.statusbar.SetStatusText(u'ImageViewer powered by Python' ,0) versionfield = 15*' ' + time.strftime('%Y-%m-%d', time.localtime()) + ' ' + '(Version: ' + self.appversion + ')' self.statusbar.SetStatusText(versionfield, 1)

loadConfig(*configfilename*)

onAbout(*event*)

onAddPV(*event*)

onAutoSave(*event*)

onBookmark(*event*)

onCheckRCM(*event*)

onChooseROI(*event*)

onConfigApps(*event*)

onConfigLoad(*event*)

onConfigSave(*event*)

onDAQbtn(*event*)

onDebug(*event*)

onExit(*event*)

onFitPopup(*event*)

onMenuHL(*event*)

onOpen(*event*)

onPickMK1c(*event*)

onPickMK2c(*event*)

onPickPcc(*event*)

onResetROI(*event*)

onRmPV(*event*)

onSaveDat(*event*)

onSaveImg(*event*)

onSaveTimer(*event*)

onSetCmclass(*event*)

onSetColorRange(*event*)

onSetColormap(*event*)

onSetImgSrc(*event*)
set image data source and show in the image panel

onShowInt(*event*)

onShowXhist(*event*)

onShowYhist(*event*)

onTickTime(*event*)

onUnBookmark(*event*)

onUpdate(*event*)

onUpdateUI()

onUpdateUIInit()

printConfig()

setEnvars()

updateFont()

class ShowIntPanel(*parent*, ***kwargs*)
Bases: wx._windows.Frame

InitUI()

createMenu()

createPanel()

createStatusbar()

onSaveFigure(*event*)

onTimerControl(*event*)

onUpdate(*event*)

class StyleConfigPanel(*parent*, ***kwargs*)
Bases: wx._windows.Panel

createPanel()

initConfig()

initUI()

onChooseColor(*event*)

onChooseFont(*event*)

main()

felapps.utils.resutils module

felapps.utils.scanutils module

Author: Tong Zhang Created Time: 09:26, Jun. 29, 2015

Scan utilities: (p.s.) separated from felutils.py 1 correlation analyzer: 1.1: ScanAnalyzer

class ImagePanel(*parent*, *figsize*, *dpi*, *bgcolor*, ***kwargs*)
Bases: felapps.utils.pltutils.ImagePanel

doPlot()

onGetData()

onMotion(*event*)

onPress(*event*)

repaint()

class ImagePanelxy(*parent*, *figsize*, *dpi*, *bgcolor*, ***kwargs*)
Bases: felapps.utils.pltutils.ImagePanelxy

adjustErrbar(*err*, *x*, *y*, *x_error*, *y_error*)

doPlot()

onConfigPlot()

onGetData()

onMotion(*event*)

onPress(*event*)

repaint()

class ScanAnalyzer(*parent*, *size=(800, 600)*, *appversion='1.0'*, ***kwargs*)
Bases: wx._windows.Frame

createMenubar()

createPanel()

createStatusbar()

exitApp()

initUI()

onAbout(*event*)

onCheckFitting(*event*)

onCheckScan(*event*)

onCheckScan2(*event*)

onChooseFont(*event*)

onChoosePath(*event*)

onConfigApps(*event*)

onConfigLoad(*event*)

onConfigSave(*event*)

onDEBUG1(*event*)

onExit(*event*)

onMenuHL(*event*)

onOpen(*event*)

onProfDAQ(*event*)

onPushClose(*event*)

onPushPause(*event*)

onPushProfDAQ(*event*)

onPushRetake(*event*)

onPushStart(*event*)

onSave(*event*)

onScanDAQ(*event*)

onScanfigClear(*event*)

onScanlogClear(*event*)

onSetDaqFreq(*event*)

onSetImgCM(*event*)

onSetImgCMR(*event*)

onSetImgCR(*event*)

onSetImgPV(*event*)

onSetProFreq(*event*)

onSetShotNum(*event*)

onSetWaitTime(*event*)

onTickTime(*event*)

onUpdateScan(*event*)

postInit()

postSetImage()

postSetScan()

preInit()

setScanParams()

startScanDAQ(*ms*, *scanidx*)

updateFont(*font*)

updateImage()

updateScanfig()

class ScanConfigFile(*infilename='.cornalyzer.conf'*, **args*, ***kwargs*)
Bases: `felapps.utils.parseutils.ConfigFile`

parseConfigs()

class ScanThread(*target*)
Bases: `threading.Thread`

Thread do scan routine, as well as show the progress bar.

Dated: Jul. 1, 2015

run()

felapps.utils.uiutils module

custom GUI controls

Tong Zhang 2016-06-19 12:37:40 PM CST

class **CheckListCtrl**(*parent*, *log*)
Bases: wx._controls.ListCtrl, wx.lib.mixins.listctrl.CheckListCtrlMixin

OnItemActivated(*event*)

class **EditFrame**(*parent*, *init_string=None*)
Bases: wx._windows.Frame

on_cancel(*event*)

on_ok(*event*)

class **EditListFrame**(*parent*, *string_list=None*, *label=None*)
Bases: wx._windows.Frame

on_cancel(*event*)

on_ok(*event*)

class **LatticePlotPanel**(*parent*, ***kwargs*)
Bases: felapps.utils.uiutils.MyPlotPanel

identify_obj(*x*)

on_motion(*event*)

class **MyPlotPanel**(*parent*, *figsize=None*, *dpi=None*, *bgcolor=None*, *type=None*, *toolbar=None*, *aspect='auto'*, ***kwargs*)
Bases: wx._windows.Panel

fit_canvas()
tight fit canvas layout

maxlim_tcOnTextEnter(*event*)

minlim_tcOnTextEnter(*event*)

on_motion(*event*)

on_pick(*event*)

on_press(*event*)

on_release(*event*)

on_size(*event*)

refresh()

set_color(*rgb_tuple*)
set figure and canvas with the same color. :param rgb_tuple: rgb color tuple,
e.g. (255, 255, 255) for white color

set_layout()
set panel layout

xylim_choiceOnChoice(*event*)

class **MyToolbar**(*canvas*)
Bases: matplotlib.backends.backend_wxagg.NavigationToolbar2WxAgg

class **TestFrame**(*parent*, ***kwargs*)
Bases: wx._windows.Frame

test()

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

f

A

C

D

E

F

G

H

I

L

M

O

P

R

S

T

U

V

W

X