

Data processing software for RHESSI and STIX instruments

Technical Design Document

**OVERVIEW**

Object Spectral Executive (OSPEX) is an object-oriented interface for X-ray spectral analysis of solar data written in IDL [1].

Our software is oriented to reconstruct some elements of OSPEX package and provide a tool to process the RHESSI data (Reuven Ramaty High Energy Solar Spectroscopic Imager, NASA Small Explorer Mission).

In addition to this, it is planned to include observed data from the X-ray Spectrometer(STIX) aboard Solar Orbiter(launch date is February 2020).

The project is presented on GitHub repository [2].

**PURPOSE OF THIS DOCUMENT**

This document is a Technical Design Document for use OSPEX written in Python. It provides guidance and materials which are intended to assist the relevant management or technical staff. It is also useful background reading for anyone involved in developing or using the application.

**SOFTWARE DEVELOPMENT TOOLS**

This section lists the tools chosen to assist software development including testing.

Recommendations tools:

*Python 3.6*

*Libraries:*

*Astropy*

*Numpy*

*Pandas*

*Matplotlib*

*Tkinter*

Following modules should come by default with Python after installation but we recommend you to check if they work correctly. If not, please, import them manually:

*Webbrowser*

*Datetime*

*Re*

*Copy*

*Importlib*

To run:

*Linux, Mac, Windows:* When imported the libraries you should call main script from terminal*:*

1. Open the terminal*:* ***cmd***

2. Go to the directory where your program is located: ***cd*** *directory location*

3. Run the software by calling the main script*:* ***python3 main.py***

If you want to work with content and make modifications:

1. Go to the directory where your program is located:

*cd directory location*

2. Call Python:

*python3*

3. Upload file content from the plotting.py script :

*from plotting import \**

4. Choose desired file(s), "Data" folder:

*File = Input(“filename.fits”)*

Now you are able to call any parameters and functions of the .fits file

5. Some examples:

a) to load the parameter RATE from DATA:

*File.rate*

b) to plot Spectrum for Flux:

*File.plot\_spectrum\_flux()*

**CLASSES, COMPONENT DESCRIPTION AND ACRONYMS**

For the software implementation and test, this and the previous section provide sufficient information for a programmer to produce the software, and for a user, who is maybe not a developer.

The logical and physical structure of the application and components will be defined in detail:

* meaning of the classes;
* description of each parameter;
* relationships between the elements, i.e. the structure;
* initial values of each element.

|  |  |
| --- | --- |
| ***Classes*** | |
| SecondWindow | Used to describe the procedures to create an interface for Select Input part |
| Input | Provides the methods to load the parameters from input data and plot Spectrum, Time Profile and Spectrogram |
| Fitting | Performs a spectrum fitting. Creates a new window called “SPEX Fit Options” |
| BackgroundWindow | A window to edit time intervals and create background plots |
| backround\_plot | Main function to create background plots |
| ***Component description*** | |
| Rate | Count Rate data in each energy channel |
| Time | Array of Low energy ranges |
| Time\_del | Accumulation time |
| E\_min | Array of Low energy ranges |
| E\_max | Array of High energy ranges |
| deltaE | Energy range from E\_min to E\_max(3-250 keV) |
| TimeNew | Array of converted time from sec to hours:min:sec |

Additional information can also be found in the code comments.

**FUNCTIONS TO FIT AND INPUT PARAMETERS**

**Category: spectral fitting**

1. One dimensional Power Law.

Parameters:

* amplitude – model amplitude at the reference energy;
* x – reference energy;
* alpha – power law index.

2. Broken Power Law function - One dimensional power law model with a break.

Parameters:

* amplitude – model amplitude at the break energy;
* x\_break – break energy;
* alpha 1 – power law index for x<x\_break;
* alpha 2 – power law index for x>x\_break.

3. One dimensional Gaussian model.

Parameters:

* amplitude – amplitude of the Gaussian;
* mean – mean of the Gaussian;
* stddev – standard deviation of the Gaussian.

4. 1D Polynomial model.

Parameters:

* degree – degree of the series.

5. Exponential function.

Parameters:

* t0 – normalization;
* t1 – pseudo temperature.

6. Single power – law times an exponential.

Parameters:

p - first 3 parameters describe the single power – law function, e - exponential;

* p0 – normalization at epivot for power – law;
* p1 – negative power - law index;
* p2 – epivot (keV) for power – law;
* e3 – normalization for exponential;
* e4 – pseudo temperature for exponential.

**CONVERTING BETWEEN COUNTS AND PHOTONS**

The input data (at least from a RHESSI spectrum FITS file) is in counts.  The fit functions (model) calculate photon flux.  Whenever you plot the data in photons, or the model in counts, a conversion between counts and photons is being calculated.

The efficiency factors (or conversion factors) used to convert counts to photons depends on both the response matrix and a model.  The efficiency factors are the ratio of the model count spectrum, which is the model photon spectrum folded through the detector response, to the model photon spectrum [1].

The method that we will use is called forward folding. Basically the process is:

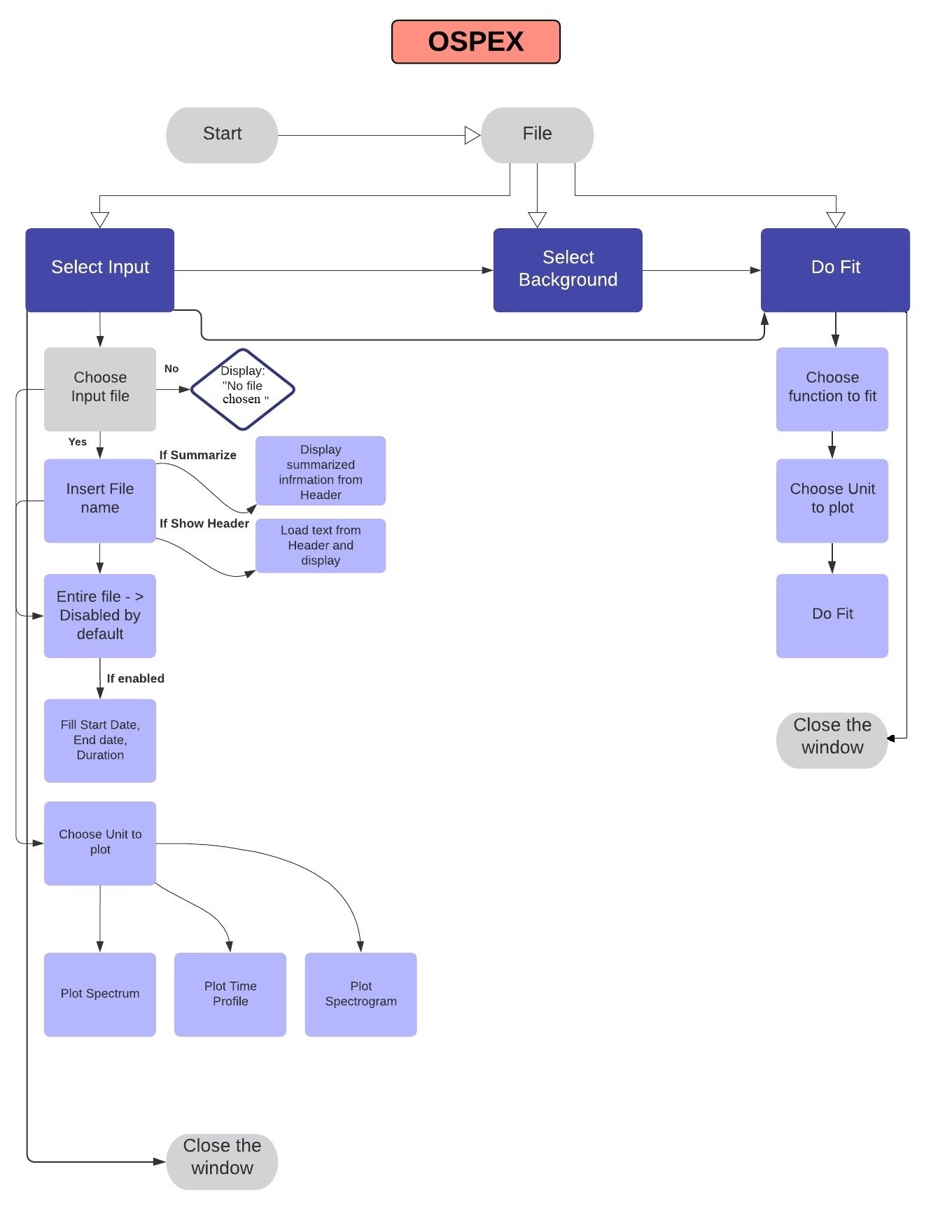
* assume a photon model spectrum with a set a variable parameters
* convolve this spectrum with the instrument response to produce counts in the instruments energy space
* compare these trial counts with the observed counts statistically
* iterate until a convergence criterion is met.

Let’s consider the implementation based on the model of power law function.

First we will define a power law differential flux function. Then its integral via Simpson’s rule. Finally we should calculate the photon fluxes of the photon model for each photon energy bound in the response matrix. These will be the “true” fluxes of the assumedphoton model.

We still have a differential area in our flux and have not accounted for dispersion. Thus, we must convolve these true fluxes with our matrix. The operation is a dot product between the vector of photon fluxes we just computed and the matrix. In this way we can compare our assumed spectrum with the real data.

**APPLICATION ARCHITECTURE**



**REFERENCES:**

[1] <https://hesperia.gsfc.nasa.gov/ssw/packages/spex/doc/ospex_explanation.htm>

[2] <https://github.com/LAbdrakhmanovaOBSPM/OSPEX-Object-Spectral-Executive-in-Python>