# Contents

1 Features 3

2 Table of Contents 5
   2.1 About ............................................. 5
   2.2 Installation Guide ................................. 5
   2.3 Savu User Guide ................................... 6
   2.4 Savu Developer Guide ............................. 14
   2.5 API Documentation ................................ 21

3 Indices and tables 109

   Python Module Index 111

   Index 113

i
Savu is a Python package to assist with the processing and reconstruction of parallel-beam tomography data. The project originated in the Data Analysis Group at the Diamond Light Source (UK synchrotron) to address the growing, and increasingly complex, needs of the tomography community.

Designed to allow greater flexibility in tomography data processing, Savu is capable of processing N-dimensional full-field tomography and mapping tomography data, along with concurrent processing of multiple datasets such as those collected as part of a multi-modal setup. Savu process lists, tailored to a specific experiment and passed to the framework at runtime along with the data, detail the processing steps that are required.

A Savu process list is created using the Savu configurator tool, which stacks together plugins chosen from a repository. Each plugin performs a specific independent task, such as correction, filtering, reconstruction. For a list of available plugins see plugin API.

Savu is currently in use across the tomography beamlines at Diamond to reconstruct both full-field tomography data and multi-modal, mapping tomography data.
CHAPTER 1

Features

• Full-field and mapping tomography data processing
• multi-modal data processing
• Absorption, fluorescence, diffraction and ptychography data processing
• Handles N-dimensional data and multiple datasets
• Supports multiple data formats
• Runs in serial or parallel on your local machine
• Runs in parallel across a cluster
• Supports very large data processing with parallel HDF5 (not limited by RAM)
• Allows flexible data slicing (e.g. alternate between projection and sinogram processing)
• Plugin architecture with CPU and GPU plugins
• Processing tailored to a specific experimental setup
• Easy integration of new functionality
2.1 About

The project is named ‘Savu’, after a python subspecies known for their small size, good temperament, easy feeding habits and tolerance for a wide range of temperatures. [1]

The Savu Tomography reconstruction pipeline project aims to mimic these behaviours being a small package, which is easy to use and reliable, chomps its way through vast amounts of data, and finally is portable to a wide range of systems.

Be aware though, the following is also true “Savu Pythons are typically calm in disposition, and generally tolerate gentle handling. Like all snakes, however, care must be exercised when working around them.” [2]

2.2 Installation Guide

2.2.1 Table of Contents

Installing Savu

Requirements:

- An MPI implementation (tested with openmpi 3.1.4)
- fftw (tested with 3.3.7)
- cuda (tested with 9.2)

1. Download the latest version of savu and extract.
2. Run the following command and follow the installation instructions:

```bash
bash savu_v2.4/savu_installer.sh
```
3. Check the log file /tmp/<tmpfolder>/savu_error_log.txt for installation errors (correct log file path printed to screen during installation process).

2.3 Savu User Guide

Description of who can use Savu (internal and external users) and how it can be used (cluster or single-threaded).

2.3.1 Savu Diamond User Guide

Introduction

Tomography data collected at Diamond has, in recent years, been processed using the Tomo Recon GPU cluster-based code available through DAWN. A steady increase in the popularity of tomographic imaging, due to improvements in data acquisition and computer technology, has led to a broadening of the range of tomographic experiments, and their complexity, across multiple fields.

In full-field tomography, where the whole region-of-interest is irradiated by the beam simultaneously, time-resolved imaging is becoming increasingly popular. In mapping tomography, where a thin beam of X-rays is translated and rotated across the region of interest, multi-modal data collection is common and incorporates a variety of measurements, such as X-ray absorption, diffraction and fluorescence.

This wide range of experimental requirements leads to a wider range of software processing requirements. Savu, developed in the Data Analysis Group at Diamond Light Source Ltd., is the new tomography data processing tool that has been developed to allow greater flexibility in tomography data processing. Custom process lists are passed to Savu at runtime to enable processing to be tailored to a specific experimental setup. The framework is capable of processing multiple, n-dimensional, very large datasets, and is written in Python to allow easy integration of new functionality, allowing researchers and beam line staff greater flexibility in integrating new, cutting-edge processing techniques.

A quick comparison of the old and new tomography software is given in the table below.

<table>
<thead>
<tr>
<th></th>
<th>Tomo Recon</th>
<th>Savu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data type</td>
<td>Full-field tomography data</td>
<td>Full-field and mapping tomography data</td>
</tr>
<tr>
<td>Data dimensions</td>
<td>3-D</td>
<td>N-D</td>
</tr>
<tr>
<td>Data format</td>
<td>Nxtomo NEXUS format</td>
<td>Multiple formats (any possible)</td>
</tr>
<tr>
<td>Output format</td>
<td>tiff</td>
<td>Multiple formats (hdf5 - tiff coming soon)</td>
</tr>
<tr>
<td>Data size</td>
<td>Limited by RAM</td>
<td>No RAM limit (uses parallel hdf5)</td>
</tr>
<tr>
<td>Datasets per run</td>
<td>One dataset</td>
<td>Multiple datasets</td>
</tr>
<tr>
<td>Data slicing</td>
<td>Sinogram only</td>
<td>Flexible (e.g sinogram/projection)</td>
</tr>
<tr>
<td>Processing</td>
<td>Fixed: correction, ring removal, FBP</td>
<td>Custom: Tailored process lists</td>
</tr>
<tr>
<td>New functionality</td>
<td>No integration</td>
<td>Easy integration</td>
</tr>
</tbody>
</table>

Process lists

Savu is a framework that does nothing if you run it on its own. It requires a process list, passed to it at runtime along with the data, to detail the processing steps it should follow. A Savu process list is created using the Savu configurator tool, which stacks together plugins chosen from a repository. Each plugin performs a specific independent task, such as correction, filtering, reconstruction. For a list of available plugins see plugin API.

Plugins are grouped into categories of similar functionality. Loaders and savers are two of these categories and each process list must begin with a loader plugin and optionally end with a saver plugin (hdf5 is the default), with at least one processing plugin in-between. The loader informs the framework of the data location and format along with
important metadata such as shape, axis information, and associated patterns (e.g. sinogram, projection). Therefore, the choice of loader is dependent upon the format of the data.

**Note:** Savu plugins can run on the CPU or the GPU. If you are running the single-threaded version of Savu and you don’t have a GPU you will be limited to CPU plugins.

Example: View a process list in the Savu configurator.

```plaintext
>>> module load savu
>>> savu_config
>>> help  # show the available commands
>>> list  # list the available plugins
>>> open /dls/science/groups/das/SavuTraining/process_lists/simple_tomo_pipeline.nxs
   # open a process list
>>> disp -v  # view parameter descriptions
>>> disp -v -a  # view hidden parameters
>>> exit  # exit the configurator
```

**Note:** The process lists created by the configurator are in NeXus (.nxs) format (http://www.nexusformat.org/).

For examples of how to create and amend process lists see 5. Create a process list and 4. Amend a process list.

**Running Savu**

To run Savu you require a data file and a process list (link to process list).

```plaintext
>>> module load savu
```

To run Savu across the cluster (in parallel):

```plaintext
>>> savu_mpi <data_path> <process_list_path> <output_folder> <optional_args>
```

To run Savu on your local machine (single threaded):

```plaintext
>>> savu <data_path> <process_list_path> <output_folder> <optional_args>
```

The full list of aliases provided with `module load savu` is given below:

<table>
<thead>
<tr>
<th>Alias</th>
<th>Description</th>
<th>Required input parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>savu</td>
<td>Run single threaded Savu</td>
<td>&lt;data_path&gt; &lt;process_list_path&gt; &lt;output_path&gt;</td>
</tr>
<tr>
<td>savu_mpi</td>
<td>Run mpi Savu across the cluster</td>
<td>&lt;data_path&gt; &lt;process_list_path&gt; &lt;output_path&gt;</td>
</tr>
<tr>
<td>savu_mpi_preview</td>
<td>Run mpi Savu across 1 node (20 cores)</td>
<td>&lt;data_path&gt; &lt;process_list_path&gt; &lt;output_path&gt;</td>
</tr>
<tr>
<td>savu_config</td>
<td>Create or amend process lists</td>
<td></td>
</tr>
</tbody>
</table>

Optional arguments:

<table>
<thead>
<tr>
<th>short</th>
<th>long</th>
<th>argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-f</td>
<td>-folder</td>
<td>folder_name</td>
<td>Override the output folder name</td>
</tr>
<tr>
<td>-d</td>
<td>-tmp</td>
<td>path_to_folder</td>
<td>Store intermediate files in this (temp) directory</td>
</tr>
<tr>
<td>-l</td>
<td>-log</td>
<td>path_to_folder</td>
<td>Store log files in this directory</td>
</tr>
<tr>
<td>-v, -q</td>
<td>--verbose, --quiet</td>
<td>path_to_folder</td>
<td>Verbosity of output log messages</td>
</tr>
</tbody>
</table>
Note: Savu produces a hdf5 file for each plugin in the process list. It is recommended, if you are running Savu on a full dataset, to pass the optional argument `-d <tmp_dir>` where `tmp_dir` is the temporary directory for a visit.

Training Examples

Test data and process lists can be found in the directory `/dls/science/groups/das/SavuTraining` inside the data and process_lists directories respectively. Create a SavuTraining directory in your home directory and copy the data and process lists into this folder. First, open a terminal and follow the commands below:

```bash
>>> mkdir SavuTraining
>>> cd SavuTraining
>>> cp -r /dls/science/groups/das/SavuTraining/process_lists .
>>> cp -r /dls/science/groups/das/SavuTraining/data .
>>> module load savu
>>> python process_lists/refresh.py  # ensure the process lists are up-to-date with the current version of Savu.
```

1. Run a single-threaded Savu job on your local machine

View the simple_tomo_pipeline_cpu.nxs process list inside the configurator

```bash
>>> savu_config
>>> open process_lists/simple_tomo_pipeline_cpu.nxs
>>> disp -v
>>> exit
```

Run the single-threaded version of Savu with the data file `data/24737.nxs` and the process list `process_lists/simple_tomo_pipeline_cpu.nxs` and output to the current directory.

```bash
>>> savu <data_file> <process_list> .
```

2. Run a parallel Savu job on the cluster

View the simple_tomo_pipeline.nxs file in the configurator. Use the same data file as above, but this time use the `simple_tomo_pipeline.nxs` process list, which contains GPU processes.

The MPI version of Savu will run on the cluster.

```bash
>>> savu_mpi <data_file> <process_list> .
```

Re-run the mpi job but send the intermediate files to a temporary directory:

```bash
>>> savu_mpi <data_file> <process_list> . -d /dls/tmp
```

Note: `/dls/tmp` is for training purposes only and should not be used during a visit.
3. View the output data in DAWN

Once the run is complete, the current directory will contain all the output hdf5 files and the .nxs file that links the files together. Do not change the names of the files as this will break the link to the data. You can view the data in Dawn by opening the .nxs file (see In DAWN).

4. Amend a process list

The process list tomo_pipeline.nxs is a typical full-field tomography reconstruction pipeline. However, the experimental setup will determine which plugins should remain ‘on’ and what values the plugin parameters should take. Follow the list of commands below for some examples of what you can do.

Example 1

1. Open the process list.

```plaintext
>>> savu_config
>>> open process_lists/tomo_pipeline.nxs
```

2. View all available commands.

```plaintext
>>> help
```


```plaintext
>>> move 4 3
```

4. Turn the Paganin filter off (and turn the reconstruction log parameter back on).

```plaintext
>>> set 3 off
```

5. Display only the astra recon plugin with parameter descriptions.

```plaintext
>>> disp 6 -v
```

6. Turn the astra recon log parameter to True.

```plaintext
>>> mod 6.3 True
```

7. Apply previewing to reconstruct the middle 10 sinograms only (Previewing).

```plaintext
>>> mod 1.1 [:, mid-5:mid+6, :]
```

8. Manually entering centre of rotation (Sinogram centering).

```plaintext
>>> set 5 off
>>> mod 6.7 86
```

9. Save the process list and exit.

```plaintext
>>> save process_lists/test.nxs
>>> exit
```

Now run `savu_mpi_preview` with `data/24737.nxs` and the new process list `process_lists/test.nxs` and view the output in DAWN.
Example 2

1. Open the process list.

```plaintext
>>> savu_config
>>> open process_lists/test.nxs
```

2. Apply parameter tuning to centre value (parameter).

```plaintext
>>> mod 6.7 84:87:0.5;
```

3. Modify the reconstruction algorithm to CGLS_CUDA and increase iterations.

```plaintext
>>> disp 6 -v
>>> mod 6.6 CGLS_CUDA
>>> mod 6.4 10
```

4. Apply parameter tuning to Paganin Ratio parameter.

```plaintext
>>> set 3 on
>>> mod 6.3 False
>>> mod 3.1 50;100;200
```

5. Save the process list and exit.

```plaintext
>>> save process_lists/test2.nxs
>>> exit
```

Now run `savu_mpi_preview` with `data/24737.nxs` and the new process list `process_lists/test2.nxs` and view the output in DAWN.

5. Create a process list

Here is the list of commands used to create the process list `tomo_pipeline.nxs` used in the previous example.

Full pipeline with auto-centering

```plaintext
>>> savu_config # open the configurator
>>> add NxtomoLoader # add the loader plugin (use tab completion)
>>> add DarkFlatFieldCorrection # add the correction plugin
>>> add RavenFilter # add the ring artefact removal plugin
>>> add PaganinFilter # add contrast enhancement plugin
>>> add VoCentering # add auto-centering plugin
>>> add AstraReconGpu # add reconstruction plugin
>>> mod 6.3 False # don’t take the log of the data in recon (required by paganin)
>>> mod 5.1 [:, mid-5:mid+6, :] # apply centering to mid 10 sinograms only
>>> save tomo_pipeline.nxs # save the process list
>>> exit # exit the configurator
```
Apply previewing

```python
>>> savu_config # open the configurator
>>> open tomo_pipeline.nxs # open the full data process list
>>> mod 1.1 [:, mid-2:mid+3, :] # process the middle 5 sinograms only
>>> ref 5 -d # refresh auto-centering to default parameters
   ←(remove previewing)
>>> save tomo_pipeline_preview.nxs # save the process list
>>> exit # exit the configurator
```

Apply manual centering

```python
>>> savu_config # open the configurator
>>> open tomo_pipeline_preview.nxs # open the preview process list
>>> set 5 off # turn the auto-centering plugin off
>>> mod 6.7 86 # manually enter the centre value to the recon
>>> save tomo_pipeline_preview2.nxs # save the process list
>>> exit # exit the configurator
```

Apply parameter tuning to the centre of rotation

```python
>>> savu_config # open the configurator
>>> open tomo_pipeline_preview2.nxs # open the preview process list
>>> mod 6.7 85;85.5;86;86.5 # apply 4 different values to the centre of rotation
   ←param in the reconstruction
>>> save tomo_pipeline_preview3.nxs # save the process list
>>> exit
```

Special features

Previewing

Previewing enables the process list to be applied to a subset of the data. Each loader plugin has a preview parameter that is empty by default (apply processing to all the data). The preview requires a list as input with entries for each data dimension. Each entry in the preview list should be of the form start:stop:step:chunk, where stop, step and chunk are optional (defaults: stop = start + 1, step = 1, chunk = 1) but must be given in that order. For more information see `set_preview()`

Previewing Examples

The 3-D NxtomoLoader plugin maps the data dimensions (0, 1, 2) to the axis labels (rotation_angle, detector_y, detector_x) respectively.

```python
>>> savu_config
>>> add NxtomoLoader
>>> mod 1.1 [:, mid-5:mid+6, :] # process the middle 10 sinograms only
>>> mod 1.1 [0:end:2, mid-5:mid+6, :] # process every other projection
>>> mod 1.1 [0:end:2, mid-5:mid+6, 300:end-300] # crop 300 pixels from the sides of the detector
```
Sinogram centering

Automatic calculation OR manual input of the centre of rotation are possible in Savu.

Auto-centering

The auto-centering plugin (VoCentering) can be added to a process list before the reconstruction plugin. The value calculated in the centering routine is automatically passed to the reconstruction and will override the centre_of_rotation parameter in the reconstruction plugin. The auto-centering plugin is computationally expensive and should only be applied to previewed data. There are two ways to achieve this:

1. Apply previewing in the loader plugin to reduce the size of the processed data.

and/or

2. Apply previewing in VoCentering plugin (this will not reduce the size of the data).

Note: If you have applied previewing in the loader and again in the centering plugin you will be applying previewing to the previewed (reduced size) data.

See Full pipeline with auto-centering

Manual-centering

Ensure the VoCentering algorithm is not in the process list (remove it or turn it off if it is already inside your list). Modify the centre_of_rotation value in the reconstruction plugin, see Apply manual centering. If the manual centering value is approximate you can apply parameter tuning, see Apply parameter tuning to the centre of rotation

Parameter tuning

If you wish to test a preview reconstruction with a range of values for a parameter, for instance, if the centering is not quite optimal, then you can add different values separated by semi-colons. Each ‘tuned’ parameter will add an extra dimension to the data.

Parameter tuning examples

```
>>> mod 6.7 85;86;87  # three distinct values
>>> mod 6.7 84:86:0.5; # a range of values (start:stop:step) with semi-colon at the end
>>> mod 6.6 FBP;CGLS  # values can be strings
```

See Example 2 and Apply parameter tuning to the centre of rotation.

View the Savu output

In DAWN

Open a new terminal window and type:
Choose the Data Browsing perspective and click on File -> open, navigate to an output folder and click on the .nxs file.

**Warning:** The DAWN module must be loaded in a separate terminal as it will reset relevant paths.

**In Avizo**

Start avizo

```bash
>>> module load avizo
>>> avizo
```

In Avizo GUI, Click on Open Data /(File->Open Data). This should show a dialog box with list of output data entries. To view final output select entry/final_result_tomo/data and press OK button. This will load the data.

1. **2D view**

To view 2D slices, Select the data, right click and a pop up will be shown as below. Select Ortho Slice and Click ok button to show 2D slice.

2. **3D view**

To view 3D volume, Select the data, right click on it and a pop up will be shown as below. Select Volume Rendering and Click OK button to show 3D volume.
2.4 Savu Developer Guide

2.4.1 Savu Developer Information

Release Strategy

When a new release is required the following process should be done. Make a new branch for convinience if fixes are needed, it should be called vX.Y-release i.e v0.1-release. After the branch is made, move to it and then create a release tag for it called vX.Y.Z i.e v0.1.0, where Z starts at 0, and future minor release verisons can be made with v0.1.1 etc later on down the branch.

Once this is done, Zenodo.org will automatically create and archive an artefact.

We should then update the pypi stuff.

Developing on a feature branch

Make a new Branch for development and move to it  git checkout -b new_branch

Make modifications and commit as normal  git commit -m “test commit”

push this branch up to github  git push –set-upstream origin new_branch

get all the latest data from github on all branches  git fetch

to keep up to date, merge recent changes from master into the development branch and fix issues if there are any  
git merge origin/master

Continue working with the branch untill you are happy with the new feature, merge master into it as shown before and fix up issues, then merge the branch into master

move to master  git checkout master

update  git pull

merge in the new branch  git merge new_branch
there should be no problems if you have merge master in first, so just push back up git push

Testing a new plugin using DAWN

DAWN can be downloaded from http://www.dawnsci.org/ and general user tutorials are found at https://www.youtube.com/user/DAWNScience

Using the Debug perspective, create a new test, e.g. “plugin_test_recon.py” to test your plugin in “/Savu/savu/test/”, in this case the “example_filter_back_projection.py” plugin for reconstructing data, setting the self.plugin_name appropriately. After saving the file, right-click on it in the PyDev Package Explorer window and Run As a Python unit-test

Include file u’/home/docs/checkouts/readthedocs.org/user_builds/savu/checkouts/latest/savu/test/plugin_test_recon.py’ not found or reading it failed

This runs a series of tests and produces an output file with the result of the plugin, whether it be a filter or a reconstruction, allowing for visualisation of the data, providing a check of whether the process has worked successfully.

The output file is saved in a tmp directory as a .h5 file, e.g. “/tmp/tmp32bexK.h5”. This can be viewed in DAWN.

Adding C/C++ extensions to a plugin

There are numerous ways to create python bindings to external C/C++ libraries, which may be useful to recycle existing code or to improve performance. Two different approaches have currently been tested: Cython (to link to external C code) and Boost.Python (to link to external C++ code). Cython is essentially python with C-types and requires a C-API, a python wrapper and a makefile, whilst Boost.Python is a wrapper for the Python/C API and requires a wrapper and a makefile. By building the makefile a shared library (*.so) file is created and can be added to the \lib directory in the Savu framework and imported as a python module.

Cython Example

http://docs.cython.org/src/tutorial/clibraries.html

1) A C interface: A *.pxd file, which is similar to a C header file, providing C function definitions required in the python code. For example, cdezing.pxd:

```python
1  cdef extern from "./options.h":
2      ctypedef struct Options:
3          unsigned char versionflag
4          unsigned char f_call_num
5          size_t cropwd
6          unsigned int nlines
7          float outlier_mu
8          unsigned char returnflag
9          unsigned int npad
10
11  cdef extern from "./timestamp.h":
12      void timestamp_open(const char * const logname)
13      void timestamp_close()
14      void timestamp_init()
15      void timestamp(const char * const stampmsg)
16
17  cdef extern from "./dezing_functions.h":
18      void runDezing(Options * ctrlp, unsigned int thisbatch,unsigned char * inbuf,
19          unsigned char * outbuf )
```
2) A python wrapper: A *.pyx file that must have a different name to the *.pyd file above. For example, dezing.pyx:

```python
import numpy as np
cimport numpy as np
cimport cdezing

cdef cdezing.Options ctrl
cdef unsigned int batchsize

def getversion():
    global ctrl
    ctrl.versionflag=1
cdezing.runDezing(&ctrl,0,NULL,NULL)

def setup_size(array_size, outlier_mu, npad, logfile="dezing.log", versionflag=0): #,
    /* ... some other python code here ... */
cdezing.timestamp( "calling c function setup" )
cdezing.runDezing(&ctrl,batchsize, NULL, NULL)
pass

def setup(np.ndarray[np.uint16_t,ndim=3,mode="c"] inarray,np.ndarray[np.uint16_t,
    ndim=3,mode="c"] outarray, outlier_mu, npad, logfile="dezing.log", versionflag=0): #,
    /* ... some other python code here ... */
cdezing.timestamp( "calling c function setup" )
cdezing.runDezing(&ctrl,batchsize, inbuf, outbuf)
pass

def run(np.ndarray[np.uint16_t,ndim=3,mode="c"] inarray,np.ndarray[np.uint16_t,ndim=3,
    mode="c"] outarray): #,bytes summary):
    /* ... some other python code here ... */
cdezing.runDezing(&ctrl,batchsize, <unsigned char *>np.PyArray_DATA(inarray),
    <unsigned char *> np.PyArray_DATA(outarray))
pass

def cleanup(): #,bytes summary):
    /* ... some other python code here ... */
cdezing.runDezing(&ctrl,batchsize, NULL,NULL)
cdezing.timestamp_close()
```

3) Makefile: In python this is a setup.py file. For example, setup.py:

(continues on next page)
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext
setup(
    cmdclass={'build_ext':build_ext},
    ext_modules=[
        Extension("dezing", ["dezing.pyx"],
            libraries=['dezing'])
    ]
)

Compile this file, passing appropriate C compiler flags if necessary, to obtain a *.so file.

Literal block expected; none found.

e.g. export CFLAGS="-I . $CFLAGS" export LDFLAGS="-L . $LDFLAGS" python setup.py build_ext -i

The output file for this example is a dezing.so file. Transfer this file to lib and import as a python module, e.g. import dezing

**Boost.Python Example**

http://www.boost.org/doc/libs/1_58_0/libs/python/doc/

Boost.python aims to expose C++ classes/functions to python, without changing the original code.

1) A python wrapper: Create the python module and define the external function names. For example, example_wrapper.cpp

```cpp
#define NPY_NO_DEPRECATED_API NPY_1_7_API_VERSION
#include "include/numpy_boost_python.hpp"

// ...other relevant header files...

using namespace boost::python;

#include "example.hpp"
#include "example.cpp"

BOOST_PYTHON_MODULE(example)
{
    import_array();
    numpy_boost_python_register_type<float, 1>();
    numpy_boost_python_register_type<float, 3>();
    def("cpp_function1", example_function1);
    def("cpp_function2", example_function2);
    def("cpp_function3", example_function3);
}
```


For example, example_makefile

```bash
cxx = g++
pythoninc = -I/usr/include/python2.6 -I/usr/lib64/python2.6/site-packages/numpy/core/
    include
```

(continues on next page)
Savu Documentation, Release 2.4

(continued from previous page)

```plaintext
PYTHONLIB = -lboost_python

CXXFLAGS = -q -O2 -fPIC -ansi -Wall -Wno-long-long -DUSE_TIMER=true -fopenmp -I../src
\rightarrow $(PYTHONINC)

LD = $(CXX)
LDFLAGS = $(CXXFLAGS) -shared
LIBS = $(PYTHONLIB)

PYOBJS = example_wrapper.o
OBJECTS = # ...lots of .o files... #

example.so: $(PYOBJS) $(OBJECTS)
\$(LD) $(LDFLAGS) -o $@ $(PYOBJS) $(OBJECTS) $(LIBS)

# ...more code here... #
```

The output file for this example is a example.so file. Transfer this file to lib and import as a python module, e.g. import example, then simply access a function from within your python code as example.example_function1(...).

The class and header files example.cpp and example.hpp (below) along with example_wrapper.cpp, illustrate how to incorporate numpy arrays into the extension.

Include file u'/home/docs/checkouts/readthedocs.org/user_builds/savu/checkouts/latest/extension_examples/example.hpp' not found or reading it failed

```plaintext
void example_function1(numpy_boost<float, 3> &pixels, const real param_n, const real param_r, const int num_series)
{
    /* ... some other c++ code here ... */
}

numpy_boost<float, 3> example_function2(const numpy_boost<float, 3> &pixels, const numpy_boost<float, 1> &angles, double rotation_centre, int resolution, int niterations, int nthreads)
{
    /* ... some other c++ code here ... */
}

void example_function3()
{
    /* ... some other c++ code here ... */
}
```

### 2.4.2 Plugin Developer Guide

**Contributing to Savu on Github**

**Github Savu**

To become a Savu developer you will need your own Savu Github repository and a clone on your local machine. The basic commands to do this are listed below; for a more detailed explanation see

https://help.github.com/articles/fork-a-repo/

Chapter 2. Table of Contents
Set up your Savu Github repository (DO THIS ONCE)

1. Navigate to https://github.com/DiamondLightSource/Savu and click on Fork (right hand corner).
2. Navigate to YOUR FORK of the repository and copy the url e.g. (ensure SSH is chosen).
3. Open a terminal (or command prompt) on your local machine, change a relevant folder to install Savu, and follow the instructions below:

```bash
>>> git clone <ssh url>
>>> ssh-keygen -t rsa -b 4096 -C "your Github email@example.com".
>>> git remote add upstream git@github.com:DiamondLightSource/Savu.git
```

Note: You may need to generate an ssh key. See https://help.github.com/articles/generating-a-new-ssh-key/

Maintain YOUR Savu repository

You can now go ahead and make changes to your local copy of Savu. These changes need to be frequently committed to your local repository and periodically pushed to your remote repository on GitHub.

To see a list of files that you have recently updated and NOT yet committed:

```bash
>>> git status
```

To commit these changes locally:

```bash
>>> git commit -a -m "your commit message here"
```

To push these changes to your remote GitHub repository:

```bash
>>> git push
```

Keep YOUR repository up-to-date with the ORIGINAL

```bash
>>> git pull upstream
```

Note: You may need to specify the branch you want to update from at the end of the command, such as ‘master’

Then follow the steps in Maintain YOUR Savu repository.
Tracking and untracking files

```bash
>>> git status
```
gives a list of tracked and untracked files (if there are any). Untracked files are only held locally and changes to these files will not be tracked by the repository. If you create a new file, it will not automatically be tracked by the repository.

To track a file

```bash
>>> git status
>>> git add <untracked file_path>
```
Then commit and push as per *Maintain YOUR Savu repository*.

To untrack a file

```bash
>>> git rm <option> <file_path>
```
option is `--cached` if you want to untrack the file but keep a local copy.
option is `-f` if you want to completely delete the file.

Add your changes to the ORIGINAL repository

Navigate to your repository and click on the green `New pull request` button.

Developing a Savu plugin

Plugin Developer API

API Documentation

Information on specific functions, classes, and methods.
tocctree glob pattern u’../api_plugin_dev/*’ didn’t match any documents

Create a test for your plugin

Testing something else in line with text *example*

In order to submit a new plugin to Savu on Github, you **MUST** provide a test for your new plugin. To create a test follow the steps below:

1. Choose a *test template*
2. Choose a *test dataset*
3. *Amend the parameters* r1,…,r8 in the file.
4. Save the file.
5. Add the file to your local repository.

Test templates

If your plugin is not dependent on any others (i.e. it can be run on its own on raw or corrected data), then download the sample test WITHOUT a process list. This will test the plugin with default parameters.

If your plugin is dependent on other plugins, you will need to create a process list and download the sample test WITH a process list.

Test data

List of test data available. What to do if you require different test data. You can submit a new test dataset to Savu, with the requirement that it is less than 10MB in size.

Amending the parameters

See the real test modules:

1. median_filter_test.py tests the median_filter_plugin.py plugin WITH NO PROCESS LIST.
2. median_filter_test.py tests the median_filter_plugin.py plugin WITH NO PROCESS LIST.

Save the file as “your_module_name.py”

Warning: Ensure the test file name has the same name as the module name (r1)

Note: Have a look at the real test for the median_filter_plugin.py module.

List of test data available. What to do if you require different test data.

Internal crossreferences, like example.

This is an example crossreference target.

2.5 API Documentation

2.5.1 Framework API

Information on specific functions, classes, and methods.

savu

savu.tomo_recon module

savu.version module

Get the Savu version
savu.core

savu.core.plugin_runner module

savu.core.transport_setup module

class MPI_setup (options, name='MPI_setup')
    Bases: object

    _MPI_setup__add_console_logging ()
    _MPI_setup__add_user_logging (options)
    _MPI_setup__mpi_setup (options)
        Set MPI process specific values and logging initialisation.
    _MPI_setup__set_dictionary (options)
        Fill the options dictionary with MPI related values.
    _MPI_setup__set_logger (level, fmt, fname=None)
    _MPI_setup__set_logger_parallel (number, rank, options)
        Set parallel logger.
    _MPI_setup__set_logger_single (options)
        Set single-threaded logging.
    call_mpi__barrier ()
        Call MPI_barrier before an experiment is created.

savu.core.utils module

    _get_log_level (options)
        Gets the right log level for the flags -v or -q
    _output_summary (mpi_flag, plugin)
    _savu_decoder (data)
    _savu_encoder (data)
    _send_email (address)

    add_base (this, base)
        Add a base class to a class.
        
        Params class this a class instance
        Params class base a class to add as a base class

    add_base_classes (this, bases)
        Add multiple base classes to a class.
        
        Params class this a class instance.
        Params list(class) bases a list of base classes

    add_syslog_log_handler (logger, syslog_address, syslog_port)
    add_user_log_handler (logger, user_log_path)
    add_user_log_level ()
docstring_parameter(*sub)
    Decorator to add strings to a doc string.

get_available_gpus()

get_memory_usage_linux(*kb=False, *mb=True)
    Parameters
    • *kb – Return the value in Kilobytes
    • *mb – Return the value in Megabytes
    Returns The string of the value in either KB or MB

Field list ends without a blank line; unexpected unindent.

:rtype str

import_class(class_name)
    Import a class.

    Params class name
    Returns class instance
    Return type instance of class_name

logfunction(func)
    Decorator to add logging information around calls for use with .

logmethod(func)
    Decorator to add logging information around calls for use with .

user_message(message)

user_messages_from_all(header, message_list)

savu.core.basic_plugin_runner module

savu.core.checkpointing module

class Checkpointing(exp, name='Checkpointing')
    Bases: object
    Contains all checkpointing associated methods.

    _Checkpointing__does_file_exist(thefile, level)
    _Checkpointing__load_data()
    _Checkpointing__set_checkpoint_info()
    _Checkpointing__set_dataset_metadata(f, dtype)
    _Checkpointing__set_start_values(v1, v2, v3)
    _Checkpointing__write_plugin_checkpoint()
    _Checkpointing__write_subplugin_checkpoint(ti, pi)
    _create_dataset(f, name, val)
    _get_checkpoint_params()
    _get_dataset_metadata(dtype, name)
_get_timer()

_initialise(comm)
    Create a new checkpoint file

_reset_indices()

_set_checkpoint_info_from_file(level)

_set_timer()

get_checkpoint_plugin()

get_level()

get_proc_idx()

get_start_values()

get_trans_idx()

is_time_to_checkpoint(transport, ti, pi)

output_plugin_checkpoint()

set_completed_plugins(n)

savu.core.transports

savu.core.transports.base_transport module

class BaseTransport
    Bases: object

    Implements functions that control the interaction between the data and plugin layers.

    _BaseTransport__create_dataset(entry, name, data)
        Create a function that re-adds missing dimensions of length 1.

        Parameters
data (Data) -- Dataset

        Returns expansion function

        Return type lambda

    _BaseTransport__create_expand_function(data)
        Create a function that removes dimensions of length 1.

        Parameters
data (Data) -- Dataset

        Returns squeeze function

        Return type lambda

    _BaseTransport__get_checkpoint_params(plugin)

    _BaseTransport__output_axis_labels(data, entry)

    _BaseTransport__output_data(entry, data, name)

    _BaseTransport__output_data_patterns(data, entry)

    _BaseTransport__output_data_type(entry, data, name)
_get_all_slice_lists (data_list, dtype)
Get all slice lists for the current process.

Parameters
- data_list (list(Data)) – Datasets

Returns
A list of dictionaries containing slice lists for each dataset

Return type
list(dict)

_get_filenames (plugin_dict)

_get_input_data (plugin, trans_data, nproc, ntrans)

_get_link_type (name)

_get_output_data (result, count)

_initialise (plugin)

_log_completion_status (count, nTrans, name)

_output_metadata (data, entry, name, dump=False)

_output_metadata_dict (entry, mData)

_populate_nexus_file (data)

_process_loop (plugin, prange, tdata, count, pDict, result, cp)

_remove_excess_data (data, result, slice_list)
Remove any excess results due to padding for fixed length process frames.

_return_all_data (count, result, end)
Transfer plugin results for current frame to backing files.

Parameters
- count (int) – The current frame index.
- result (list(np.ndarray)) – plugin results
- end (bool) – True if this is the last entry in the slice list.

_set_file_details (files)

_set_functions (data_list, name)
Create a dictionary of functions to remove (squeeze) or re-add (expand) dimensions, of length 1, from each dataset in a list.

Parameters
- data_list (list(Data)) – Datasets
- name (str) – "squeeze" or "expand"

Returns
A dictionary of lambda functions

Return type
dict

_set_global_frame_index (plugin, frame_list, nProc)
Convert the transfer global frame index to a process global frame index.

_setup_h5_files ()

_transfer_all_data (count)
Transfer data from file and pad if required.

Parameters
- count (int) – The current frame index.
Returns All data for this frame and associated padded slice lists

Return type list(np.ndarray), list(tuple(slice))

_transport_checkpoint()
The framework has determined it is time to checkpoint. What should the transport mechanism do? Over-
ride if appropriate.

_transport_initialise(options)
Any initial setup required by the transport mechanism on start up. This is called before the experiment is
initialised.

_transport_kill_signal()
An opportunity to send a kill signal to the framework. Return True or False.

_transport_load_plugin(exp, plugin_dict)
This method is called before each plugin is loaded

_transport_post_plugin()
This method is called directly AFTER each plugin is executed.

_transport_post_plugin_list_run()
This method is called AFTER the full plugin list has been processed.

_transport_pre_plugin()
This method is called directly BEFORE each plugin is executed, but after the plugin is loaded.

_transport_pre_plugin_list_run()
This method is called after all datasets have been created but BEFORE the plugin list is processed.

_transport_process(plugin)
Organise required data and execute the main plugin processing.

Parameters plugin(plugin) – The current plugin instance.

_transport_terminate_dataset(data)
A dataset that will subequently be removed by the framework.

Parameters data(Data) – A data object to finalise.

_transport_update_plugin_list()
This method provides an opportunity to add or remove items from the plugin list before plugin list check.

process_setup(plugin)

savu.core.transports.basic_transport module

class BasicTransport
Bases: savu.core.transports.base_transport.BaseTransport

_BasicTransport__set_hdf5_transport()

_BasicTransport__unset_hdf5_transport()

_transport_initialise(options)
Any initial setup required by the transport mechanism on start up. This is called before the experiment is
initialised.

_transport_post_plugin()
This method is called directly AFTER each plugin is executed.

_transport_pre_plugin()
This method is called directly BEFORE each plugin is executed, but after the plugin is loaded.
_transport_pre_plugin_list_run()
This method is called after all datasets have been created but BEFORE the plugin list is processed.

_transport_terminate_dataset(data)
A dataset that will subsequently be removed by the framework.

Parameters data (Data) – A data object to finalise.

savu.core.transports.hdf5_transport module

class Hdf5Transport
Bases: savu.core.transports.base_transport.BaseTransport

_metadata_dump(f, gname)

_transport_checkpoint()
The framework has determined it is time to checkpoint. What should this transport mechanism do?

_transport_cleanup(i)
Any remaining cleanup after kill signal sent

_transport_initialise(options)
Any initial setup required by the transport mechanism on start up. This is called before the experiment is initialised.

_transport_kill_signal()
An opportunity to send a kill signal to the framework. Return True or False.

_transport_post_plugin()
This method is called directly AFTER each plugin is executed.

_transport_pre_plugin()
This method is called directly BEFORE each plugin is executed, but after the plugin is loaded.

_transport_pre_plugin_list_run()
This method is called after all datasets have been created but BEFORE the plugin list is processed.

_transport_terminate_dataset(data)
A dataset that will subsequently be removed by the framework.

Parameters data (Data) – A data object to finalise.

_transport_update_plugin_list()
This method provides an opportunity to add or remove items from the plugin list before plugin list check.

savu.core.transports.dosna_transport module

savu.data

savu.data.chunking module

class Chunking(exp, patternDict)
Bases: object

A class to save tomography data to a hdf5 file

_chunking__adjust_chunk_size(chunks, ttype, shape, adjust)
Adjust the chunk size to as close to 1MB as possible

_chunking__check_adjust_dims(adjust, chunks, up_down)
_Chunking__core_core (dim, adj_idx, adjust, shape)
_Chunking__core_other (dim, adj_idx, adjust, shape)
_Chunking__core_slice (dim, adj_idx, adjust, shape)
_Chunking__decrease_chunks (chunks, ttype, adjust)
  Decrease the chunk size to below but as close to 1MB as possible
_Chunking__get_adjustable_dims ()
  Get all core dimensions and fastest changing slice dimension (all potentially adjustable)
_Chunking__get_idx_decrease (chunks, adjust)
  Determine the chunk dimension to decrease
_Chunking__get_idx_increase (chunks, adjust)
  Determine the chunk dimension to increase
_Chunking__get_idx_order (adjust, chunks, direction)
_Chunking__get_max_frames_dict ()
_Chunking__get_shape (shape, ddict)
  Get shape taking into account padding.
_Chunking__increase_chunks (chunks, ttype, shape, adjust)
  Increase the chunk size as close to 1MB as possible
_Chunking__lustre_workaround (chunks, shape)
_Chunking__max_frames_per_process (shape, nFrames, allslices=None)
  Calculate the max possible frames per process
_Chunking__set_adjust_params (shape)
  Set adjustable dimension parameters (the dimension number, increment and max value)
_Chunking__set_chunks (chunks, shape, adjust)
  Calculate initial chunk values
_Chunking__set_volume_bounds (adjust, dim, chunks)
_Chunking__slice_other (dim, adj_idx, adjust, shape)
_Chunking__slice_slice (dim, adj_idx, adjust, shape)
_calculate_chunking (shape, ttype, chunk_max=None)
  Calculate appropriate chunk sizes for this dataset

savu.data.experiment_collection module

savu.data.plugin_list module

savu.data.framework_citations module

get_framework_citations ()
  return a list of NXcite objects
savu.data.meta_data module

class MetaData (options={}, ordered=False)
    Bases: object

    The MetaData class creates a dictionary of all meta data which can be accessed using the get and set methods. It also holds an instance of PluginList.

    _set_dictionary (ddict)
        Set the meta data dictionary

    delete (entry)
        Delete an entry from the meta data dictionary.

            Parameters entry (str) – The dictionary key entry to delete.

    get (maplist, setFlag=False, value=True, units=False)
        Get a value from the meta data dictionary, given its key(s).

            Params maplist Dictionary key(s).

            Returns Value from the dictionary corresponding to the given key(s)

            Return type value

        Dictionaries within dictionaries are accessed by placing successive keys in a list.

    get_dictionary ()
        Get the meta_data dictionary.

            Returns A dictionary.

            Return type dict

    set (name, value)
        Create and set an entry in the meta data dictionary.

            Parameters

                • name (str or list(str)) – dictionary key(s). If name is a list then each successive name will become an entry in the dictionary which has the previous name as its key.

                • value (value) – dictionary value

        For example,

            >>> MetaDataObj.set([\'name1\', \'name2\'], 3)
            >>> MetaDataObj.get_dictionary()
            {'name1': {'name2': 3}}

savu.data.data_structures

savu.data.data_structures.data module

class Data (name, exp)
    Bases: savu.data.data_structures.data_create.DataCreate

    The Data class dynamically inherits from transport specific data class and holds the data array, along with associated information.

    _Data__check_dims ()
        Check the shape and nDims entries in the data_info meta_data dictionary are equal.
_Data__check_pattern(pattern_name)
Check if a pattern exists.

_Data__convert_pattern_dimensions(dtype)
Replace negative indices in pattern kwargs.

_Data__get_available_pattern_list()
Get a list of ALL pattern names that are currently allowed in the framework.

_Data__get_dirs_for_volume(dim1, dim2, sdir, dim3=None)
Calculate core_dir and slice_dir for a volume pattern.

_Data__get_dirs_for_volume_3D()

_Data__initialise_data_info(name)
Initialise entries in the data_info meta data.

_Data__set_main_axis(pname)
Set the main_dir pattern kwarg to the fastest changing dimension

_clear_plugin_data()
Set encapsulated PluginData object to None.

_finalise_patterns()
Adds a main axis (fastest changing) to SINOGRAM and PROJECTON patterns.

_get_plugin_data()
Get encapsulated PluginData object.

_get_transport_data()

_non_negative_directions(ddirs, nDims)
Replace negative indexing values with positive counterparts.

    Params tuple(int) ddirs  data dimension indices
    Params int nDims  The number of data dimensions
    Returns  non-negative data dimension indices
    Return type  tuple(int)

_set_name(name)

_set_plugin_data(plugin_data_obj)
Encapsulate a PluginData object.

_set_previous_pattern(pattern)

_set_transport_data(transport)
Import the data transport mechanism

    Returns  instance of data transport
    Return type  transport_data

add_pattern(dtype, **kwargs)
Add a pattern.

    Params str dtype  The type of pattern to add, which can be anything from the savu.data.
    data_structures.utils.pattern_list pattern_list  savu.data.
    data_structures.utils.pattern_list pattern_list:

    Keyword Arguments
    • core_dims(tuple) – Dimension indices of core dimensions
• **slice_dims** *(tuple)* – Dimension indices of slice dimensions

**add_volume_patterns** *(x, y, z)*
Adds volume patterns

- **Params int x** dimension to be associated with x-axis
- **Params int y** dimension to be associated with y-axis
- **Params int z** dimension to be associated with z-axis

**amend_axis_label_values** *(slice_list)*
Amend all axis label values based on the slice_list parameter. This is required if the data is reduced.

**get_axis_label_keys** *
Get axis_label names

- **Returns** A list containing associated axis names for each dimension
- **Return type** list(str)

**get_axis_labels** *
Get axis labels.

- **Returns** Axis labels
- **Return type** list(dict)

**get_core_dimensions** *
Get the core data dimensions associated with the current pattern.

- **Returns** value associated with pattern key *core_dims*
- **Return type** tuple

**get_data_dimension_by_axis_label** *(name, contains=False)*
Get the dimension of the data associated with a particular axis_label.

- **Parameters name** *(str)* – The name of the axis_label
- **Keyword Arguments** contains *(bool)* – Set this flag to true if the name is only part of the axis_label name

- **Returns** The associated axis number
- **Return type** int

**get_data_patterns** *
Get data patterns associated with this data object.

- **Returns** A dictionary of associated patterns.
- **Return type** dict

**get_itemsize** *
Returns bytes per entry

**get_name** *(orig=False)*
Get data name.

- **Keyword Arguments orig** *(bool)* – Set this flag to true to return the original cloned dataset name if this dataset is a clone

- **Returns** the name associated with the dataset
- **Return type** str
get_preview()
    Get the Preview instance associated with the data object

get_previous_pattern()

get_shape()
    Get the dataset shape
    Returns data shape
    Return type tuple

get_slice_dimensions()
    Get the slice data dimensions associated with the current pattern.
    Returns value associated with pattern key slice_digs
    Return type tuple

set_axis_labels(*args)
    Set the axis labels associated with each data dimension.
    Parameters str – Each arg should be of the form name unit. If name is a
data_obj.meta_data entry, it will be output to the final .nxs file.

set_original_shape(shape)
    Set the original data shape before previewing

set_shape(shape)
    Set the dataset shape.

class DataMapping
    Bases: object
    A class providing helper functions to multi-modal loaders.
    check_is_map(proj_dir)
    check_is_tomo(proj_dir, rotation)

get_axes()

get_motor_type()

get_motors()

set_axes(axes)

set_motor_type(motor_type)

set_motors(motors)

class Padding(pData)
    Bases: object
    A class that organises padding of the data. An instance of Padding can be associated with a Data ob-
ject in a plugin that inherits from BaseFilter, inside savu.plugins.base_filter.BaseFilter.

set_filter_padding()

__Padding__set_dims()

__get_padding_directions()
    Get padding directions.
Returns: padding dictionary

Return type: dict

_get_plugin_padding_directions()
Get padding directions.

Returns: padding dictionary

Return type: dict

_pad_direction(pad_str)
Pad the data in a specified dimension.

param str pad_str A string of the form ‘dim.pad’, ‘dim.after.pad’ or ‘dim.before.pad’

• ‘dim’ should be replaced with the dimension to pad
• ‘pad’ should be replaced with the amount to pad
• ‘before’ and ‘after’ are optional positional keywords specifying padding ‘before’ or ‘after’ the data for the specified dimension index (if neither are specified BOTH will be padded)

pad_directions(pad_list)
Pad multiple, individually specified, dimensions.

param list(dict) pad_list A list of strings of the form ‘dim.pad’, ‘dim.after.pad’ or ‘dim.before.pad’

• ‘dim’ should be replaced with the dimension to pad
• ‘pad’ should be replaced with the amount to pad
• ‘before’ and ‘after’ are optional positional keywords specifying padding ‘before’ or ‘after’ the data for the specified dimension index (if neither are specified BOTH will be padded)

pad_frame_edges(padding)
Pad all the edges of a frame of data with the same pad amount (i.e pad in the core dimensions).

Parameters

padding (int) – The pad amount

pad_mode (mode)

pad_multi_frames(padding)
Add extra frames before and after the current frame of data (i.e pad in the fastest changing slice dimension).

Parameters

padding (int) – The pad amount

savu.data.data_structures.data_create module

class DataCreate(name='DataCreate')

Bases: object

Class that deals with creating a data object.

_DataCreate__add_extra_dims_labels()
Add axis labels to extra dimensions created by parameter tuning.

_DataCreate__amend_axis_labels(*args)
Helper function to remove, replace/add or insert axis_labels into existing axis_labels
**_DataCreate__copy_labels (copy_data)_**
Copy axis labels.

**_DataCreate__copy_patterns (copy_data)_**
Copy patterns

**_DataCreate__copy_patterns_removing_dimensions (pattern_list, all_patterns, nDims)_**
Copy patterns but remove specified dimensions from them.

**_DataCreate__create_axis_labels (axis_labels)_**
Create axis labels.

**_DataCreate__create_dataset_from_kwargs (kwargs)_**
Create dataset from kwargs.

**_DataCreate__create_dataset_from_object (data_obj)_**
Create a dataset from an existing Data object.

**_DataCreate__find_and_set_shape (data)_**
Add any extra dimensions, from parameter tuning, to the shape and set the shape in the framework.

**_DataCreate__insert_axis_labels (label)_**
Insert axis labels.

**_DataCreate__remove_axis_labels (label, removed_dims)_**
Remove axis labels.

**_DataCreate__replace_axis_labels (label)_**
Replace or add axis labels.

**_set_data_patterns (patterns)_**
Add missing dimensions to patterns and populate data info dict.

**create_dataset (*args, **kwargs)_**
Set up required information when an output dataset has been created by a plugin.

- **arg Data** A data object
- **keyword tuple shape** The shape of the dataset
- **keyword list axis_labels** The axis_labels associated with the datasets
- **keyword patterns** The patterns associated with the dataset (optional, see note below)
- **keyword type dtype** Type of the data (optional: Defaults to np.float32)
- **keyword bool remove** Remove from framework after completion (no link in .nxs file) (optional: Defaults to False.)
- **keyword bool raw** Keep dark and flats (ImageKey or NoImageKey)

**Note:**

**Creating a dataset** Each new dataset requires the following information:

- **shape**
- **axis_labels**
- **patterns**

This function can be used to setup the required information in one of two ways:

1. Passing a **Data** object as the only argument: All required information is copied from this data object. For example,
2. Passing kwargs: shape and axis_labels are required (see above for other optional arguments). For example,

```python
>>> out_dataset[0].create_dataset(axis_labels=labels, shape=new_shape)
```

**Warning:** If pattern keyword is not used, patterns must be added after `create_dataset()` by calling `add_pattern()`.

---

**Note:**

**shape keyword argument** Options to pass are:

1. Data object: Copy shape from the Data object.

```python
>>> out_dataset[0].create_dataset(axis_labels=labels, shape=in_dataset[0])
```

2. tuple: Define shape explicity.

```python
>>> out_dataset[0].create_dataset(axis_labels=labels, shape=(10, 20, 30))
```

---

**Note:**

**axis_labels keyword argument** Options to pass are:

1. Data object: Copy all labels from the Data object.

```python
>>> out_dataset[0].create_dataset(axis_labels=in_dataset[0], shape=new_shape)
```

2. `{Data_obj: list}`: Copy labels from the Data object and then remove or insert.
   - To remove dimensions: list_entry = ‘dim’. For example, to remove the first and last axis_labels from the copied list:

```python
>>> out_dataset[0].create_dataset(axis_labels={in_dataset[0]: ['1', '-1']}, shape=new_shape)
```

   - To add/replace dimensions: list_entry = ‘dim.name.unit’.  

```python
>>> out_dataset[0].create_dataset(axis_labels={in_dataset[0]: ['2.det_x.pixel', '3.det_y.pixel']}, shape=new_shape)
```

   - To insert dimensions: list_entry = ‘~dim.name.unit’.

```python
>>> out_dataset[0].create_dataset(axis_labels={in_dataset[0]: ['~2.det_x.pixel', '~3.det_y.pixel']}, shape=new_shape)
```
(or a combination, where each successive step is applied after the previous changes have been made.)

3. list: Where each element is of the form ‘name.unit’.

```python
>>> out_dataset[0].create_dataset(axis_labels=['rotation.deg', 'det_x.pixel', 'det_y.pixel'], shape=new_shape)
```

Note:

**patterns keyword argument** Options to pass are:

1. Data object: Copy all patterns from the Data object.

   ```python
   >>> out_dataset[0].create_dataset(axis_labels=labels, shape=new_shape, patterns=in_dataset[0])
   ```

2. {Data_obj: list}: Copy only the patterns given in the list from the Data object.

   • Copy the patterns: list_entry = ‘name’

     ```python
     >>> out_dataset[0].create_dataset(axis_labels=labels, shape=new_shape, patterns={in_dataset[0]: ['SINOGRAM', 'PROJECTION']})
     ```

   • Copy patterns but remove dimensions: list_entry = ‘name1.r1,r2…’:

     ```python
     >>> out_dataset[0].create_dataset(axis_labels=labels, shape=new_shape, patterns={in_dataset[0]: ['SINOGRAM.1', 'PROJECTION.1']})
     ```

   • Copy ALL patterns but remove dimensions: list_entry = ‘*..r1,r2…’:

     ```python
     >>> out_dataset[0].create_dataset(axis_labels=labels, shape=new_shape, patterns={in_dataset[0]: '*.*'})
     ```

`get_dtype()`

`set_dtype(dtype)`

`savu.data.data_structures.data_notes module`

`_create()`

**Note:**

**Creating a dataset** Each new dataset requires the following information:

• shape

• axis_labels

• patterns
This function can be used to setup the required information in one of two ways:

1. Passing a `Data` object as the only argument: All required information is copied from this data object. For example,

```python
>>> out_dataset[0].create_dataset(in_dataset[0])
```

2. Passing kwargs: `shape` and `axis_labels` are required (see above for other optional arguments). For example,

```python
>>> out_dataset[0].create_dataset(axis_labels=labels, shape=new_shape)
```

**Warning:** If `pattern` keyword is not used, patterns must be added after `create_dataset()` by calling `add_pattern()`.

---

`_padding()`

- ‘dim’ should be replaced with the dimension to pad
- ‘pad’ should be replaced with the amount to pad
- ‘before’ and ‘after’ are optional positional keywords specifying padding ‘before’ or ‘after’ the data for the specified dimension index (if neither are specified BOTH will be padded)

`_set_preview_note()`

Each `preview_list` element should be of the form `start:stop:step:chunk`, where `stop, step` and `chunk` are optional (defaults: `stop = start + 1, step = 1, chunk = 1`) but must be given in that order.

**Note:**

`start:stop[:step]` represents the set of indices specified by:

```python
>>> indices = range(start, stop[, step])
```

For more information see `range()`

`start:stop:step:chunk (chunk > 1)` represents the set of indices specified by:

```python
>>> a = np.tile(np.arange(start, stop, step), (chunk, 1))
>>> b = np.transpose(np.tile(np.arange(chunk)-chunk/2, (a.shape[1], 1)))
>>> indices = np.ravel(np.transpose(a + b))
```

Chunk indicates how many values to take around each value in `range(start, stop, step)`. It is only available for slicing dimensions.

**Warning:** If any indices are out of range (or negative) then the list is invalid. When `chunk > 1`, new start and end values will be:

```python
>>> new_start = start - int(chunk/2)
>>> new_end = range(start, stop, step)[-1] + (step - int(chunk/2))
```

**accepted values:** Each entry is executed using `eval()` so simple formulas are allowed and may contain the following keywords:
• : is a simplification for 0:end:1:1 (all values)
• mid is int(shape[dim]/2)-1
• end is shape[dim]

_shape()_

Note:

**shape keyword argument** Options to pass are:

1. Data object: Copy shape from the Data object.

```python
>>> out_dataset[0].create_dataset(axis_labels=labels, shape=in_dataset[0])
```

2. tuple: Define shape explicity.

```python
>>> out_dataset[0].create_dataset(axis_labels=labels, shape=(10, 20, 30))
```

axis_labels()

Note:

**axis_labels keyword argument** Options to pass are:

1. Data object: Copy all labels from the Data object.

```python
>>> out_dataset[0].create_dataset(axis_labels=in_dataset[0], shape=new_shape)
```

2. {Data_obj: list}: Copy labels from the Data object and then remove or insert.
   - To remove dimensions: list_entry = ‘dim’. For example, to remove the first and last axis_labels from the copied list:

   ```python
   >>> out_dataset[0].create_dataset(axis_labels={in_dataset[0]: ['1', '-1']}, shape=new_shape)
   ```

   - To add/replace dimensions: list_entry = ‘dim.name.unit’.

   ```python
   >>> out_dataset[0].create_dataset(axis_labels={in_dataset[0]: ['2.det_x.pixel', '3.det_y.pixel']}, shape=new_shape)
   ```

   - To insert dimensions: list_entry = ‘~dim.name.unit’.

   ```python
   >>> out_dataset[0].create_dataset(axis_labels={in_dataset[0]: ['~2.det_x.pixel', '~3.det_y.pixel']}, shape=new_shape)
   ```

   (or a combination, where each successive step is applied after the previous changes have been made.)

3. list: Where each element is of the form ‘name.unit’.
```python
>>> out_dataset[0].create_dataset(axis_labels=['rotation.deg', 'det_x.pixel', 'det_y.pixel'], shape=new_shape)
```

---

**image_key()**

This is a helper function to be used after `savu.data.data_structures.data_create.DataCreate.create_dataset()`.

```python
>>> out_dataset[0].create_dataset(in_dataset[0])
>>> out_dataset[0].trim_output_data(in_dataset[0], image_key=0)
```

if `in_dataset[0]` is a plugin input dataset with an image_key and 0 is the data index.

**patterns()**

---

**Note:**

**patterns keyword argument** Options to pass are:

1. Data object: Copy all patterns from the Data object.
   ```python
   >>> out_dataset[0].create_dataset(axis_labels=labels, shape=new_shape,
   ... patterns=in_dataset[0])
   ```

2. {Data_obj: list}: Copy only the patterns given in the list from the Data object.
   - Copy the patterns: list_entry = 'name'
     ```python
     >>> out_dataset[0].create_dataset(axis_labels=labels, shape=new_shape,
     ... patterns={in_dataset[0]: ['SINOGRAM', 'PROJECTION']})
     ```
   - Copy patterns but remove dimensions: list_entry = 'name1.r1,r2...':
     ```python
     >>> out_dataset[0].create_dataset(axis_labels=labels, shape=new_shape,
     ... patterns={in_dataset[0]: ['SINOGRAM.1', 'PROJECTION.1']})
     ```
   - Copy ALL patterns but remove dimensions: list_entry = '*r1,r2...':
     ```python
     >>> out_dataset[0].create_dataset(axis_labels=labels, shape=new_shape,
     ... patterns={in_dataset[0]: '*.0'})
     ```

---

**savu.data.data_structures.plugin_data module**

**class PluginData(data_obj, plugin=None)**

**Bases:** object

The PluginData class contains plugin specific information about a Data object for the duration of a plugin. An instance of the class is encapsulated inside the Data object during the plugin run

```python
_pluginData__check_dimensions(indices, core_dir, slice_dir, nDims)
_pluginData__check_distribution(mft)
```
_PluginData__get_slice_size (mft)
  Calculate the number of frames transfer in each dimension given mft.

_PluginData__log_max_frames (mft, mfp, check=True)

_PluginData__perform_checks (nFrames)

_PluginData__set_core_shape (shape)
  Set the core shape to hold only the shape of the core dimensions

_PluginData__set_max_frames (mft, mft_shape, mfp)

_PluginData__set_pattern (name, first_sdim=None)
  Set the pattern related information in the meta data dict.

_PluginData__set_core_shape ()
  Set the shape of the plugin data processing chunk.

_PluginData__set_slice_dimensions (first_sdim=None)
  Set the slice dimensions in the pluginData meta data dictionary. Reorder pattern slice dims to ensure
  first_sdim is at the front.

_calculate_max_frames ()

_get_data_slice_list (plist)
  Convert a plugin data slice list to a slice list for the whole dataset, i.e. add in any missing dimensions.

_get_fixed_dimensions ()
  Get the fixed data directions and their indices

    Returns  Fixed directions and their associated values
    Return type  list(list(int), list(int))

_get_max_frames_process ()
  Get the number of frames to process for each run of process_frames.

    If the number of frames is not divisible by the previewing chunk value then amend the number of frames
to gcd(frames, chunk)

    Returns  Number of frames to process
    Return type  int

_get_max_frames_transfer ()
  Get the number of frames to transfer for each run of process_frames.

_get_no_squeeze ()

_get_plugin_data_size_params ()

_get_preview ()

_get_shape_before_tuning ()
  Return the shape of the full dataset used during each run of the plugin (i.e. ignore extra dimensions due to
  parameter tuning).

_set_meta_data ()

_set_no_squeeze ()

_set_padded_shape ()

_set_padding_dict ()
_set_shape_before_tuning (shape)
    Set the shape of the full dataset used during each run of the plugin (i.e. ignore extra dimensions due to
    parameter tuning).

_set_shape_transfer (slice_size)

get_core_dimensions ()
    Return the position of the core dimensions in relation to the data handed to the plugin.

get_core_shape ()
    Get the shape of the core dimensions only.
    Returns shape of core dimensions
    Return type tuple

get_current_frame_idx ()
    Returns the index of the frames currently being processed.

get_data_dimension_by_axis_label (label, contains=False)
    Return the dimension of the data in the plugin that has the specified axis label.

get_frame_limit ()

get_padded_shape ()
    Get the shape of the data (with padding) that is passed to the plugin process_frames method.

get_pattern ()
    Get the current pattern.
    Returns dict of the pattern name against the pattern.
    Return type dict

get_pattern_name ()
    Get the pattern name.
    Returns the pattern name
    Return type str

get_shape ()
    Get the shape of the data (without padding) that is passed to the plugin process_frames method.

get_shape_transfer ()
    Get the shape of the plugin data to be transferred each time.

get_slice_dimension ()
    Return the position of the slice dimension in relation to the data handed to the plugin.

get_total_frames ()
    Get the total number of frames to process (all MPI processes).
    Returns Number of frames
    Return type int

plugin_data_setup (pattern, nFrames, split=None, slice_axis=None)
    Setup the PluginData object.
    Parameters
        • pattern (str) – A pattern name
- **nFrames**(int) – How many frames to process at a time. Choose from ‘single’, ‘multiple’, ‘fixed_multiple’ or an integer (an integer should only ever be passed in exceptional circumstances)

**plugin_data_transfer_setup**(copy=None, calc=None)
Set up the plugin data transfer frame parameters. If copy=pData (another PluginData instance) then copy

**set_bytes_per_frame()**
Return the size of a single frame in bytes.

**set_fixed_dimensions**(dims, values)
Fix a data direction to the index in values list.

Parameters
- **dims**(list(int)) – Directions to fix
- **value**(list(int)) – Index of fixed directions

**set_slicing_order**(order)
Reorder the slice dimensions. The fastest changing slice dimension will always be the first one stated in the pattern key slice_dir. The input param is a tuple stating the desired order of slicing dimensions relative to the current order.

---

**savu.data.data_structures.preview module**

**class Preview**(data_obj)
Bases: object

The Data class dynamically inherits from transport specific data class and holds the data array, along with associated information.

**_Preview__add_preview_param**(name, value)

**_Preview__check_preview_indices**()

**_Preview__convert_nprocs**(preview_list)

**_Preview__set_reduced_shape**(starts, stops, steps, chunks)
Set new shape if data is reduced by previewing.

**_Preview__set_starts_stops_steps**(starts, stops, steps, chunks, shapeChange=True, load=False)
Add previewing params to data_info dictionary and set reduced shape.

**_add_preview_defaults**(plist)
Fill in missing values in preview list entries.

Param preview list with entries of the form start[:stop:step:chunk]

Returns preview list with missing values replaced by defaults

Return type list

**_get_preview_indices**(preview_list)
Get preview_list starts, stops, steps, chunks separate components with integer values.

Params preview_list

Returns separate list of starts, stops, steps, chunks integer values

Return type list(list(int))
_get_preview_slice_list()
Amend the axis label values based on the previewing parameters.

_reset_preview()

_unset_preview()
Unset preview (revert=True) if it was only required in the plugin.

convert_indices(idx, dim)
convert keywords to integers.

get_data_obj()

get_starts_stops_steps(key=None)
Returns preview parameter starts, stops, steps, chunks lists.

Keyword Arguments key (str) – the list to return.

Returns if key is none return separate preview_list components, where each list has length equal to number of dataset dimensions, else only the key list.

Return type list(list(int))

set_preview(preview_list, **kwargs)
Reduces the data to be processed to a subset of the original.

Parameters preview (list) – previewing parameters, where len(preview_list) equals the number of data dimensions.

Keyword Arguments revert (bool) – revert input dataset to the original size after plugin processing.

Each preview_list element should be of the form start:stop:step:chunk, where stop, step and chunk are optional (defaults: stop = start + 1, step = 1, chunk = 1) but must be given in that order.

Note:
start:stop[:step] represents the set of indices specified by:

```python
>>> indices = range(start, stop[, step])
```

For more information see range()

start:stop:step:chunk (chunk > 1) represents the set of indices specified by:

```python
>>> a = np.tile(np.arange(start, stop, step), (chunk, 1))
>>> b = np.transpose(np.tile(np.arange(chunk)-chunk/2, (a.shape[1], 1)))(a.shape[1], 1))
>>> indices = np.ravel(np.transpose(a + b))
```

Chunk indicates how many values to take around each value in range(start, stop, step).
It is only available for slicing dimensions.

**Warning:**  If any indices are out of range (or negative) then the list is invalid. When chunk > 1, new start and end values will be:

```python
>>> new_start = start - int(chunk/2)
>>> new_end = range(start, stop, step)[-1] + (step - int(chunk/2))
```
accepted values: Each entry is executed using `eval()` so simple formulas are allowed and may contain the following keywords:

- `:` is a simplification for 0:end:1:1 (all values)
- `mid` is int(shape[dim]/2)-1
- `end` is shape[dim]

**savu.data.data_structures.utils module**

`_deepcopy_data_object (dObj, new_obj)`
Deepcopy data object, associating hdf5 objects that can not be copied.

`get_available_pattern_types ()`
`get_pattern_rank (pattern)`

**savu.data.data_structures.data_types**

**savu.data.data_structures.data_types.data_plus_darks_and_flats module**

`class DataWithDarksAndFlats (data_obj, proj_dim, image_key)`
Bases: `savu.data.data_structures.data_types.base_type.BaseType`

`_DataWithDarksAndFlats__get_data (key)`
`_DataWithDarksAndFlats__get_preview_image_key (slice_list)`
`_DataWithDarksAndFlats__get_preview_index (key)`
`_DataWithDarksAndFlats__get_reduced_index (key, slice_list)`
Get the projection index of a specific image key value when previewing has been applied.

`_base_extra_params ()`
Global class parameter names that are updated outside of `__init__`

`_calc_mean (data)`
`_get_image_key_data_shape ()`
`_getitem_imagekey (idx)`
`_getitem_noimagekey (idx)`
`_override_data_type (data)`
`_set_dark_and_flat ()`
`_set_scale (name, scale)`

`dark_image_key_data ()`
Get the dark data.

`dark_mean ()`
Get the averaged dark projection data.

`flat_image_key_data ()`
Get the flat data.

`flat_mean ()`
Get the averaged flat projection data.
get_dark_flat_slice_list()
get_image_key()
get_index (key, full=False)
    Get the projection index of a specific image key value.

    Params:
    int key  the image key value

get_shape()
    Get full stitched shape of a stack of files

set_dark_scale(dscale)
set_flat_scale(fscale)
update_dark(data)
update_flat(data)

class ImageKey (data_obj, image_key, proj_dim, ignore=None)
Bases: savu.data.data_structures.data_types.data_plus_darks_and_flats.DataWithDarksAndFlats

    This class is used to get data from a dataset with an image key.

    __ignore_image_key_entries (ignore)

    __finish_cloning()

clonedata_args (args, kwargs, extras)
    List the arguments required to clone this datatype

dark()
    Get the dark data.

flat()
    Get the flat data.

map_input_args (args, kwargs, cls, extras)
    List all information required to keep this data set after a plugin has completed (may require conversion to another type)

class NoImageKey (data_obj, image_key, proj_dim, dark=None, flat=None)
Bases: savu.data.data_structures.data_types.data_plus_darks_and_flats.DataWithDarksAndFlats

    This class is used to get data from a dataset with separate darks and flats.

    __post_clone_updates()

    __set_dark_path (path, imagekey=[])

    __set_fake_key (fakekey)

    __set_flat_path (path, imagekey=[])

clonedata_args (args, kwargs, extras)
    Gather all information required to keep this dataset after a plugin has completed (may require a conversion to a different data_type.

dark()
    Get the dark data.

flat()
    Get the flat data.
get_shape()
Get full stitched shape of a stack of files

davu.data.data_structures.data_types.base_type module

class BaseType
Bases: object

__get_extras_vals__(vals)
__update_extra_params__(newObj, extras)
_base_post_clone_updates__(obj, extras)
_get_parameters__(dtype_dict)
_post_clone_updates__
_set_parameters__(args, kwargs, cls, cls_path, extras)
_str_to_value__(obj, val)

add_base_class_with_instance__(base, inst)
Add a base class instance to a class (merging of two data types).

Params
class base a class to add as a base class

Params instance inst an instance of the base class

base_data_args()
Create a dictionary of required input arguments, required for checkpointing.

clonedata_args__(args, kwargs, extras)
Gather all information required to keep this dataset after a plugin has completed (may require a conversion
to a different data_type.

create_next_instance__(newObj)

get_clone_args()

get_map_args()

get_shape()
Get full stitched shape of a stack of files

map_input_args__(args, kwargs, cls, extras)
Gather all information required to recreate a datatype: For checkpointing and cloning

davu.data.data_structures.data_types.image_data module

davu.data.data_structures.data_types.map_3dto4d_h5 module

class Map3dto4dh5__(data_obj, n_angles)
Bases: davu.data.data_structures.data_types.base_type.BaseType

This class converts a 3D dataset to a 4D dataset.

clonedata_args__(args, kwargs, extras)
Gather all information required to keep this dataset after a plugin has completed (may require a conversion
to a different data_type.
get_shape()  
Get full stitched shape of a stack of files

savu.data.data_structures.data_types.mrc module

savu.data.data_structures.data_types.replicate module

class Replicate(data_obj, reps)  
Bases: savu.data.data_structures.data_types.base_type.BaseType
Class to replicate the slice list of a dataset (not the data itself!)

__set_patterns(data_obj, patterns)

__reset()

clone_data_args(args, kwargs, extras)
Gather all information required to keep this dataset after a plugin has completed (may require a conversion
to a different data_type.

get_shape()  
Get full stitched shape of a stack of files

savu.data.data_structures.data_types.stitch_data module

class StitchData(data_obj_list, stack_or_cat, dim, remove=[])  
Bases: savu.data.data_structures.data_types.base_type.BaseType
This class is used to combine multiple data objects.

_get_lists_cat(idx)

_get_lists_stack(idx)

_getitem_cat(obj, sl)

_getitem_stack(obj, sl)

_set_in_slice_list(idx, val_list, entry)

_set_out_slice_list(idx, val_list)

_set_shape()

clone_data_args(args, kwargs, extras)
Gather all information required to keep this dataset after a plugin has completed (may require a conversion
to a different data_type.

dark_mean()  
Get the averaged dark projection data.

flat_mean()  
Get the averaged flat projection data.

get_shape()  
Get full stitched shape of a stack of files
savu.data.transport_data

savu.data.transport_data.base_transport_data module

class BaseTransportData(data_obj, name='BaseTransportData')
Bases: object

Implements functions associated with the transport of the data.

_BaseTransportData__convert_str(val, b_per_p)
_BaseTransportData__get_bool_slice_dir_index(dim, dir_idx)
_BaseTransportData__get_boundaries(nFrames)
_BaseTransportData__get_optimum_distribution(nFrames)

The number of frames each process should retrieve from file at a time.
_BaseTransportData__refine_distribution_for_multi_mfp(mft, size_list, fchoices)

_calc_max_frames_process(nFrames)
_calc_max_frames_transfer(nFrames)

Calculate the number of frames to transfer from file at a time.
_calc_max_frames_transfer_multi(nFrames)

Multiple transfer per process
_calc_max_frames_transfer_single(nFrames)

Only one transfer per process
_find_best_frame_distribution(flist, nframes, nprocs, idx=False)

Determine which of the numbers in the list of possible frame chunks gives the best distribution of frames per process.
_find_closest_lower(vlist, value)

_find_multiples_of_b_that_divide_a(a, b)

Find all positive multiples of b that divide a.
_get_data_obj()
_get_frame_choices(sdir, max_mft)

Find all possible combinations of increasing slice dimension sizes with their product less than max_mft and return a list of these products.
_get_padded_data(input_slice_list)

Fetch the data with relevant padding (as determined by the plugin).
_get_slice_dir_index(dim, boolean=False)
_get_slice_dir_matrix(dim)
_get_slice_list_per_process(expInfo)

A slice list required by the current process.
_set_boundaries()

savu.data.transport_data.hdf5_transport_data module

class Hdf5TransportData(data_obj, name='Hdf5TransportData')
data.transport_data.slice_lists.SliceLists
The Hdf5TransportData class performs the organising and movement of data.

```
_Hdf5TransportData__combine_dicts(d1, d2)
_calc_max_frames_transfer(nFrames)
    Calculate the number of frames to transfer from file at a time.
_get_padded_data(slice_list, end=False)
    Fetch the data with relevant padding (as determined by the plugin).
_get_slice_lists_per_process(dtype)
```

```py
savu.data.transport_data.basic_transport_data module

class BasicTransportData(data_obj, name='BasicTransportData')
           data.transport_data.slice_lists.SliceLists

    The Hdf5TransportData class performs the organising and movement of data.

    _calc_max_frames_transfer(nFrames)
        Calculate the number of frames to transfer from file at a time.

    _get_padded_data(input_slice_list)
        Fetch the data with relevant padding (as determined by the plugin).

    _get_slice_lists_per_process(dtype)
```

```py
savu.data.transport_data.dosna_transport_data module

class DosnaTransportData(data_obj, name='DosnaTransportData')
    Bases: savu.data.transport_data.hdf5_transport_data.Hdf5TransportData

    The DosnaTransportData class performs the organising and movement of data.

```py
savu.data.transport_data.slice_lists module

class GlobalData(dtype, transport)
    Bases: object

    The GlobalData class organises the movement and slicing of the data from file.

    _get_dict()

    _get_dict_in()

    _get_dict_out()

    _get_padded_data(slice_list, end=False)

    _get_slice_list(shape, current_sl=None, pad=False)

class LocalData(dtype, transport_data)
    Bases: object

    The LocalData class organises the slicing of transferred data to give the shape requested by a plugin for each run of ‘process_frames’.

    _LocalData__get_unpad_slice_list(reps)
```
Savu Documentation, Release 2.4

```python
_get_dict()
_get_dict_in()
_get_dict_out()
_get_slice_list()
Splits a file transfer slice list into a list of (padded) slices required for each loop of process_frames.

class SliceLists (name='Hdf5TransportData')
Bases: object

The Hdf5TransportData class performs the organising and movement of data.

.SliceLists__chunk_length_repeat (slice_dirs, shape)
For each slice dimension, determine 3 values relevant to the slicing.

    Returns chunk, length, repeat chunk: how many repeats of the same index value before an
    increment length: the slice dimension length (sequence length) repeat: how many times does
    the sequence of chunked numbers repeat

    Return type [int, int, int]
.SliceLists__get_shape_of_slice_dirs (slice_dirs, shape)
.SliceLists__get_split_frame_entries (slice_list, dims, length)
.SliceLists__get_split_length (max_frames, shape)
.SliceLists__split_frames (slice_list, split_list)
_banked_list (slice_list, max_frames, pad=False)
_fix_list_length (sl, length, dim)
_get_core_slices (core_dirs)
_get_frames_per_process (slice_list)
_get_global_single_slice_list (shape)
_get_local_single_slice_list (shape)
_get_process_data ()
_get_slice_dirs_index (slice_dirs, shape, value, calc=None)
returns a list of arrays for each slice dimension, where each array gives the indices for that slice dimension.

_group_dimension (sl, dim, step)
_group_slice_list_in_multiple_dimensions (slice_list, max_frames, group_dim, pad=False)
    Group the slice list in multiple dimensions - prepare a slice list for file transfer.
_group_slice_list_in_one_dimension (slice_list, max_frames, group_dim, pad=False)
    Group the slice list in one dimension, stopping at boundaries - prepare a slice list for multi-frame plugin
    processing.
_pad_slice_list (slice_list, inc_start_str, inc_stop_str)
    Amend the slice lists to include padding. Includes variations for transfer and process slice lists.
_single_slice_list (nSlices, nDims, core_slice, core_dirs, slice_dirs, fix, index)
_split_list (the_list, size)
```
2.5.2 Plugin API

Information on specific functions, classes, and methods.

savu

savu.tomo_recon module

savu.version module

Get the Savu version

savu.core

savu.core.plugin_runner module

savu.core.transport_setup module

class MPI_setup(options, name='MPI_setup')

    call_mpi__barrier()
    Call MPI_barrier before an experiment is created.

savu.core.utils module

add_base(this, base)
    Add a base class to a class.

    Parameters
    ----------
    this : class
        a class instance
    base : class
        a class to add as a base class

add_base_classes(this, bases)
    Add multiple base classes to a class.

    Parameters
    ----------
    this : class
        a class instance.
    bases : list(class)
        a list of base classes

add_syslog_log_handler(logger, syslog_address, syslog_port)

add_user_log_handler(logger, user_log_path)

add_user_log_level()

docstring_parameter(*sub)
    Decorator to add strings to a doc string.

get_available_gpus()

get_memory_usage_linux(kb=False, mb=True)

    Parameters
    ----------
    * kb – Return the value in Kilobytes
    * mb – Return the value in Megabytes
**Returns**  The string of the value in either KB or MB

Field list ends without a blank line; unexpected unindent.

```python
:rtype str

import_class (class_name)

Import a class.

**Params**  class name

**Returns**  class instance

**Return type**  instance of class_name

logfunction (func)

Decorator to add logging information around calls for use with .

logmethod (func)

Decorator to add logging information around calls for use with .

user_message (message)

user_messages_from_all (header, message_list)

**savu.core.basic_plugin_runner module**

**savu.core.checkpointing module**

**savu.core.transports**

**savu.core.transports.base_transport module**

**savu.core.transports.basic_transport module**

**class BasicTransport**

Implements functions that control the interaction between the data and plugin layers.

```
savu.core.transports.hdf5_transport module

class Hdf5Transport

savu.core.transports.dosna_transport module

savu.data

savu.data.chunking module

class Chunking(exp, patternDict)
   A class to save tomography data to a hdf5 file

savu.data.experiment_collection module

savu.data.plugin_list module

savu.data.framework_citations module

def get_framework_citations()
   return a list of NXcite objects

savu.data.meta_data module

class MetaData(options={}, ordered=False)
   The MetaData class creates a dictionary of all meta data which can be accessed using the get and set methods. It also holds an instance of PluginList.

   delete(entry)
      Delete an entry from the meta data dictionary.

      Parameters
      entry (str) – The dictionary key entry to delete.

   get(maplist, setFlag=False, value=True, units=False)
      Get a value from the meta data dictionary, given its key(s).

      Params
      maplist Dictionary key(s).
      setFlag bool
      value bool
      units bool

      Returns
      Value from the dictionary corresponding to the given key(s)

      Return type
      value

      Dictionaries within dictionaries are accessed by placing successive keys in a list.

   get_dictionary()
      Get the meta_data dictionary.

      Returns
      A dictionary.

      Return type
      dict

   set(name, value)
      Create and set an entry in the meta data dictionary.

      Parameters
• **name**(str or list(str)) – dictionary key(s). If name is a list then each successive name will become an entry in the dictionary which has the previous name as its key.

• **value**(value) – dictionary value

For example,

```python
>>> MetaDataObj.set(['name1', 'name2'], 3)
>>> MetaDataObj.get_dictionary()
{'name1': {'name2': 3}}
```

### savu.data.data_structures

#### savu.data.data_structures.data module

class Data (name, exp)
The Data class dynamically inherits from transport specific data class and holds the data array, along with associated information.

**add_pattern**(dtype, **kwargs)
Add a pattern.

**Keyword Arguments**
- **core_dims**(tuple) – Dimension indices of core dimensions
- **slice_dims**(tuple) – Dimension indices of slice dimensions

**add_volume_patterns**(x, y, z)
Adds volume patterns

**Params**
- **x**(int) dimension to be associated with x-axis
- **y**(int) dimension to be associated with y-axis
- **z**(int) dimension to be associated with z-axis

**amend_axis_label_values**(slice_list)
Amend all axis label values based on the slice_list parameter. This is required if the data is reduced.

**get_axis_label_keys**()
Get axis_label names

**Returns**
A list containing associated axis names for each dimension

**Return type**
list(str)

**get_axis_labels**()
Get axis labels.

**Returns**
Axis labels

**Return type**
list(dict)

**get_core_dimensions**()
Get the core data dimensions associated with the current pattern.

**Returns**
value associated with pattern key core_dims
Return type: tuple

get_data_dimension_by_axis_label(name, contains=False)
Get the dimension of the data associated with a particular axis_label.

Parameters:
- name (str) – The name of the axis_label

Keyword Arguments:
- contains (bool) – Set this flag to true if the name is only part of the axis_label name

Returns: The associated axis number
Return type: int

get_data_patterns()
Get data patterns associated with this data object.

Returns: A dictionary of associated patterns.
Return type: dict

get_itemsize()
Returns bytes per entry

get_name(orig=False)
Get data name.

Keyword Arguments:
- orig (bool) – Set this flag to true to return the original cloned dataset name if this dataset is a clone

Returns: the name associated with the dataset
Return type: str

get_preview()
Get the Preview instance associated with the data object

get_previous_pattern()

get_shape()
Get the dataset shape

Returns: data shape
Return type: tuple

get_slice_dimensions()
Get the slice data dimensions associated with the current pattern.

Returns: value associated with pattern key slice_dims
Return type: tuple

set_axis_labels(*args)
Set the axis labels associated with each data dimension.

Parameters:
- str – Each arg should be of the form name.unit. If name is a data_obj.meta_data entry, it will be output to the final .nxs file.

set_original_shape(shape)
Set the original data shape before previewing

set_shape(shape)
Set the dataset shape.
savu.data.data_structures.data_add_ons module

class DataMapping

A class providing helper functions to multi-modal loaders.

    check_is_map (proj_dir)
    check_is_tomo (proj_dir, rotation)
    get_axes()
    get_motor_type()
    get_motors()
    set_axes(axes)
    set_motor_type(motor_type)
    set_motors(motors)

class Padding(pData)

A class that organises padding of the data. An instance of Padding can be associated with a Data object in a plugin that inherits from BaseFilter, inside savu.plugins.base_filter.BaseFilter.

    set_filter_padding()

    pad_directions(pad_list)

        Pad multiple, individually specified, dimensions.

        param list(dict) pad_list A list of strings of the form ‘dim.pad’, ‘dim.after.pad’ or ‘dim.before.pad’

        • ‘dim’ should be replaced with the dimension to pad
        • ‘pad’ should be replaced with the amount to pad
        • ‘before’ and ‘after’ are optional positional keywords specifying padding ‘before’ or ‘after’ the data for the specified dimension index (if neither are specified BOTH will be padded)

    pad_frame_edges (padding)

        Pad all the edges of a frame of data with the same pad amount (i.e pad in the core dimensions).

        Parameters padding (int) – The pad amount

    pad_mode (mode)

    pad_multi_frames (padding)

        Add extra frames before and after the current frame of data (i.e pad in the fastest changing slice dimension).

        Parameters padding (int) – The pad amount

savu.data.data_structures.data_create module

class DataCreate(name='DataCreate')

Class that deals with creating a data object.

    create_dataset (*args, **kwargs)

        Set up required information when an output dataset has been created by a plugin.

        arg Data A data object
        keyword tuple shape The shape of the dataset
**keyword list axis_labels**  The axis_labels associated with the datasets

**keyword patterns**  The patterns associated with the dataset (optional, see note below)

**keyword type dtype**  Type of the data (optional: Defaults to np.float32)

**keyword bool remove**  Remove from framework after completion (no link in .nxs file) (optional: Defaults to False.)

**keyword bool raw**  Keep dark and flats (ImageKey or NoImageKey)

---

**Note:**

**Creating a dataset**  Each new dataset requires the following information:

- shape
- axis_labels
- patterns

This function can be used to setup the required information in one of two ways:

1. Passing a `Data` object as the only argument: All required information is copied from this data object. For example,

```
>>> out_dataset[0].create_dataset(in_dataset[0])
```

2. Passing kwargs: `shape` and `axis_labels` are required (see above for other optional arguments). For example,

```
>>> out_dataset[0].create_dataset(axis_labels=labels, shape=new_shape)
```

**Warning:** If `pattern` keyword is not used, patterns must be added after `create_dataset()` by calling `add_pattern()`.

---

**Note:**

**shape keyword argument**  Options to pass are:

1. Data object: Copy shape from the Data object.

```
>>> out_dataset[0].create_dataset(axis_labels=labels, shape=in_dataset[0])
```

2. tuple: Define shape explicitly.

```
>>> out_dataset[0].create_dataset(axis_labels=labels, shape=(10, 20, 30))
```

---

**Note:**

**axis_labels keyword argument**  Options to pass are:

1. Data object: Copy all labels from the Data object.
2. `{Data_obj: list}`: Copy labels from the Data object and then remove or insert.

   - To remove dimensions: list_entry = `dim`. For example, to remove the first and last axis_labels from the copied list:
     ```python
     >>> out_dataset[0].create_dataset(axis_labels={in_dataset[0]: ['1', '-1']}, shape=new_shape)
     ```

   - To add/replace dimensions: list_entry = `dim.name.unit`.
     ```python
     >>> out_dataset[0].create_dataset(axis_labels={in_dataset[0]: ['2.det_x.pixel', '3.det_y.pixel']}, shape=new_shape)
     ```

   - To insert dimensions: list_entry = `~dim.name.unit`.
     ```python
     >>> out_dataset[0].create_dataset(axis_labels={in_dataset[0]: ['~2.det_x.pixel', '~3.det_y.pixel']}, shape=new_shape)
     ```

   (or a combination, where each successive step is applied after the previous changes have been made.)

3. list: Where each element is of the form `name.unit`.

   ```python
   >>> out_dataset[0].create_dataset(axis_labels=['rotation.deg', 'det_x.pixel', 'det_y.pixel'], shape=new_shape)
   ```

---

**Note:**

**patterns** keyword argument Options to pass are:

1. Data object: Copy all patterns from the Data object.

   ```python
   >>> out_dataset[0].create_dataset(axis_labels=labels, shape=new_shape, patterns=in_dataset[0])
   ```

2. `{Data_obj: list}`: Copy only the patterns given in the list from the Data object.

   - Copy the patterns: list_entry = `name`
     ```python
     >>> out_dataset[0].create_dataset(axis_labels=labels, shape=new_shape, patterns={in_dataset[0]: ['SINOGRAM', 'PROJECTION']})
     ```

   - Copy patterns but remove dimensions: list_entry = `name1.r1,r2...`:
     ```python
     >>> out_dataset[0].create_dataset(axis_labels=labels, shape=new_shape, patterns={in_dataset[0]: ['SINOGRAM.1', 'PROJECTION.1']})
     ```

   - Copy ALL patterns but remove dimensions: list_entry = `*.r1,r2...`:
```python
>>> out_dataset[0].create_dataset(axis_labels=labels, shape=new_shape, patterns={in_dataset[0]: '*.0'})
```

**get_dtype()**

**set_dtype(dtype)**

```python
def savu.data.data_structures.data_notes module

**axis_labels()**

---

**Note:**

**axis_labels keyword argument** Options to pass are:

1. Data object: Copy all labels from the Data object.

```python
>>> out_dataset[0].create_dataset(axis_labels=in_dataset[0], shape=new_shape)
```

2. {Data_obj: list}: Copy labels from the Data object and then remove or insert.
   - To remove dimensions: list_entry = ‘dim’. For example, to remove the first and last axis_labels from the copied list:

```python
>>> out_dataset[0].create_dataset(axis_labels={in_dataset[0]: ['1', '-1']}, shape=new_shape)
```

   - To add/replace dimensions: list_entry = ‘dim.name.unit’.

```python
>>> out_dataset[0].create_dataset(axis_labels={in_dataset[0]: ['2.det_x.pixel', '3.det_y.pixel']}, shape=new_shape)
```

   - To insert dimensions: list_entry = ‘~dim.name.unit’.

```python
>>> out_dataset[0].create_dataset(axis_labels={in_dataset[0]: ['~2.det_x.pixel', '~3.det_y.pixel']}, shape=new_shape)
```

(or a combination, where each successive step is applied after the previous changes have been made.)

3. list: Where each element is of the form ‘name.unit’.

```python
>>> out_dataset[0].create_dataset(axis_labels=['rotation.deg', 'det_x.pixel', 'det_y.pixel'], shape=new_shape)
```

---

**image_key()**

This is a helper function to be used after `savu.data.data_structures.data_create.DataCreate.create_dataset()`. 

---

2.5. API Documentation
if in_dataset[0] is a plugin input dataset with an image_key and 0 is the data index.

patterns()

Note:

patterns keyword argument  Options to pass are:

1. Data object: Copy all patterns from the Data object.

   >>> out_dataset[0].create_dataset(axis_labels=labels, shape=new_shape, patterns=in_dataset[0])

2. {Data_obj: list}: Copy only the patterns given in the list from the Data object.

   • Copy the patterns: list_entry = 'name'

   >>> out_dataset[0].create_dataset(axis_labels=labels, shape=new_shape, patterns={in_dataset[0]: ['SINOGRAM', 'PROJECTION']})

   • Copy patterns but remove dimensions: list_entry = 'name1.r1,r2...':

   >>> out_dataset[0].create_dataset(axis_labels=labels, shape=new_shape, patterns={in_dataset[0]: ['SINOGRAM.1', 'PROJECTION.1']})

   • Copy ALL patterns but remove dimensions: list_entry = '*.r1,r2...':

   >>> out_dataset[0].create_dataset(axis_labels=labels, shape=new_shape, patterns={in_dataset[0]: '*.0'})

savu.data.data_structures.plugin_data module

class PluginData (data_obj, plugin=None)
The PluginData class contains plugin specific information about a Data object for the duration of a plugin. An instance of the class is encapsulated inside the Data object during the plugin run

get_core_dimensions ()
  Return the position of the core dimensions in relation to the data handed to the plugin.

get_core_shape ()
  Get the shape of the core dimensions only.

  Returns  shape of core dimensions

  Return type  tuple

get_current_frame_idx ()
  Returns the index of the frames currently being processed.

get_data_dimension_by_axis_label (label, contains=False)
  Return the dimension of the data in the plugin that has the specified axis label.
get_frame_limit()

get_padded_shape()
Get the shape of the data (with padding) that is passed to the plugin process_frames method.

get_pattern()
Get the current pattern.

Returns dict of the pattern name against the pattern.

Return type dict

get_pattern_name()
Get the pattern name.

Returns the pattern name

Return type str

get_shape()
Get the shape of the data (without padding) that is passed to the plugin process_frames method.

get_shape_transfer()
Get the shape of the plugin data to be transferred each time.

get_slice_dimension()
Return the position of the slice dimension in relation to the data handed to the plugin.

get_total_frames()
Get the total number of frames to process (all MPI processes).

Returns Number of frames

Return type int

plugin_data_setup(pattern, nFrames, split=None, slice_axis=None)
Setup the PluginData object.

Parameters

• pattern (str) – A pattern name
• nFrames (int) – How many frames to process at a time. Choose from ‘single’, ‘multiple’, ‘fixed_multiple’ or an integer (an integer should only ever be passed in exceptional circumstances)

plugin_data_transfer_setup(copy=None, calc=None)
Set up the plugin data transfer frame parameters. If copy=pData (another PluginData instance) then copy

set_bytes_per_frame()
Return the size of a single frame in bytes.

set_fixed_dimensions(dims, values)
Fix a data direction to the index in values list.

Parameters

• dims(list(int)) – Directions to fix
• value(list(int)) – Index of fixed directions

set_slicing_order(order)
Reorder the slice dimensions. The fastest changing slice dimension will always be the first one stated in the pattern key slice_dir. The input param is a tuple stating the desired order of slicing dimensions relative to the current order.
savu.data.data_structures.preview module

class Preview(data_obj)
The Data class dynamically inherits from transport specific data class and holds the data array, along with associated information.

convert_indices(idx, dim)
convert keywords to integers.

get_data_obj()

get_starts_stops_steps(key=None)
Returns preview parameter starts, stops, steps, chunks lists.

Keyword Arguments key (str) – the list to return.

Returns if key is none return separate preview_list components, where each list has length equal to number of dataset dimensions, else only the key list.

Return type list(list(int))

set_preview(preview_list, **kwargs)
Reduces the data to be processed to a subset of the original.

Parameters preview (list) – previewing parameters, where len(preview_list) equals the number of data dimensions.

Keyword Arguments revert (bool) – revert input dataset to the original size after plugin processing.

Each preview_list element should be of the form start:stop:step:chunk. where stop, step and chunk are optional (defaults: stop = start + 1, step = 1, chunk = 1) but must be given in that order.

Note:

start:stop[:step] represents the set of indices specified by:

```python
>>> indices = range(start, stop[, step])
```

For more information see range()

start:stop:step:chunk (chunk > 1) represents the set of indices specified by:

```python
>>> a = np.tile(np.arange(start, stop, step), (chunk, 1))
>>> b = np.transpose(np.tile(np.arange(chunk)-chunk/2, (a.shape[1], 1)))
>>> indices = np.ravel(np.transpose(a + b))
```

Chunk indicates how many values to take around each value in range(start, stop, step). It is only available for slicing dimensions.

Warning: If any indices are out of range (or negative) then the list is invalid. When chunk > 1, new start and end values will be:

```python
>>> new_start = start - int(chunk/2)
>>> new_end = range(start, stop, step)[-1] + (step - int(chunk/2))
```
**accepted values**: Each entry is executed using `eval()` so simple formulas are allowed and may contain the following keywords:

- `:` is a simplification for `0:end:1:1` (all values)
- `mid` is `int(shape[dim]/2)-1`
- `end` is `shape[dim]`

**savu.data.data_structures.utils module**

- `get_available_pattern_types()`
- `get_pattern_rank(pattern)`

**savu.data.data_structures.data_types**

**savu.data.data_structures.data_types.data_plus_darks_and_flats module**

**class DataWithDarksAndFlats**(data_obj, proj_dim, image_key)

- `dark_image_key_data()`
  Get the dark data.
- `dark_mean()`
  Get the averaged dark projection data.
- `flat_image_key_data()`
  Get the flat data.
- `flat_mean()`
  Get the averaged flat projection data.
- `get_dark_flat_slice_list()`
- `get_image_key()`
- `get_index(key, full=False)`
  Get the projection index of a specific image key value.
  - Params `int key` the image key value
- `get_shape()`
  Get full stitched shape of a stack of files
- `set_dark_scale(dscale)`
- `set_flat_scale(fscale)`
- `update_dark(data)`
- `update_flat(data)`

**class ImageKey**(data_obj, image_key, proj_dim, ignore=None)

This class is used to get data from a dataset with an image key.

- `clone_data_args(args, kwargs, extras)`
  List the arguments required to clone this datatype
- `dark()`
  Get the dark data.
**flat**

Get the flat data.

**map_input_args** (*args, **kwargs, cls, extras*)

List all information required to keep this data set after a plugin has completed (may require conversion to another type)

**class NoImageKey** (*data_obj, image_key, proj_dim, dark=None, flat=None*)

This class is used to get data from a dataset with separate darks and flats.

**clone_data_args** (*args, **kwargs, extras*)

Gather all information required to keep this dataset after a plugin has completed (may require a conversion to a different data_type).

**dark**

Get the dark data.

**flat**

Get the flat data.

**get_shape**

Get full stitched shape of a stack of files

---

**savu.data.data_structures.data_types.base_type module**

**class BaseType**

**add_base_class_with_instance** (*base, inst*)

Add a base class instance to a class (merging of two data types).

- **Params**
  - **class base** a class to add as a base class
  - **Params**
    - **instance inst** an instance of the base class

**base_data_args**

Create a dictionary of required input arguments, required for checkpointing.

**clone_data_args** (*args, **kwargs, extras*)

Gather all information required to keep this dataset after a plugin has completed (may require a conversion to a different data_type).

**create_next_instance** (*newObj*)

**get_clone_args**

**get_map_args**

**get_shape**

Get full stitched shape of a stack of files

**map_input_args** (*args, **kwargs, cls, extras*)

Gather all information required to recreate a datatype: For checkpointing and cloning

---

**savu.data.data_structures.data_types.image_data module**

**savu.data.data_structures.data_types.map_3dto4d_h5 module**

**class Map3dto4dh5** (*data_obj, n_angles*)

This class converts a 3D dataset to a 4D dataset.
**clone_data_args** *(args, kwargs, extras)*
Gather all information required to keep this dataset after a plugin has completed (may require a conversion to a different data_type).

**get_shape**
Get full stitched shape of a stack of files

**savu.data.data_structures.data_types.mrc module**

**savu.data.data_structures.data_types.replicate module**

**class Replicate** *(data_obj, reps)*
Class to replicate the slice list of a dataset (not the data itself!)

**clone_data_args** *(args, kwargs, extras)*
Gather all information required to keep this dataset after a plugin has completed (may require a conversion to a different data_type).

**get_shape**
Get full stitched shape of a stack of files

**savu.data.data_structures.data_types.stitch_data module**

**class StitchData** *(data_obj_list, stack_or_cat, dim, remove=[])*
This class is used to combine multiple data objects.

**clone_data_args** *(args, kwargs, extras)*
Gather all information required to keep this dataset after a plugin has completed (may require a conversion to a different data_type).

**dark_mean**
Get the averaged dark projection data.

**flat_mean**
Get the averaged flat projection data.

**get_shape**
Get full stitched shape of a stack of files

**savu.data.transport_data**

**savu.data.transport_data.base_transport_data module**

**class BaseTransportData** *(data_obj, name='BaseTransportData')*
Implements functions associated with the transport of the data.

**savu.data.transport_data.hdf5_transport_data module**

**class Hdf5TransportData** *(data_obj, name='Hdf5TransportData')*
The Hdf5TransportData class performs the organising and movement of data.
The Hdf5TransportData class performs the organising and movement of data.

The DosnaTransportData class performs the organising and movement of data.

The GlobalData class organises the movement and slicing of the data from file.

The LocalData class organises the slicing of transferred data to give the shape requested by a plugin for each run of ‘process_frames’.  

The Hdf5TransportData class performs the organising and movement of data.

The base class from which all plugins should inherit. :param in_datasets: Create a list of the dataset(s) to process. Default: []. :param out_datasets: Create a list of the dataset(s) to create. Default: [].

This method is called immediately after post_process().

This method is called after the plugin has been created by the pipeline framework as a pre-processing step.

This method is called directly after each call to process frames and before returning the data to file.

This method is called before each call to process frames

Provide a summary to the user for the result of the plugin.

e.g.

- Warning, the sample may have shifted during data collection
- Filter operated normally

**Returns** A list of string summaries

`final_parameter_updates()`
An opportunity to update the parameters after they have been set.

`get_citation_information()`
Gets the Citation Information for a plugin

**Returns** A populated `savu.data.plugin_info.CitationInformation`

`get_current_slice_list()`
Get the slice list of the current frame being processed.

`get_global_frame_index()`
Get the global frame index.

`get_parameters(name)`
Return a plugin parameter

- **Params** `str name` parameter name (dictionary key)
- **Returns** the associated value in `self.parameters`
- **Return type** dict value

`get_process_frames_counter()`

`get_slice_dir_reps(nData)`
Return the periodicity of the main slice direction.

- **Params** `int nData` The number of the dataset in the list.

`initialise(params, exp, check=False)`

`initialise_parameters()`

`nClone_datasets()`
The number of output datasets that have an clone - i.e. they take it in turns to be used as output in an iterative plugin.

`nFrames()`
The number of frames to process during each call to `process_frames`.

`nInput_datasets()`
The number of datasets required as input to the plugin

**Returns** Number of input datasets

`nOutput_datasets()`
The number of datasets created by the plugin

**Returns** Number of output datasets

`plugin_process_frames(data)`

`post_process()`
This method is called after the process function in the pipeline framework as a post-processing step. All processes will have finished performing the main processing at this stage.

- **Parameters** `exp (experiment class instance)` – An experiment object, holding input and output datasets

`pre_process()`
This method is called immediately after `base_pre_process()`.
process_frames \( (data) \)

This method is called after the plugin has been created by the pipeline framework and forms the main processing step

**Parameters**
- \( data \) (list(np.array)) – A list of numpy arrays for each input dataset.

set_current_slice_list \( (sl) \)

set_filter_padding \( (in\_data, out\_data) \)

Should be overridden to define how wide the frame should be for each input data set

set_global_frame_index \( (frame\_idx) \)

set_preview \( (data, params) \)

setup()

This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).

**savu.plugins.plugin_datasets module**

class PluginDatasets(*args, **kwargs)

The base class from which all plugins should inherit.

check_nDatasets \( (names, nSets, dtype, clones=0) \)

get()

Get a list of meta_data objects associated with the in/out_datasets.

**Returns**
- All MetaData objects associated with out data objects.
- **Return type** list(MetaData(in_datasets)), list(MetaData(out_datasets))

get_datasets()

Get PluginData objects associated with in_datasets and out_datasets.

**Returns**
- Two lists of all PluginData objects associated with in_datasets and out_datasets respectively.
- **Return type** list(PluginData(in_datasets)), list(PluginData(out_datasets))

get_in_datasets()

Get Data objects associated with the in_datasets.

**Returns**
- A list of all Data objects associated with in_datasets for the current plugin.
- **Return type** list(Data)

get_in_meta_data()

Get a list of meta_data objects associated with the in_datasets.

**Returns**
- All MetaData objects associated with in data objects.
- **Return type** list(MetaData)

get_out_datasets()

Get Data objects associated with the out_datasets.

**Returns**
- A list of all Data objects associated with out_datasets for the current plugin.
- **Return type** list(Data)

get_out_meta_data()

Get a list of meta_data objects associated with the out_datasets.
Returns All MetaData objects associated with out data objects.

Return type list(MetaData)

get_plugin_datasets ()
Get PluginData objects associated with in_datasets and out_datasets.

Returns Two lists of all PluginData objects associated with in_datasets and out_datasets respectively.

Return type list(PluginData(in_datasets)), list(PluginData(out_datasets))

get_plugin_in_datasets ()
Get PluginData objects associated with the in_datasets.

Returns A list of all PluginData objects associated with in_datasets for the current plugin.

Return type list(PluginData)

get_plugin_out_datasets ()
Get PluginData objects associated with the out_datasets.

Returns A list of all PluginData objects associated with out_datasets for the current plugin.

Return type list(PluginData)

savu.plugins.plugin_datasets_notes module

datasets_notes ()
Get {0} objects associated with the {1}_datasets.

Returns A list of all {0} objects associated with {1}_datasets for the current plugin.

Return type list({0})

mData_notes ()
Get a list of meta_data objects associated with the {0}_datasets.

Returns All MetaData objects associated with {0} data objects.

Return type list(MetaData)

two_datasets_notes ()
Get {0} objects associated with in_datasets and out_datasets.

Returns Two lists of all {0} objects associated with in_datasets and out_datasets respectively.

Return type list({0}(in_datasets)), list({0}(out_datasets))

savu.plugins.utils module

blockPrint ()
Disable printing to stdout

dawn_compatible (plugin_output_type=2)

enablePrint ()
Enable printing to stdout

get_plugin (plugin_name, params, exp, check=False)
Get an instance of the plugin class and populate default parameters.

Parameters plugin_name (str.) – Name of the plugin to import
**Returns** An instance of the class described by the named plugin.

```python
get_plugins_paths (examples=True)
```
This gets the plugin paths, but also adds any that are not on the pythonpath to it.

```python
is_template_param (param)
```
Identifies if the parameter should be included in an input template and returns the default value of the parameter if it exists.

```python
load_class (name, cls_name=None)
```
Returns an instance of the class associated with the module name.

**Parameters**
- **name** – Module name or path to a module file
- **cls_name** – Name of class

**Returns**
An instance of the class associated with module.

```python
parse_array_index_as_string (string)
```

```python
parse_config_string (string)
```

```python
plugin_loader (exp, plugin_dict, check=False)
```

```python
register_plugin (clazz)
```
decorator to add plugins to a central register

```python
savu.plugins.absorption_corrections
```

```python
savu.plugins.absorption_corrections.base_absorption_correction module
```

```python
savu.plugins.absorption_corrections.mc_near_absorption_correction module
```

```python
savu.plugins.alignment
```

```python
savu.plugins.alignment.projection_shift module
```

```python
savu.plugins.alignment.projection_vertical_alignment module
```

```python
class ProjectionVerticalAlignment
```
Correct for vertical shift over projection images.

**Config_warn** Requires the PluginShift plugin to precede it.

```python
pre_process ()
```
This method is called immediately after base_pre_process().

```python
process_frames (data)
```
This method is called after the plugin has been created by the pipeline framework and forms the main processing step

**Parameters**
- **data** (`list(np.array)`) – A list of numpy arrays for each input dataset.

```python
savu.plugins.alignment.sinogram_alignment module
```

```python
savu.plugins.alignment.sinogram_clean module
```

```python
class SinogramClean
```
A plugin to calculate the centre of rotation using the Vo Method
Parameters

- **ratio** – The ratio between the size of object and FOV of the camera. Default: 2.0.
- **row_drop** – Drop lines around vertical center of the mask. Default: 20.
- **out_datasets** – The default names. Default: [].

```python
get_max_frames()
```

```python
process_frames(data)
```
This method is called after the plugin has been created by the pipeline framework and forms the main processing step

**Parameters**

- **data** (*list (np.array)*) – A list of numpy arrays for each input dataset.

*savu.plugins.analysis*

*savu.plugins.analysis.stxm_analysis module*

**class StxmAnalysis**
This plugin performs basic STXM analysis of diffraction patterns. 

- **mask_file** - takes in a mask currently in hdf format. Default: None.
- **mask_path** - path to the mask inside the file. Default: '/mask'.
- **threshold** - intensity threshold for the dark field. Default: 0.05.
- **out_datasets** - A. Default: ['bf','df','dpc_x','dpc_y','combined_dpc'].

```python
filter_frames(data)
```

```python
get_max_frames()
```

```python
nOutput_datasets()
```
The number of datasets created by the plugin

**Returns**

- **Number of output datasets**

```python
pre_process()
```
This method is called immediately after base_pre_process().

```python
setup()
```
This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).

*savu.plugins.analysis.base_analysis module*

**class BaseAnalysis** *(name = 'BaseAnalysis')*
This setup method in this class needs to be updated

```python
get_max_frames()
```

```python
get_plugin_pattern()
```

```python
nInput_datasets()
```
The number of datasets required as input to the plugin

**Returns**

- **Number of input datasets**

```python
nOutput_datasets()
```
The number of datasets created by the plugin

**Returns**

- **Number of output datasets**
setup()
This method is first to be called after the plugin has been created. It determines input/output datasets and
plugin specific dataset information such as the pattern (e.g. sinogram/projection).

savu.plugins.analysis.histogram module

class Histogram
This plugin uses peakutils to find peaks in spectra. This is then metadata.

get_max_frames()

process_frames(data)
This method is called after the plugin has been created by the pipeline framework and forms the main
processing step

Parameters data (list(np.array)) – A list of numpy arrays for each input dataset.

setup()
This method is first to be called after the plugin has been created. It determines input/output datasets and
plugin specific dataset information such as the pattern (e.g. sinogram/projection).

savu.plugins.analysis.stats module

class Stats
Parameters

• out_datasets – the output dataset. Default: ['stats'].

• required_stats – create a list of required stats calcs. Default: ['max'].

• direction – which direction to perform this. Default: ‘PROJECTION’.

get_max_frames()

post_process()
This method is called after the process function in the pipeline framework as a post-processing step. All
processes will have finished performing the main processing at this stage.

Parameters exp (experiment class instance) – An experiment object, holding input
and output datasets

process_frames(data)
This method is called after the plugin has been created by the pipeline framework and forms the main
processing step

Parameters data (list(np.array)) – A list of numpy arrays for each input dataset.

setup()
This method is first to be called after the plugin has been created. It determines input/output datasets and
plugin specific dataset information such as the pattern (e.g. sinogram/projection).

savu.plugins.azimuthal_integrators

savu.plugins.azimuthal_integrators.base_azimuthal_integrator module

savu.plugins.azimuthal_integrators.pyfai_azimuthal_integrator_with_bragg_filter module
savu.plugins.azimuthal_integrators.pyfai_azimuthal_integrator module

savu.plugins.azimuthal_integrators.pyfai_azimuthal_integrator_separate module

savu.plugins.basic_operations

savu.plugins.basic_operations.basic_operations module

class BasicOperations
    A class that performs basic mathematical operations on datasets. How should the information be passed to the plugin?

    Parameters
    • operations -- Operations to perform. Default: [].
    • pattern -- Pattern associated with the datasets. Default: ‘PROJECTION’.

get_max_frames()

nInput_datasets()
    The number of datasets required as input to the plugin

    Returns  Number of input datasets

nOutput_datasets()
    The number of datasets created by the plugin

    Returns  Number of output datasets

pre_process()
    This method is called immediately after base_pre_process().

process_frames(data)
    This method is called after the plugin has been created by the pipeline framework and forms the main processing step

    Parameters  data (list(np.array)) -- A list of numpy arrays for each input dataset.

setup()
    Initial setup of all datasets required as input and output to the plugin. This method is called before the process method in the plugin chain.

savu.plugins.basic_operations.no_process_plugin module

class NoProcessPlugin
    The base class from which all plugins should inherit. :param pattern: Explicitly state the slicing pattern. Default: None. :param dummy: Dummy parameter for testing. Default: 10.

get_max_frames()

nInput_datasets()
    The number of datasets required as input to the plugin

    Returns  Number of input datasets

nOutput_datasets()
    The number of datasets created by the plugin

    Returns  Number of output datasets
process_frames(data)
This method is called after the plugin has been created by the pipeline framework and forms the main processing step

Parameters
data(list(np.array)) – A list of numpy arrays for each input dataset.

setup()
Initial setup of all datasets required as input and output to the plugin. This method is called before the process method in the plugin chain.

savu.plugins.basic_operations.arithmetic_operations module

Error in “py:module” directive: maximum 1 argument(s) allowed, 2 supplied.

.. py:module:: savu.plugins.basic_operations.arithmetic_operations
   :noindex:

   .. module:: perform elementary arithmetic operations on data: addition, subtraction, multiplication and division
   :platform: Unix
   :synopsis: perform elementary arithmetic operations on data: addition, subtraction, multiplication and division

class ArithmeticOperations
   Basic arithmetic operations on data: addition, subtraction, multiplication and division. Operations can be performed by extracting scalars from METADATA (min, max, mean) OR providing a scalar value.

   Parameters

   • scalar_value – A scalar value value for arithmetic operation (it not in metadata). Default: None.
   • operation – arithmetic operation to apply to data, choose from addition, subtraction, multiplication and division. Default: ‘division’.
   • metadata_value – A type of scalar extracted from metadata (min, max, mean). Default: ‘max’.

get_max_frames()

nInput_datasets()
The number of datasets required as input to the plugin

   Returns
   Number of input datasets

nOutput_datasets()
The number of datasets created by the plugin

   Returns
   Number of output datasets

pre_process()
This method is called immediately after base_pre_process().

process_frames(data)
This method is called after the plugin has been created by the pipeline framework and forms the main processing step

   Parameters
data(list(np.array)) – A list of numpy arrays for each input dataset.
**setup()**

This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).

### savu.plugins.basic_operations.data_rescale module

### savu.plugins.basic_operations.get_data_statistics module

Error in “module” directive: maximum 1 argument(s) allowed, 14 supplied.

```
.. module:: A plugin to calculate global statistics (max, min sum, mean) of the input data
   :platform: Unix
   :synopsis: A plugin to calculate global statistics (max, min, sum, mean) of the input data
```

```python
class GetDataStatistics
    Collect input data global statistics.
    Parameters out_datasets – The default names. Default: ['data, data_statistics'].
    get_max_frames()
    nInput_datasets()
        The number of datasets required as input to the plugin
        Returns Number of input datasets
    nOutput_datasets()
        The number of datasets created by the plugin
        Returns Number of output datasets
    post_process()
        This method is called after the process function in the pipeline framework as a post-processing step. All processes will have finished performing the main processing at this stage.
        Parameters exp(experiment class instance) – An experiment object, holding input and output datasets
    processFrames(data)
        This method is called after the plugin has been created by the pipeline framework and forms the main processing step
        Parameters data(list(np.array)) – A list of numpy arrays for each input dataset.
    setup()
        This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).
```

### savu.plugins.centering

### savu.plugins.centering.vo_centering module

### savu.plugins.centering.vo_centering_iterative module
savu.plugins.centering.vo_centering_gpu module

savu.plugins.component_analysis

savu.plugins.component_analysis.base_component_analysis module

class BaseComponentAnalysis(name)
    A base plugin for doing component analysis. This sorts out the main features of a component analysis

    Parameters
    
    • in_datasets – A list of the dataset(s) to process. Default: [].
    • out_datasets – A list of the dataset(s) to process. Default: ['scores', 'eigenvectors'].
    • number_of_components – The number expected components. Default: 3.
    • chunk – The chunk to work on. Default: ‘SINOGRAM’.
    • whiten – To subtract the mean or not. Default: 1.

    get_max_frames()
    get_plugin_pattern()

    nInput_datasets()
        The number of datasets required as input to the plugin

        Returns Number of input datasets

    nOutput_datasets()
        The number of datasets created by the plugin

        Returns Number of output datasets

    remove_nan_inf(data)
        converts the nans to nums and sets the infs to the max float size not strictly true, but does allow fitting to take place

    setup()
        This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).

savu.plugins.component_analysis.ica module

savu.plugins.component_analysis.pca module

savu.plugins.corrections

savu.plugins.corrections.base_correction module

class BaseCorrection(name='BaseCorrection')
    A base class for dark and flat field correction plugins.

    get_max_frames()

    nInput_datasets()
        The number of datasets required as input to the plugin

        Returns Number of input datasets
nOutput_datasets()
    The number of datasets created by the plugin

    Returns Number of output datasets

setup()
    Initial setup of all datasets required as input and output to the plugin. This method is called before the
    process method in the plugin chain.

savu.plugins.corrections.dark_flat_field_correction module

class DarkFlatFieldCorrection
    A Plugin to apply a simple dark and flat field correction to data. :
    :param pattern: Data processing pattern is 'PROJECTION' or 'SINOGRAM'. Default: 'PROJECTION'. :
    :param lower_bound: Set all values below the lower_bound to this value. Default: None. :
    :param upper_bound: Set all values above the upper bound to this value. Default: None. :
    :param warn_proportion: Output a warning if this proportion of values, or greater, are
    below and/or above the lower/upper bounds, e.g enter 0.05 for 5%. Default: 0.05.

correct_proj(data)
correct_sino(data)

executive_summary()
    Provide a summary to the user for the result of the plugin.

    e.g.
    • Warning, the sample may have shifted during data collection
    • Filter operated normally

    Returns A list of string summaries

fixed_flag()

pre_process()
    This method is called immediately after base_pre_process().

savu.plugins.corrections.monitor_correction module

class MonitorCorrection
    corrects the data to the monitor counts. This plugin corrects data[0] from data[1] by dividing. We
    allow a scale and offset due to I18’s uncalibrated ic

    Parameters
    • in_datasets – A list of the dataset(s) to process. Default: ['to_be_corrected','monitor'].
    • nominator_scale –
      a. Default: 1.0.
    • nominator_offset –
      b. Default: 0.0.
    • denominator_scale –
      c. Default: 1.0.
- **denominator_offset**
  - d. Default: 0.0.

- **pattern** – the pattern to apply it to. Default: "PROJECTION".

    get_max_frames()

    nInput_datasets()
    The number of datasets required as input to the plugin
    Returns Number of input datasets

    nOutput_datasets()
    The number of datasets created by the plugin
    Returns Number of output datasets

    process_frames(data)
    This method is called after the plugin has been created by the pipeline framework and forms the main
    processing step

    Parameters data (list(np.array)) – A list of numpy arrays for each input dataset.

    setup()
    This method is first to be called after the plugin has been created. It determines input/output datasets and
    plugin specific dataset information such as the pattern (e.g. sinogram/projection).

savu.plugins.corrections.convert_360_180_sinogram module

savu.plugins.corrections.time_based_correction module

class TimeBasedCorrection (name='TimeBasedCorrection')
Apply a time-based dark and flat field correction to data.

Parameters in_range – Set to True if you require values in the range [0, 1]. Default: False.

    calc_average(data, key)

    calculate_dark_field(frames, distance)

    calculate_flat_field(frames, distance)

    find_nearest_frames(idx_list, value)
    Find the index of the two entries that ‘value’ lies between in ‘idx_list’ and calculate the distance between
each of them.

    get_max_frames()

    in_range(data, flat)

    pre_process()
    This method is called immediately after base_pre_process().

    process_frames(data)
    This method is called after the plugin has been created by the pipeline framework and forms the main
    processing step

    Parameters data (list(np.array)) – A list of numpy arrays for each input dataset.
savu.plugins.corrections.camera_rot_correction module

savu.plugins.corrections.distortion_correction module

savu.plugins.corrections.monitor_correction_nd module

class MonitorCorrectionNd
corrects the data to the monitor counts. This plugin corrects data[0] from data[1] by dividing. We allow a scale and offset due to I18's uncalibrated ic:

:param in_datasets: A list of the dataset(s) to process. Default: ['to_be_corrected','monitor'].
:param nominator_offset: b. Default: 0.0.
:param denominator_scale: c. Default: 1.0.
:param denominator_offset: d. Default: 0.0.
:param data_pattern: the pattern to apply it to. Default: "SPECTRUM".
:param monitor_pattern: the pattern to apply it to. Default: "CHANNEL".

filter_frames (data)

nInput_datasets ()
The number of datasets required as input to the plugin

Returns Number of input datasets

nOutput_datasets ()
The number of datasets created by the plugin

Returns Number of output datasets

setup ()
This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).

savu.plugins.corrections.subpixel_shift module

savu.plugins.corrections.time_based_plus_drift_correction module

savu.plugins.corrections.timeseries_field_corrections module

synopsis A Plugin to apply a simple dark and flatfield correction to some raw timeseries data

class TimeseriesFieldCorrections
A Plugin to apply a simple dark and flatfield correction to some raw timeseries data

executive_summary ()
Provide a summary to the user for the result of the plugin.

e.g.

- Warning, the sample may have shifted during data collection
- Filter operated normally

Returns A list of string summaries

pre_process ()
This method is called immediately after base_pre_process().

process_frames (data)
This method is called after the plugin has been created by the pipeline framework and forms the main processing step
Parameters **data** (*list(np.array)*) – A list of numpy arrays for each input dataset.

```
savu.plugins.corrections.xrd_absorption_approximation module
savu.plugins.corrections.distortion_correction_deprecated module
savu.plugins.developing
savu.plugins.developing.testing_sino_align module
savu.plugins.driver
savu.plugins.driver.all_cpus_plugin module

class AllCpusPlugin
    The base class from which all plugins should inherit.

    **process** (*data, output, processes, process*)
    This method is called after the plugin has been created by the pipeline framework, self.available_cpus contains the number of cpus which can be used by this process

    Parameters
    • **data** (*savu.data.data_structures*) – The input data object.
    • **data** – The output data object
    • **processes** – The number of processes which will be doing the work
    • **path** (*int*) – The specific process which we are

    **run_process** (*data, output, processes, process*)
```

```
savu.plugins.driver.cpu_plugin module

class CpuPlugin
    The base class from which all plugins should inherit.
```

```
savu.plugins.driver.gpu_plugin module
```

```
savu.plugins.driver.plugin_driver module

class PluginDriver
    The base class from which all plugins should inherit.

    **get_communicator**()
```

```
savu.plugins.driver.base_driver module

class BaseDriver
    The base class from which all driver plugins should inherit.
```
get_communicator()
   This method should return an MPI communicator

plugin_barrier()  

savu.plugins.driver.basic_driver module

class BasicDriver
   The base class from which all plugins should inherit.
   get_communicator()
   get_mem_multiply()
   plugin_barrier(msg="")

savu.plugins.driver.multi_threaded_plugin module

class MultiThreadedPlugin
   Initiates a multi-threaded plugin on one core of each node.
   get_mem_multiply()
   min_max_cpus()
      Sets the bounds on the number of CPUs required by the plugin, such that if bounds=[b1, b2] then b1 is the lower bound and b2 is the upper bound. Set each entry to None if there are no bounds.
   min_max_gpus()
      Sets the bounds on the number of GPUs required by the plugin, such that if bounds=[b1, b2] then b1 is the lower bound and b2 is the upper bound. Set each entry to None if there are no bounds and to 0 if no GPUs are required.

savu.plugins.driver.iterative_plugin module

class IterativePlugin
   Allows the plugin to be repeated, keeping the parameters from the previous iteration.
   create_clone(clone, data)
   get_alternating_datasets()
   get_iteration()
   get_original_datasets()
   set_alternating_datasets(d1, d2)
   set_alternating_patterns(patterns)
   set_iteration_datasets(itr, in_data, out_data, pattern=None)
   set_iterations(nIterations)
   set_processing_complete()
**savu.plugins.driver.single_node_multi_threaded_plugin module**

**savu.plugins.filters**

**savu.plugins.filters.denoise_bregman_filter module**

**savu.plugins.filters.band_pass module**

```python
class BandPass
    A plugin to filter each frame with a BandPass T
    Parameters
    • blur_width – Kernel Size. Default: (0, 3, 3).
    • type – filter type (High|Low). Default: ‘High’.

    process_frames(data)
    The second method we need to implement from the Filter class and the part of the code that actually does all the work. The input here ‘data’ will contain the 3D block of data to process, and we need to return the data for the single frame in the middle of this. In this case we use the scipy median filter with the ‘kernel_size’ parameter, and return the same size data as you had originally.
```

**savu.plugins.filters.base_filter module**

```python
class BaseFilter(name='BaseFilter')
    A Plugin to apply a simple dark and flatfield correction to some raw timeseries data

    get_max_frames()

    get_plugin_pattern()

    nInput_datasets()
    The number of datasets required as input to the plugin
    Returns Number of input datasets

    nOutput_datasets()
    The number of datasets created by the plugin
    Returns Number of output datasets

    raw_data()
    Return True if the output dataset should retain ImageKey/NoImageKey instances if they exist, i.e. keep the darks and flats NB. This is only available if out_dataset is created from an in_dataset

    setup()
    This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).
```

**savu.plugins.filters.paganin_filter module**

**savu.plugins.filters.ccpi_denoising_cpu module**

**savu.plugins.filters.ccpi_denoising_gpu module**
savu.plugins.filters.list_to_projections module

savu.plugins.filters.dials_find_spots module

savu.plugins.filters.find_peaks module

savu.plugins.filters.median_filter module

savu.plugins.filters.dezinger module

class Dezinger
A plugin for cleaning x-ray strikes based on statistical evaluation of the near neighbourhood:

:ivar: outlier_mu: Threshold for defecting outliers, greater is less sensitive. Default: 10.0.

:ivar: kernel_size: Number of frames included in average. Default: 5.

:ivar: mode: output mode, 0=normal 5=zinger strength 6=zinger yes/no. Default: 0.

executive_summary()
Provide a summary to the user for the result of the plugin.

e.g.

- Warning, the sample may have shifted during data collection
- Filter operated normally

Returns A list of string summaries

get_max_frames()

post_process()
This method is called after the process function in the pipeline framework as a post-processing step. All processes will have finished performing the main processing at this stage.

Parameters

exp (experiment class instance) – An experiment object, holding input and output datasets

pre_process()
This method is called immediately after base_pre_process().

process_frames(data)
This method is called after the plugin has been created by the pipeline framework and forms the main processing step

Parameters

data (list(np.array)) – A list of numpy arrays for each input dataset.

raw_data()
Return True if the output dataset should retain ImageKey/NoImageKey instances if they exist, i.e. keep the darks and flats NB. This is only available if out_dataset is created from an in_dataset

set_filter_padding(in_data, out_data)
Should be overridden to define how wide the frame should be for each input data set

savu.plugins.filters.poly_background_estimator module

class PolyBackgroundEstimator
This plugin uses peakutils to find peaks in spectra. This is then metadata:

:ivar: out_datasets: Create a list of the dataset(s). Default: ['Peaks'].


:ivar: MaxIterations: max

```
get_max_frames()
```

```
poly_background_estimator(xdata, ydata, n=2, weights=None, maxIterations=12, pvalue=0.9, fixed=False)
```

Background estimator based on orthogonal polynomials

Input: xdata, ydata (numpy arrays of same length) pvalue : ratio of variance in poly to poly value at which to stop. 0.9 default

Output: background, polynomial weights, polynomials


```
process_frames(data)
```

This method is called after the plugin has been created by the pipeline framework and forms the main processing step

Parameters data (list(np.array)) – A list of numpy arrays for each input dataset.

```
setup()
```

This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).

```
savu.plugins.filters.dezinger_simple module
```

class DezingerSimple

A plugin for cleaning x-ray strikes based on statistical evaluation of the near neighbourhood :param outlier_mu: Threshold for detecting outliers, greater is less sensitive. Default: 1000.0. :param kernel_size: Number of frames included in average - if the number is not odd use kernel_size+1. Default: 5.

```
executive_summary()
```

Provide a summary to the user for the result of the plugin.

e.g.

- Warning, the sample may have shifted during data collection
- Filter operated normally

Returns A list of string summaries

```
get_max_frames()
```

Setting nFrames to multiple with an upper limit of 4 frames.

```
pre_process()
```

This method is called immediately after base_pre_process().

```
process_frames(data)
```

This method is called after the plugin has been created by the pipeline framework and forms the main processing step

Parameters data (list(np.array)) – A list of numpy arrays for each input dataset.

```
raw_data()
```

Return True if the output dataset should retain ImageKey/NoImageKey instances if they exist, i.e. keep the darks and flats NB. This is only available if out_dataset is created from an in_dataset
**set_filter_padding** *(in_data, out_data)*

Should be overridden to define how wide the frame should be for each input data set.

---

**savu.plugins.filters.dezinger_sinogram module**

Error in “module” directive: maximum 1 argument(s) allowed, 11 supplied.

```plaintext
.. module:: Remove zingers (caused by scattered X-rays hitting the CCD chip directly)
    :platform: Unix
    :synopsis: A plugin working in sinogram space to removes zingers
```

**class DezingerSinogram**

Method to remove scratches in the reconstructed image caused by zingers:

- **param tolerance**: Threshold for detecting zingers, greater is less sensitive. Default: 0.08.

  **pre_process()**
  
  This method is called immediately after base_pre_process().

  **process_frames**(data)
  
  This method is called after the plugin has been created by the pipeline framework and forms the main processing step.

  - **Parameters data** *(list(np.array))*: A list of numpy arrays for each input dataset.

  **setup()**
  
  This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).

---

**savu.plugins.filters.pymca module**

**savu.plugins.filters.quantisation_filter module**

**class QuantisationFilter**

A plugin to quantise an image into discrete levels.

**Parameters**

- **explicit_min_max**: False if min/max intensity comes from the metadata, True if it’s user-defined. Default: False.
- **min_intensity**: Global minimum intensity. Default: 0.
- **max_intensity**: Global maximum intensity. Default: 65535.
- **levels**: Number of levels. Default: 5.

  **pre_process()**
  
  This method is called immediately after base_pre_process().

  **process_frames** *(data)*
  
  The second method we need to implement from the Filter class and the part of the code that actually does all the work. The input here ‘data’ will contain the 3D block of data to process, and we need to return the data for the single frame in the middle of this. In this case we return the same size data as you had originally.
**savu.plugins.filters.spectrum_crop module**

```python
class SpectrumCrop
crops a spectrum to a range

Parameters

- **crop_range** – range to crop to. Default: [2., 18.].
- **axis** – range to crop to. Default: “energy”.

**get_max_frames()**

**nOutput_datasets()**
The number of datasets created by the plugin

**process_frames(data)**
This method is called after the plugin has been created by the pipeline framework and forms the main processing step

**setup()**
This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).
```

**savu.plugins.filters.strip_background module**

```python
class StripBackground
1D background removal. PyMca magic function

Parameters

- **iterations** – Number of iterations. Default: 100.
- **window** – Half width of the rolling window. Default: 10.
- **SG_filter_iterations** – How many iterations until smoothing occurs. Default: 5.
- **SG_width** – What is the savitzgy golay window. Default: 35.
- **SG_polyorder** – What is the savitzgy-golay poly order. Default: 5.
- **out_datasets** – A list of the dataset(s) to process. Default: ['in_datasets[0]', 'background'].

**get_max_frames()**

**nOutput_datasets()**
The number of datasets created by the plugin

**process_frames(data)**
This method is called after the plugin has been created by the pipeline framework and forms the main processing step

**setup()**
This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).```
savu.plugins.filters.threshold_filter module

class ThresholdFilter
A plugin to quantise an image into discrete levels.

Parameters

- **explicit_threshold** – False if plugin calculates black/white threshold, True if it’s user-defined. Default: True.

- **intensity_threshold** – Threshold for black/white quantisation. Default: 32768.

pre_process()
This method is called immediately after base_pre_process().

process_frames(data)
The second method we need to implement from the Filter class and the part of the code that actually does all the work. The input here ‘data’ will contain the 3D block of data to process, and we need to return the data for the single frame in the middle of this. In this case we return the same size data as you had originally.

savu.plugins.filters.hilbert_filter module

class HilbertFilter
A plugin to apply Hilbert filter (for phase gradient image) on projections.

get_max_frames()

pre_process()
This method is called immediately after base_pre_process().

process_frames(data)
This method is called after the plugin has been created by the pipeline framework and forms the main processing step

Parameters data (list(np.array)) – A list of numpy arrays for each input dataset.

savu.plugins.filters.image_interpolation module

class ImageInterpolation
A plugin to interpolate an image by a factor a wrapper on scipy.misc.imresize. :param size: int, float or tuple. Default: 2.0. :param interp: nearest lanczos bilinear bicubic cubic. Default:‘bicubic’.

get_max_frames()

get_plugin_pattern()

process_frames(data)
This method is called after the plugin has been created by the pipeline framework and forms the main processing step

Parameters data (list(np.array)) – A list of numpy arrays for each input dataset.

setup()
This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).
savu.plugins.filters.umpa module
savu.plugins.filters.fresnel_filter module
savu.plugins.fitters
savu.plugins.fitters.base_fitter module
savu.plugins.fitters.ral_fit module
savu.plugins.fitters.reproduce_fit module
savu.plugins.fitters.simple_fit module
savu.plugins.fluo_fitters
savu.plugins.fluo_fitters.base_fluo_fitter module
savu.plugins.fluo_fitters.fastxrf_fitting module
savu.plugins.fluo_fitters.simple_fit_xrf module
savu.plugins.kinematics
savu.plugins.kinematics.stage_motion module
savu.plugins.loaders
savu.plugins.loaders.base_loader module

class BaseLoader(name='baseLoader')
A base plugin from which all data loader plugins should inherit.

U*param preview A slice list of required frames. Default: [].

*param data_file
Inline emphasis start-string without end-string.

Hidden parameter for Savu template. Default: “<>”.

~param in_datasets
~param out_datasets

data_mapping()
get_NXapp(ltype, nx_file, entry)
finds an application definition in a nexus file

get_NXdata(nx_file, detector_list)
get_experiment()

set_data_reduction_params(data_obj)
savu.plugins.loaders.random_hdf5_loader module

class RandomHdf5Loader (name='RandomHdf5Loader')
A hdf5 dataset of a specified size is created at runtime using numpy random sampling (numpy.random) and saved to file. This created dataset will be used as the input file, and the input file path passed to Savu will be ignored (use a dummy).

*param size A list specifying the required data size. Default: [].

*param axis_labels A list of the axis labels to be associated with each dimension, of the form ['name1.unit1', 'name2.unit2',...]. Default: [].

*param patterns A list of data access patterns e.g. [SINORAMG.0c.1s.2c, PROJECTION.0s.1c.2s], where 'c' and 's' represent core and slice dimensions respectively and every dimension must be specified. Default: [].

*param file_name Assign a name to the created h5 file. Default: 'input_array'.

Parameters

• dtype – A numpy array data type. Default: ‘int16’.

• dataset_name – The name assigned to the dataset. Default: ‘tomo’.

• angles – A python statement to be evaluated or a file - if the value is None, values will be in the interval [0, 180]. Default: None.

• pattern – Pattern used to create and store the hdf5 dataset - default is the first pattern in the pattern dictionary. Default: None.

• range – Set the distribution interval. Default: [1, 10].

setup()
This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).

savu.plugins.loaders.hdf5_template_loader module

savu.plugins.loaders.stitch_data_template_loader module

savu.plugins.loaders.multi_savu_loader module

savu.plugins.loaders.image_template_loader module

savu.plugins.loaders.savu_nexus_loader module

class SavuNexusLoader (name='SavuNexusLoader')
A class to load datasets, and associated metadata, from a Savu output nexus file.

By default, the last instance of each unique dataset name will be loaded. Opt instead to load a subset of these datasets, or individual datasets by populating the parameters.

*param preview A slice list of required frames to apply to ALL datasets, else a dictionary of slice lists where the key is the dataset name. Default: {}.

*param datasets Override the default by choosing specific dataset(s) to load, by stating the NX-data name. Default: [].
U*param names* Override the dataset names associated with the datasets parameter above. Default: `[]`.

**setup()**
This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).

```python
savu.plugins.loaders.full_field_loaders
savu.plugins.loaders.full_field_loaders.image_loader module
savu.plugins.loaders.full_field_loaders.fake_nxtomo_loader module
savu.plugins.loaders.full_field_loaders.multi_nxtomo_loader module

class MultiNxtomoLoader (name='MultiNxtomoLoader')
A class to load multiple scans in Nexus format into one dataset.

Parameters
  • name – The name assigned to the dataset. Default: ‘tomo’.
  • file_name – The shared part of the name of each file (not including .nxs). Default: None.
  • data_path – Path to the data inside the file. Default: ‘entry1/tomo_entry/data/data’.
  • stack_or_cat – Stack or concatenate the data (4D and 3D respectively). Default: ‘stack’.
  • stack_or_cat_dim – Dimension to stack or concatenate. Default: 3.
  • axis_label – New axis label, if required, in the form ‘name.units’. Default: ‘scan.number’.
  • range – The start and end of file numbers. Default: [0, 10].

get_dark_flat_slice_list (data_obj)
```

**setup()**
This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).

```python
savu.plugins.loaders.full_field_loaders.mrc_loader module
savu.plugins.loaders.full_field_loaders.nxtomo_loader module

class NxtomoLoader (name='NxtomoLoader')
A class to load tomography data from a hdf5 file

Parameters
  • name – The name assigned to the dataset. Default: ‘tomo’.
  • data_path – Path to the data inside the file. Default: ‘entry1/tomo_entry/data/data’.
```
• **image_key_path** – Path to the image key entry inside the nxs file. Set this parameter to “None” if use this loader for radiography. Default: ‘entry1/tomo_entry/instrument/detector/image_key’.

• **dark** – Optional path to the dark field data file, nxs path and scale value. Default: [None, None, 1].

• **flat** – Optional Path to the flat field data file, nxs path and scale value. Default: [None, None, 1].

• **angles** – A python statement to be evaluated or a file. Default: None.

• **3d_to_4d** – If this is 4D data stored in 3D then pass an integer value equivalent to the number of projections per 180-degree scan. Default: False.

• **ignore_flats** – List of batch numbers of flats (start at 1) to ignore. Default: None.

`executive_summary()`

Provide a summary to the user for the result of the plugin.

**e.g.**

- Warning, the sample may have shifted during data collection
- Filter operated normally

**Returns** A list of string summaries

`log_warning(msg)`

`setup()`

This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).

`savu.plugins.loaders.full_field_loaders.dxchange_loader module`

**class DxchangeLoader** *(name='DxchangeLoader')*

A class to load tomography data from a hdf5 file

**Parameters**

• **data_path** – Path to the data. Default: ‘exchange/data’.

• **dark** – dark data path and scale value. Default: [‘exchange/data_dark’, 1].

• **flat** – flat data path and scale value. Default: [‘exchange/data_white’, 1].

U*param logfile path to the log file. Default: None.

*param angles Inline emphasis start-string without end-string.

- Hidden. Default: [1, 2, 3].

~param image_key_path Not required Default: None.

`savu.plugins.loaders.full_field_loaders.random_3d_tomo_loader module`

**class Random3dTomoLoader** *(name='Random3dTomoLoader')*

A hdf5 dataset of a specified size is created at runtime using numpy random sampling (numpy.random), saved with relevant meta_data to a NeXus file, and used as input. It recreates the behaviour of the nxtomo loader but
with random data. The input file path passed to Savu will be ignored (use a dummy). Note: Further extensions planned to allow the generated data to be re-loaded with the nxtomo_loader.

**Parameters**

- **axis_labels** — A list of axis labels. Default: ['rotation_angle.degrees', 'detector_y.angles', 'detector_x.angles'].

  *param patterns*  
  Inline emphasis start-string without end-string.

  Patterns. Default: ['SINOGRAM.0c.1s.2c', 'PROJECTION.0s.1c.2c'].

  *param dataset_name*  
  Inline emphasis start-string without end-string.

  The name assigned to the dataset. Default: ‘tomo’.

  *param image_key*  
  Specify position of darks and flats (in that order) in the data. Default: [[0, 1], [2, 3]]

**setup()**

This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).

### savu.plugins.loaders.mapping_loaders

### savu.plugins.loaders.mapping_loaders.base_multi_modal_loader module

**class BaseMultiModalLoader**(name='BaseMultiModalLoader')

This class provides a base for all multi-modal loaders

- **add_patterns_based_on_acquisition**(data_obj, ltype)
- **get_motor_dims**(key)
- **multi_modal_setup**(ltype, data_str, name, patterns=True)
- **set_motors**(data_obj, entry, ltype)

### savu.plugins.loaders.mapping_loaders.mm_loader module

### savu.plugins.loaders.mapping_loaders.nxfluo_loader module

**class NxfluoLoader**(name='NxfluoLoader')

A class to load tomography data from an NXFluo file.

**Parameters**

- **fluo_offset** — fluo scale offset. Default: 0.0.
- **fluo_gain** — fluo gain. Default: 0.01.
- **name** — The name assigned to the dataset. Default: ‘fluo’.

**setup()**

This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).
savu.plugins.loaders.mapping_loaders.nxmonitor_loader module

class NxmonitorLoader (name='NxmonitorLoader')
A class to load tomography data from an NXmonitor file
:param name: The name assigned to the dataset.
Default: 'monitor'.

setup()
This method is first to be called after the plugin has been created. It determines input/output datasets and
plugin specific dataset information such as the pattern (e.g. sinogram/projection).

savu.plugins.loaders.mapping_loaders.nxptycho_loader module

class NxptychoLoader (name='NxptychoLoader')
A class to load tomography data from an NXstxm file
:param name: The name assigned to the dataset. Default: 'ptycho'.

setup()
Define the input nexus file

Parameters path (str) – The full path of the NeXus file to load.

savu.plugins.loaders.mapping_loaders.nxstxm_loader module

class NxstxmLoader (name='NxstxmLoader')
A class to load tomography data from an NXstxm file
:param name: The name assigned to the dataset. Default: 'stxm'.

setup()
This method is first to be called after the plugin has been created. It determines input/output datasets and
plugin specific dataset information such as the pattern (e.g. sinogram/projection).

savu.plugins.loaders.mapping_loaders.nxxrd_loader module

savu.plugins.loaders.mapping_loaders.txm_loader module

class TxmLoader (name='TxmLoader')
A class to load tomography data from the txm

setup()
Define the input nexus file

Parameters path (str) – The full path of the NeXus file to load.

savu.plugins.loaders.mapping_loaders.p2r_fly_scan_detector_loader module

class P2rFlyScanDetectorLoader (name='P2rFlyScanDetectorLoader')
A class to load p2r fly scan detector data from a Nexus file.

Parameters

• data_path – Path to the data inside the file. Default: ‘entry1/tomo_entry/data/data’.
• **image_key_path** – Path to the image key entry inside the nxs file. Default: ‘en-
try1/tomo_entry/instrument/detector/image_key’.

• **dark** – Optional path to the dark field data file, nxs path and scale value. Default: [None, 
None, 1].

• **flat** – Optional Path to the flat field data file, nxs path and scale value. Default: [None, 
None, 1].

• **angles** – A python statement to be evaluated or a file. Default: None.

• **3d_to_4d** – Set to true if this reshape is required. Default: False.

• **ignore_flats** – List of batch numbers of flats (start at 1) to ignore. Default: None.

**setup()**

This method is first to be called after the plugin has been created. It determines input/output datasets and 
plugin specific dataset information such as the pattern (e.g. sinogram/projection).

**savu.plugins.loaders.mapping_loaders.i08_loaders**

**savu.plugins.loaders.mapping_loaders.i08_loaders.i08_fluo_loader module**

class I08FluoLoader (name='I08FluoLoader')

A class to load i08s xrf data :param mono_path: The mono energy. Default: ‘/en-
try/instrument/PlaneGratingMonochromator/pgm_energy’.

**setup()**

Define the input nexus file

**Parameters path (str)** – The full path of the NeXus file to load.

**savu.plugins.loaders.mapping_loaders.i13_loaders**

**savu.plugins.loaders.mapping_loaders.i13_loaders.i13_stxm_monitor_loader module**

class I13StxmMonitorLoader (name='I13StxmMonitorLoader')

A class to load tomography data from the txm

**setup()**

Define the input nexus file

**Parameters path (str)** – The full path of the NeXus file to load.

**savu.plugins.loaders.mapping_loaders.i13_loaders.i13_fluo_loader module**

class I13FluoLoader (name='I13FluoLoader')

A class to load tomography data from an NXstxm file :param fluo_offset: fluo scale offset. Default: 0.0. :param 
fluo_gain: fluo gain. Default: 0.01. :param is_tomo: The mono energy. Default: True. :param theta_step: The 

**setup()**

Define the input nexus file
Parameters path (str) – The full path of the NeXus file to load.

savu.plugins.loaders.mapping_loaders.i13_loaders.i13_ptycho_loader module

class I13PtychoLoader (name='I13PtychoLoader')
A class to load tomography data from an NXstxm file
:param is_tomo: The mono energy. Default: True.
:param theta_start: The theta start. Default: -90.0.
:param theta_end: The theta end. Default: 90.0.

setup()

Define the input nexus file

Parameters path (str) – The full path of the NeXus file to load.

savu.plugins.loaders.mapping_loaders.i13_loaders.i13_speckle_loader module

class I13SpeckleLoader (name='I13SpeckleLoader')
A class to load tomography data from an NXstxm file
:param signal_key: Path to the signals. Default: '/entry/sample'.
:param reference_key: Path to the reference. Default: '/entry/reference'.
:param angle_key: Path to the reference. Default: '/entry/theta'.
:param dataset_names: the output sets. Default: ['signal', 'reference'].

setup()

savu.plugins.loaders.mapping_loaders.i13_loaders.i13_stxm_loader module

class I13StxmLoader (name='I13StxmLoader')
A class to load tomography data from the txm

setup()

Define the input nexus file

Parameters path (str) – The full path of the NeXus file to load.

savu.plugins.loaders.mapping_loaders.i13_loaders.i13_stxm_xrf_loader module

class I13StxmXrfLoader (name='I13StxmXrfLoader')

Parameters is_map – is it. Default: True.

setup()

Define the input nexus file

Parameters path (str) – The full path of the NeXus file to load.
class I14FluoLoader (name='I14FluoLoader')
A class to load i14s xrf data
:param mono_path: The mono energy. Default: '/entry/instrument/beamline/DCM/dcm_energy'.

setup()
Define the input nexus file

Parameters
path (str) – The full path of the NeXus file to load.

class BaseI18MultiModalLoader (name='BaseI18MultiModalLoader')
This class provides a base for all multi-modal loaders
:param fast_axis: what is the fast axis called. Default: "x".
:param scan_pattern: what was the scan. Default: ['rotation','x'].
:param x: where is x in the file. Default: 'entry1/raster_counterTimer01/traj1ContiniousX'.
:param y: where is y in the file. Default: None.
:param rotation: where is rotation in the file. Default: 'entry1/raster_counterTimer01/sc_sample_thetafine'.
:param monochromator: where is the monochromator. Default: 'entry1/instrument/DCM/energy'.

add_patterns_based_on_acquisition (data_obj, ltype)
multi_modal_setup (ltype, name)
set_motors (data_obj, ltype)

class I18FluoLoader (name='I18FluoLoader')
A class to load tomography data from an NXstxm file
:param fluo_detector: path to stxm. Default: 'entry1/xspress3/AllElementSum'.
:param name: The name assigned to the dataset. Default: 'fluo'.

setup()
This method is first to be called after the plugin has been created. It determines input/output datasets and
plugin specific dataset information such as the pattern (e.g. sinogram/projection).

class I18MonitorLoader (name='I18MonitorLoader')
A class to load tomography data from an monitor file
:param monitor_detector: path to monitor. Default: 'entry1/raster_counterTimer01/I0'.
:param name: The name assigned to the dataset. Default: 'monitor'.

setup()

This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).

savu.plugins.loaders.mapping_loaders.i18_loaders.i18_stxm_loader module

class I18StxmLoader(name='I18StxmLoader')
A class to load tomography data from an NXstxm file

setup()

Define the input nexus file

Parameters path (str) – The full path of the NeXus file to load.

savu.plugins.loaders.mapping_loaders.i22_loaders

savu.plugins.loaders.mapping_loaders.i22_loaders.i22_tomo_loader module

class I22TomoLoader(name='I22TomoLoader')
A class to load tomography data for I22 that has been previously processed using dawn

setup()

savu.plugins.loaders.templates

savu.plugins.loaders.templates.i18_templates

savu.plugins.loaders.templates.malcolm_templates

savu.plugins.loaders.templates.nexus_templates

savu.plugins.loaders.utils

savu.plugins.loaders.utils.yaml_utils module

savu.plugins.loaders.utils.mrc_header module

savu.plugins.missing_data

savu.plugins.ptychography

savu.plugins.ptychography.base_ptycho module

class BasePtycho(name)
A base plugin for doing ptychography. Other ptychography plugins should inherit from this. :param in_datasets: A list of the dataset(s) to process. Default: []. :param out_datasets: A list of the dataset(s) to process. Default: ['probe', 'object_transmission', 'positions'].

get_max_frames()
get_num_object_modes()
get_num_probe_modes()
get_output_axis_units()
get_pixel_size()
get_plugin_pattern()
    sets the pattern to work in. In this case we consider a ptycho scan to be a 4D_SCAN.
get_positions()
get_size_object()
get_size_probe()
    returns tuple

nInput_datasets()
    The number of datasets required as input to the plugin
    
    Returns Number of input datasets

nOutput_datasets()
    The number of datasets created by the plugin
    
    Returns Number of output datasets

object_pattern_setup(object_labels, object_trans)
    This is where we set up the patterns, we need to add, PROJECTIONS, SINOGRAems, TIMESERIES and SPECTRA I've created the TIMESERIES because we could in theory have a time series of spectra probe_patterns: PROJECTION, TIMESERIES (for each projection), SPECTRUM (for each energy) object_patterns: PROJECTION, SINOGRAem, SPECTRUM (for each energy) position_patterns: 1D_METADATA

probe_pattern_setup(probe_labels, probe)
    This is where we set up the patterns, we need to add, PROJECTIONS, SINOGRAems, TIMESERIES and SPECTRA I've created the TIMESERIES because we could in theory have a time series of spectra probe_patterns: PROJECTION, TIMESERIES (for each projection), SPECTRUM (for each energy) object_patterns: PROJECTION, SINOGRAem, SPECTRUM (for each energy) position_patterns: 1D_METADATA

set_object_energy_patterns(object_trans, rest_obj)
set_object_rotation_patterns(object_trans, rest_obj)
set_probe_energy_patterns(probe, rest_probe)
set_probe_rotation_patterns(probe, rest_probe)
set_projection_pattern(probe, rest_probe)
set_size_object(dataset, positions, pobj=3.3e-08)
    returns tuple

set_size_probe(val)

setup()
    This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).

setup_axis_labels(in_dataset)
    This is where we set up the axis labels the 4D scan will contain labels that are: ‘xy’, ‘detectorX’, ‘detectorY’, but the data itself may be scanned in energy or rotation or something else. We want to remove all
the above, and amend them to be the following (preferably with additional scan axes at the front): probe: ‘x’, ‘y’, ‘mode_idx’ object: ‘x’, ‘y’, ‘mode_idx’ positions: ‘xy’

class DummyPtycho
This plugin performs ptychography using the ptypy package

\begin{verbatim}
process_frames(data)
\end{verbatim}
This method is called after the plugin has been created by the pipeline framework and forms the main processing step

\begin{verbatim}
Parameters data (list(np.array)) – A list of numpy arrays for each input dataset.
\end{verbatim}

class BaseRecon (name='BaseRecon')
A base class for reconstruction plugins

\begin{verbatim}
U*param centre_of_rotation Centre of rotation to use for the reconstruction. Default: 0.0.
U*param init_vol Dataset to use as volume initialiser (doesn’t currently work with preview). Default: None.

Parameters

• centre_pad – Pad the sinogram to centre it in order to fill the reconstructed volume ROI for asthetic purposes. NB: Only available for selected algorithms and will be ignored otherwise. WARNING: This will significantly increase the size of the data and the time to compute the reconstruction). Default: False.

• outer_pad – Pad the sinogram width to fill the reconstructed volume for asthetic purposes. Choose from True (defaults to sqrt(2)), False or float <= 2.1. NB: Only available for selected algorithms and will be ignored otherwise. WARNING: This will increase the size of the data and the time to compute the reconstruction). Default: False.

• force_zero – Set any values in the reconstructed image outside of this range to zero. Default: [None, None].

U*param log Take the log of the data before reconstruction (True or False). Default: True.

U*param preview A slice list of required frames. Default: [].
\end{verbatim}
**base_pre_process()**
This method is called after the plugin has been created by the pipeline framework as a pre-processing step.

**base_process_frames_after**(data)
This method is called directly after each call to process frames and before returning the data to file.

**base_process_frames_before**(data)
Reconstruct a single sinogram with the provided centre of rotation

**br_array_pad**(ctr, nPixels)

**crop_sino**(sino, cor)
Crop the sinogram so the centre of rotation is at the centre.

**executive_summary()**
Provide a summary to the user for the result of the plugin.

- Warning, the sample may have shifted during data collection
- Filter operated normally

**Returns** A list of string summaries

**get_angles()**
Get the angles associated with the current sinogram(s).

**Returns** Angles of the current frames.

**Return type** np.ndarray

**get_centre_offset**(sino, cor, detX)

**get_centre_shift**(sino, cor)

**get_cors()**
Get the centre of rotations associated with the current sinogram(s).

**Returns** Centre of rotation values for the current frames.

**Return type** np.ndarray

**get_fov_fraction**(sino, cor)
Get the fraction of the original FOV that can be reconstructed due to offset centre

**get_frame_params()**

**get_initial_data()**
Get the initial data (if it exists) associated with the current sinogram(s).

**Returns** The section of the initialisation data associated with the current frames.

**Return type** np.ndarray or None

**get_max_frames()**

**get_pad_amount()**

**get_padding_algorithms()**
A list of algorithms that allow the data to be padded.

**get_reconstruction_alg()**

**get_sino_centre_method()**

**get_vol_shape()**
**keep_sino** *(sino, cor)*
No change to the sinogram

**map_volume_dimensions** *(data)*

**nInput_datasets** *
The number of datasets required as input to the plugin

**Returns** Number of input datasets

**nOutput_datasets** *
The number of datasets created by the plugin

**Returns** Number of output datasets

**pad_sino** *(sino, cor)*
Pad the sinogram so the centre of rotation is at the centre.

**reconstruct_pre_process** *
Should be overridden to perform pre-processing in a child class

**set_centre_of_rotation** *(inData, mData, pData)*

**set_function** *(pad_shape)*

**set_mask** *(shape)*

**setup** *
This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).

`savu.plugins.reconstructions.ccpi_cgls_recon module`

`savu.plugins.reconstructions.scikitimage_filter_back_projection module`

`savu.plugins.reconstructions.scikitimage_sart module`

`savu.plugins.reconstructions.simple_recon module`

`savu.plugins.reconstructions.visual_hulls_recon module`

`savu.plugins.reconstructions.tomobar_recon_cpu module`

`savu.plugins.reconstructions.tomobar_recon module`

`savu.plugins.reconstructions.tomobar_recon_3D module`

`savu.plugins.reconstructions.astra_recons`

`savu.plugins.reconstructions.astra_recons.astra_recon_cpu module`

`savu.plugins.reconstructions.astra_recons.astra_recon_gpu module`

`savu.plugins.reconstructions.astra_recons.base_astra_recon module`
savu.plugins.reshape
davu.plugins.reshape.data_removal module

class DataRemoval
A class to remove any unwanted data from the specified pattern frame.

:param indices: A list or range of values to remove, e.g. [0, 1, 2], 0:2 (start:stop) or 0:2:1 (start:stop:step).
Default: None.
:param dim: Data dimension to reduce. Default: 0.

calc_indices

get_max_frames

nInput_datasets
The number of datasets required as input to the plugin

    Returns  Number of input datasets

nOutput_datasets
The number of datasets created by the plugin

    Returns  Number of output datasets

pre_process
This method is called immediately after base_pre_process().

process_frames
This method is called after the plugin has been created by the pipeline framework and forms the main processing step

    Parameters  data (list(np.array)) – A list of numpy arrays for each input dataset.

setup
This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).

davu.plugins.reshape.downsample_filter module
davu.plugins.reshape.mipmap module
davu.plugins.ring_removal
davu.plugins.ring_removal.ccpi_ring_artefact_filter module
davu.plugins.ring_removal.raven_filter module
davu.plugins.ring_removal.remove_all_rings module
davu.plugins.ring_removal.remove_large_rings module
davu.plugins.ring_removal.ring_removal_normalization module

Error in “module” directive: maximum 1 argument(s) allowed, 3 supplied.
.. module:: Remove stripe artefacts

:platform: Unix
:synopsis: A plugin working in sinogram space to remove stripe artefacts

class RingRemovalNormalization

Method to remove stripe artefacts in a sinogram (-> ring artefacts in a reconstructed image) using a normalization-based method. A simple improvement to handle partial stripes is included.

Parameters

.. list_item::
   :name: radius
   :type: radius

.. list_item::
   :name: number_of_chunks
   :type: number_of_chunks
   :desc: Divide the sinogram to many chunks of rows. Default: 1

pre_process()

This method is called immediately after base_pre_process().

process_frames(data)

This method is called after the plugin has been created by the pipeline framework and forms the main processing step

Parameters data(list(np.array)) -- A list of numpy arrays for each input dataset.

setup()

This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).

savu.plugins.ring_removal.ring_removal_regularization module
savu.plugins.ring_removal.ring_removal_filtering module
savu.plugins.ring_removal.ring_removal_waveletfft module
savu.plugins.ring_removal.ring_removal_fitting module
savu.plugins.ring_removal.ring_removal_sorting module
savu.plugins.ring_removal.remove_unresponsive_and_fluctuating_rings module
savu.plugins.savers
savu.plugins.savers.base_saver module

class BaseSaver(name='BaseSaver')

A base plugin from which all data saver plugins should inherit.

*param out_datasets

Hidden, dummy out_datasets entry. Default: []

Parameters in_datasets -- The name of the dataset to save. Default: []

get_frame()
get_max_frames()
get_pattern()
The number of datasets required as input to the plugin

**Returns** Number of input datasets

The number of datasets created by the plugin

**Returns** Number of output datasets

This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).

### savu.plugins.savers.hdf5_saver module

**class Hdf5Saver** (name='Hdf5Saver')

A class to save tomography data to a hdf5 file

**Parameters**

- **pattern** – Optimise data storage to this access pattern: ‘optimum’ will automate this process by choosing the output pattern from the previous plugin, if it exists, else the first pattern. Default: ‘optimum’.

**get_pattern()**

**post_process()**

This method is called after the process function in the pipeline framework as a post-processing step. All processes will have finished performing the main processing at this stage.

**Parameters**

- **exp** (experiment class instance) – An experiment object, holding input and output datasets

**pre_process()**

This method is called immediately after base_pre_process().

**process_frames(data)**

This method is called after the plugin has been created by the pipeline framework and forms the main processing step

**Parameters**

- **data** (list(np.array)) – A list of numpy arrays for each input dataset.

### savu.plugins.savers.tiff_saver module

### savu.plugins.savers.base_image_saver module

**class BaseImageSaver** (name)

**pre_process()**

This method is called immediately after base_pre_process().

**setup()**

This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).
savu.plugins.savers.edf_saver module

savu.plugins.savers.image_saver module

savu.plugins.savers.xrf_saver module

savu.plugins.savers.utils

savu.plugins.savers.utils.hdf5_utils module

class Hdf5Utils(exp)
   A class to save tomography data to a hdf5 file

   create_dataset_nofill(group, name, shape, dtype, chunks=None)

savu.plugins.segmentation

savu.plugins.segmentation.gmm_segment3D module

savu.plugins.segmentation.geo_distance module

All the plugin architecture for Savu is contained here

savu.plugins.segmentation.mask_evolve module

savu.plugins.segmentation.mask_initialiser module

savu.plugins.segmentation.morph_snakes module

savu.plugins.segmentation.morph_snakes3D module

savu.plugins.segmentation.thresh_segm module

savu.plugins.segmentation.i23segmentation

savu.plugins.segmentation.i23segmentation.final_segment_i23 module

Error in “module” directive: maximum 1 argument(s) allowed, 14 supplied.

.. module:: creates final segmentation for i23 data, apply at the end of the process list

   :platform: Unix
   :synopsis: creates final segmentation for i23 data, apply at the end of the process list

   class FinalSegmentI23
      Apply at the end when all objects have been segmented independently (crystal, liquor, whole object)

     Parameters
     :set_classes_val
        Set the values for all 4 classes (crystal, liquor, loop, vacuum).
        Default: [255, 128, 64, 0].
nInput_datasets()
    The number of datasets required as input to the plugin
    Returns Number of input datasets

nOutput_datasets()
    The number of datasets created by the plugin
    Returns Number of output datasets

pre_process()
    This method is called immediately after base_pre_process().

process_frames(data)
    This method is called after the plugin has been created by the pipeline framework and forms the main
    processing step
    Parameters data (list(np.array)) – A list of numpy arrays for each input dataset.

setup()
    This method is first to be called after the plugin has been created. It determines input/output datasets and
    plugin specific dataset information such as the pattern (e.g. sinogram/projection).

savu.plugins.segmentation.i23segmentation.i23_segment module

savu.plugins.segmentation.i23segmentation.i23_segment3D module

savu.plugins.segmentation.morphological_operations

savu.plugins.segmentation.morphological_operations.morph_proc module

savu.plugins.visualisation

savu.plugins.visualisation.ortho_slice module

savu.plugins.stats

savu.plugins.stats.min_and_max module

class MinAndMax
    A plugin to calculate the min and max values of each slice (as determined by the pattern parameter)
    U*param pattern How to slice the data. Default: ‘VOLUME_XZ’.
    Parameters
        • smoothing – Apply a smoothing filter or not. Default: True.
        • out_datasets – The default names. Default: [‘the_min’, ‘the_max’].
    U*param masking Apply a circular mask or not. Default: True.
    U*param ratio Used to calculate the circular mask. If not provided, it is calculated using the center
        of rotation. Default: None.
    U*param method Method to find the global min and the global max. Available options: ‘extrema’,
    U*param p_range Percentage range if use the ‘percentile’ method. Default: [0.0, 100.0].
circle_mask \((width, ratio)\)

**nOutput_datasets** ()
The number of datasets created by the plugin

**Returns** Number of output datasets

**post_process** ()
This method is called after the process function in the pipeline framework as a post-processing step. All processes will have finished performing the main processing at this stage.

**Parameters**

- \(exp\) (**experiment class instance**) – An experiment object, holding input and output datasets

**pre_process** ()
This method is called immediately after base_pre_process().

**process_frames** \((data)\)
This method is called after the plugin has been created by the pipeline framework and forms the main processing step

**Parameters**

- \(data\) (**list(np.array)**) – A list of numpy arrays for each input dataset.

**setup** ()
This method is first to be called after the plugin has been created. It determines input/output datasets and plugin specific dataset information such as the pattern (e.g. sinogram/projection).
CHAPTER 3

Indices and tables

- genindex
- modindex
- search
Python Module Index

a
all_cpus_plugin (Unix), 80

b
band_pass (Unix), 82
base_analysis (Unix), 71
base_component_analysis (Unix), 76
base_correction (Unix), 76
base_driver (Unix), 80
base_image_saver (Unix), 104
base_loader (Unix), 88
base_multi_modal_loader (Unix), 96
base_pycal (Unix), 97
base_recon (Unix), 99
base_saver (Unix), 103
base_transport (Unix), 52
base_transport_data (Unix), 65
base_type (Unix), 64
basic_operations (Unix), 73
basic_transport (Unix), 52

c
checkpointing (Unix), 52
chunking (Unix), 53
cpu_plugin (Unix), 80

d
dark_flat_field_correction (Unix), 77
data (Unix), 54
data_additions (Unix), 56
data_create (Unix), 56
data_notes (Unix), 69
data_removal (Unix), 102
data_with_darks_and_flats (Unix), 63
dezinger (Unix), 83
dezinger_simple (Unix), 84
docstring_parser (Unix), 66
dosna_transport_data (Unix), 66
dxchange_loader (Unix), 91

f
filter (Unix), 82
framework_citations (Unix), 53

h
hdf5_tomo_saver (Unix), 104
hdf5_transport (Unix), 53
hdf5_transport_data (Unix), 65
hdf5_utils (Unix), 105
hilbert_filter (Unix), 87
histogram (Unix), 72

i
I08FluoLoader (Unix), 94
I13Fluo_loader (Unix), 94
I13SpeckleLoader (Unix), 95
I14FluoLoader (Unix), 96
I18stxm_loader (Unix), 97
I22TomoLoader (Unix), 97
ica (Unix), 99
ImageInterpolation (Unix), 87
iterative_plugin (Unix), 81

m
map_3dto4d_h5 (Unix), 64
meta_data (Unix), 53
min_and_max (Unix), 106
monitor_correction (Unix), 79
multi_nxtomo_loader (Unix), 90
multi_threaded_plugin (Unix), 81

n
no_process_plugin (Unix), 73
nxfluo_loader (Unix), 92
nxmonitor_loader (Unix), 93
nxstxm_loader (Unix), 93
nxtomo_loader (Unix), 90

p
p2r_fly_scan_detector_loader (Unix), 93
plugin (Unix), 66
plugin_data (Unix), 60
plugin_datasets (Unix), 68
plugin_driver (Unix), 80
poly_background_estimator (Unix), 83
preview (Unix), 62
projection_vertical_alignment (Unix), 70
q
quantisation_filter (Unix), 85
r
replicate (Unix), 65
S
savu.core.checkpointing, 23
savu.core.transport_setup, 22
savu.core.transports.base_transport, 24
savu.core.transports.basic_transport, 26
savu.core.transports.hdf5_transport, 27
savu.core.utils, 22
savu.data.chunking, 27
savu.data.data_structures.data, 29
savu.data.data_structures.data_add_ons, 32
savu.data.data_structures.data_create, 33
savu.data.data_structures.data_notes, 36
savu.data.data_structures.data_types.base_type, 46
savu.data.data_structures.data_types.data_plus_darks_and_flats, 44
savu.data.data_structures.data_types.map_3dto4d_h5, 46
savu.data.data_structures.data_types.replicate, 47
savu.data.data_structures.data_types.stitch_data, 47
savu.data.data_structures.plugin_data, 39
savu.data.data_structures.preview, 42
savu.data.data_structures.utils, 44
savu.data.framework_citations, 28
savu.data.meta_data, 29
savu.data.transport_data.base_transport_data, 48
savu.data.transport_data.basic_transport_data, 49
savu.data.transport_data.dosna_transport_data, 49
savu.data.transport_data.hdf5_transport_data, 48
savu.data.transport_data.slice_lists, 49
savu.version, 21
savu_loader (Unix), 89
sinogram_clean (Unix), 70
slice_lists (Unix), 66
spectrum_crop (Unix), 86
stats (Unix), 72
stitch_data (Unix), 65
strip_background (Unix), 86
stxm_analysis (Unix), 71
t
temp_loader (Unix), 89
threshold_filter (Unix), 87
time_based_correction (Unix), 78
timeseries_field_corrections (Unix), 79
transport_setup (Unix), 51
U
utils (Unix), 69
Symbols

_BaseTransportData__convert_str() (BaseTransportData method), 48
_BaseTransportData__get_bool_slice_dir_index() (BaseTransportData method), 48
_BaseTransportData__get_boundaries() (BaseTransportData method), 48
_BaseTransportData__get_optimum_distribution() (BaseTransportData method), 48
_BaseTransportData__refine_distribution_for_multi_mfp() (BaseTransportData method), 48
_BaseTransport__create_dataset() (BaseTransport method), 24
_BaseTransport__create_expand_function() (BaseTransport method), 24
_BaseTransport__create_squeeze_function() (BaseTransport method), 24
_BaseTransport__get_checkpoint_params() (BaseTransport method), 24
_BaseTransport__output_axis_labels() (BaseTransport method), 24
_BaseTransport__output_data() (BaseTransport method), 24
_BaseTransport__output_data_patterns() (BaseTransport method), 24
_BaseTransport__output_data_type() (BaseTransport method), 24
_BaseType__get_extras_vals() (BaseType method), 46
_BaseType__update_extra_params() (BaseType method), 46
_BasicTransport__set_hdf5_transport() (BasicTransport method), 26
_BasicTransport__unset_hdf5_transport() (BasicTransport method), 26
_Checkpointing__does_file_exist() (Checkpointing method), 23
_Checkpointing__set_checkpoint_info() (Checkpointing method), 23
_Checkpointing__set_dataset_metadata() (Checkpointing method), 23
_Checkpointing__set_start_values() (Checkpointing method), 23
_Checkpointing__write_plugin_checkpoint() (Checkpointing method), 23
_Checkpointing__write_subplugin_checkpoint() (Checkpointing method), 23
_Chunking__adjust_chunk_size() (Chunking method), 27
_Chunking__check_adjust_dims() (Chunking method), 27
_Chunking__core_core() (Chunking method), 28
_Chunking__core_other() (Chunking method), 28
_Chunking__core_slice() (Chunking method), 28
_Chunking__decrease_chunks() (Chunking method), 28
_Chunking__get_adjustable_dims() (Chunking method), 28
_Chunking__get_idx_decrease() (Chunking method), 28
_Chunking__get_idx_increase() (Chunking method), 28
_Chunking__get_idx_order() (Chunking method), 28
_Chunking__get_max_frames_dict() (Chunking method), 28
_Chunking__get_shape() (Chunking method), 28
_Chunking__get_adjustable_dims() (Chunking method), 28
_Chunking__get_idx_decrease() (Chunking method), 28
_Chunking__get_idx_increase() (Chunking method), 28
_Chunking__get_idx_order() (Chunking method), 28
_Chunking__max_frames_per_process() (Chunking method), 28
_Chunking__set_adjust_params() (Chunking method), 28
<table>
<thead>
<tr>
<th>Method Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Chunking__set_chunks() (Chunking method)</td>
<td>28</td>
</tr>
<tr>
<td>_Chunking__set_volume_bounds() (Chunking method)</td>
<td>28</td>
</tr>
<tr>
<td>_Chunking__slice_other() (Chunking method)</td>
<td>28</td>
</tr>
<tr>
<td>_Chunking__slice_slice() (Chunking method)</td>
<td>28</td>
</tr>
<tr>
<td>_DataCreate__add_extra_dims_labels() (DataCreate method)</td>
<td>33</td>
</tr>
<tr>
<td>_DataCreate__amend_axis_labels() (DataCreate method)</td>
<td>33</td>
</tr>
<tr>
<td>_DataCreate__copy_labels() (DataCreate method)</td>
<td>33</td>
</tr>
<tr>
<td>_DataCreate__copy_patterns() (DataCreate method)</td>
<td>34</td>
</tr>
<tr>
<td>_DataCreate__copy_patterns_removing_dimensions() (DataCreate method)</td>
<td>34</td>
</tr>
<tr>
<td>_DataCreate__create_axis_labels() (DataCreate method)</td>
<td>34</td>
</tr>
<tr>
<td>_DataCreate__create_dataset_from_kwargs() (DataCreate method)</td>
<td>34</td>
</tr>
<tr>
<td>_DataCreate__create_dataset_from_object() (DataCreate method)</td>
<td>34</td>
</tr>
<tr>
<td>_DataCreate__find_and_set_shape() (DataCreate method)</td>
<td>34</td>
</tr>
<tr>
<td>_DataCreate__insert_axis_labels() (DataCreate method)</td>
<td>34</td>
</tr>
<tr>
<td>_DataCreate__remove_axis_labels() (DataCreate method)</td>
<td>34</td>
</tr>
<tr>
<td>_DataCreate__replace_axis_labels() (DataCreate method)</td>
<td>34</td>
</tr>
<tr>
<td>_DataWithDarksAndFlats__get_data() (DataWithDarksAndFlats method)</td>
<td>44</td>
</tr>
<tr>
<td>_DataWithDarksAndFlats__get_preview_image() (DataWithDarksAndFlats method)</td>
<td>44</td>
</tr>
<tr>
<td>_DataWithDarksAndFlats__get_preview_index() (DataWithDarksAndFlats method)</td>
<td>44</td>
</tr>
<tr>
<td>_Data__check_dims() (Data method)</td>
<td>29</td>
</tr>
<tr>
<td>_Data__check_pattern() (Data method)</td>
<td>29</td>
</tr>
<tr>
<td>_Data_convert_pattern_dimensions() (Data method)</td>
<td>30</td>
</tr>
<tr>
<td>_Data_get_available_pattern_list() (Data method)</td>
<td>30</td>
</tr>
<tr>
<td>_Data_get_dirs_for_volume() (Data method)</td>
<td>30</td>
</tr>
<tr>
<td>_Data_get_dirs_for_volume_3D() (Data method)</td>
<td>30</td>
</tr>
<tr>
<td>_Data__initialise_data_info() (Data method)</td>
<td>30</td>
</tr>
<tr>
<td>_Data_set_main_axis() (Data method)</td>
<td>30</td>
</tr>
<tr>
<td>_Hdf5TransportData__combine_dicts() (Hdf5TransportData method)</td>
<td>49</td>
</tr>
<tr>
<td>_ImageKey__ignore_image_key_entries() (ImageKey method)</td>
<td>45</td>
</tr>
<tr>
<td>_LocalData__get_unpad_slice_list() (LocalData method)</td>
<td>49</td>
</tr>
<tr>
<td>_MPI_setup__add_console_logging() (MPI_setup method)</td>
<td>22</td>
</tr>
<tr>
<td>_MPI_setup__add_user_logging() (MPI_setup method)</td>
<td>22</td>
</tr>
<tr>
<td>_MPI_setup__mpi_setup() (MPI_setup method)</td>
<td>22</td>
</tr>
<tr>
<td>_MPI_setup__set_dictionary() (MPI_setup method)</td>
<td>22</td>
</tr>
<tr>
<td>_MPI_setup__set_logger() (MPI_setup method)</td>
<td>22</td>
</tr>
<tr>
<td>_MPI_setup__set_logger_parallel() (MPI_setup method)</td>
<td>22</td>
</tr>
<tr>
<td>_MPI_setup__set_logger_single() (MPI_setup method)</td>
<td>22</td>
</tr>
<tr>
<td>_Padding__set_dims() (Padding method)</td>
<td>32</td>
</tr>
<tr>
<td>_PluginData__check_dimensions() (PluginData method)</td>
<td>39</td>
</tr>
<tr>
<td>_PluginData__check_distribution() (PluginData method)</td>
<td>39</td>
</tr>
<tr>
<td>_PluginData__get_slice_size() (PluginData method)</td>
<td>39</td>
</tr>
<tr>
<td>_PluginData__log_max_frames() (PluginData method)</td>
<td>40</td>
</tr>
<tr>
<td>_PluginData__perform_checks() (PluginData method)</td>
<td>40</td>
</tr>
<tr>
<td>_PluginData__set_core_shape() (PluginData method)</td>
<td>40</td>
</tr>
<tr>
<td>_PluginData__set_max_frames() (PluginData method)</td>
<td>40</td>
</tr>
<tr>
<td>_PluginData__set_pattern() (PluginData method)</td>
<td>40</td>
</tr>
<tr>
<td>_PluginData__set_shape() (PluginData method)</td>
<td>40</td>
</tr>
<tr>
<td>_PluginData__set_slice_dimensions() (PluginData method)</td>
<td>40</td>
</tr>
<tr>
<td>_Preview__add_preview_param() (Preview method)</td>
<td>42</td>
</tr>
<tr>
<td>_Preview__check_preview_indices() (Preview method)</td>
<td>42</td>
</tr>
<tr>
<td>_Preview__convert_nprocs() (Preview method)</td>
<td>42</td>
</tr>
<tr>
<td>_Preview__set_reduced_shape() (Preview method)</td>
<td>42</td>
</tr>
<tr>
<td>_Replicate__set_patterns() (Replicate method)</td>
<td>47</td>
</tr>
<tr>
<td>_SliceLists__chunk_length_repeat() (SliceLists method)</td>
<td>50</td>
</tr>
</tbody>
</table>
method), 48
_get_slice_dir_matrix() (BaseTransportData method), 48
_get_slice_dirs_index() (SliceLists method), 50
_get_slice_list() (GlobalData method), 49
_get_slice_list() (LocalData method), 50
_get_slice_list_per_process() (BaseTransportData method), 48
_get_slice_lists_per_process() (BasicTransportData method), 49
_get_in_slice_list() (BasicTransportData method), 49
_get_in_slice_list() (Hdf5TransportData method), 49
_get_timer() (Checkpointing method), 23
_get_transport_data() (Data method), 30
_getitem_cat() (StitchData method), 47
_getitem_imagekey() (DataWithDarksAndFlats method), 44
_getitem_noimagekey() (DataWithDarksAndFlats method), 44
_getitem_stack() (StitchData method), 47
_group_dimension() (SliceLists method), 50
_group_slice_list_in_multiple_dimensions() (SliceLists method), 50
_group_slice_list_in_one_dimension() (SliceLists method), 50
_initialise() (BaseTransport method), 25
_initialise() (Checkpointing method), 24
_log_completion_status() (BaseTransport method), 25
_metadata_dump() (Hdf5Transport method), 27
_non_negative_directions() (Data method), 30
_output_metadata() (BaseTransport method), 25
_output_metadata_dict() (BaseTransport method), 25
_output_summary() (in module savu.core.utils), 22
_override_data_type() (BaseTransport method), 25
_pad_direction() (Padding method), 33
_pad_slice_list() (SliceLists method), 50
_padded_shape() (in module savu.data.data_structures.data_notes), 37
_populate_nexus_file() (BaseTransport method), 25
_post_clone_updates() (BaseType method), 46
_post_clone_updates() (NoImageKey method), 45
_process_loop() (BaseTransport method), 25
_remove_excess_data() (BaseTransport method), 25
_reset() (Replicate method), 47
_reset_indices() (Checkpointing method), 24
_reset_preview() (Preview method), 43
_return_all_data() (BaseTransport method), 25
_savu_decoder() (in module savu.core.utils), 22
_savu_encoder() (in module savu.core.utils), 22
_send_email() (in module savu.core.utils), 22
_set_boundaries() (BaseTransportData method), 48
_set_checkpoint_info_from_file() (Checkpointing method), 24
_set_dark_and_flat() (DataWithDarksAndFlats method), 44
_set_dark_path() (NoImageKey method), 45
_set_data_patterns() (DataCreate method), 34
_set_dictionary() (MetaData method), 29
_set_fake_key() (NoImageKey method), 45
_set_file_details() (BaseTransport method), 25
_set_flat_path() (NoImageKey method), 45
_set_functions() (BaseTransport method), 25
_set_global_frame_index() (BaseTransport method), 25
_set_in_slice_list() (StitchData method), 47
_set_meta_data() (PluginData method), 40
_set_name() (Data method), 30
_set_no_squeeze() (PluginData method), 40
_set_out_slice_list() (StitchData method), 47
_set_padded_shape() (PluginData method), 40
_set_padding_dict() (PluginData method), 40
_set_parameters() (BaseType method), 46
_set_plugin_data() (Data method), 30
_set_preview_note() (in module savu.data.data_structures.data_notes), 37
_set_previous_pattern() (Data method), 30
_set_scale() (DataWithDarksAndFlats method), 44
_set_shape() (StitchData method), 47
_set_shape_before_tuning() (PluginData method), 40
_set_shape_transfer() (PluginData method), 41
_set_timer() (Checkpointing method), 24
_set_transport_data() (Data method), 30
_setup_h5_files() (BaseTransport method), 25
_shape() (in module savu.data.data_structures.data_notes), 38
_single_slice_list() (SliceLists method), 50
_split_list() (SliceLists method), 50
_str_to_value() (BaseType method), 46
_transfer_all_data() (BaseTransport method), 25
_transport_checkpoint() (BaseTransport method), 26
_transport_checkpoint() (Hdf5Transport method), 27
_transport_cleanup() (Hdf5Transport method), 27
_transport_initialise() (BaseTransport method), 26

116  Index
<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>_transport_initialise()</td>
<td>BasicTransport</td>
<td>savu.core.transports.basic_transport</td>
</tr>
<tr>
<td>_transport_initialise()</td>
<td>Hdf5Transport</td>
<td>savu.core.transports.hdf5_transport</td>
</tr>
<tr>
<td>_transport_kill_signal()</td>
<td>BaseTransport</td>
<td>savu.core.transports.base_transport</td>
</tr>
<tr>
<td>_transport_kill_signal()</td>
<td>Hdf5Transport</td>
<td>savu.core.transports.hdf5_transport</td>
</tr>
<tr>
<td>_transport_load_plugin()</td>
<td>BaseTransport</td>
<td>savu.core.transports.base_transport</td>
</tr>
<tr>
<td>_transport_post_plugin()</td>
<td>BasicTransport</td>
<td>savu.core.transports.basic_transport</td>
</tr>
<tr>
<td>_transport_post_plugin()</td>
<td>Hdf5Transport</td>
<td>savu.core.transports.hdf5_transport</td>
</tr>
<tr>
<td>_transport_post_plugin() list_run()</td>
<td>BaseTransport</td>
<td>savu.core.transports.base_transport</td>
</tr>
<tr>
<td>_transport_post_plugin()</td>
<td>BasicTransport</td>
<td>savu.core.transports.basic_transport</td>
</tr>
<tr>
<td>_transport_post_plugin()</td>
<td>Hdf5Transport</td>
<td>savu.core.transports.hdf5_transport</td>
</tr>
<tr>
<td>_transport_post_plugin() list_run()</td>
<td>BaseTransport</td>
<td>savu.core.transports.base_transport</td>
</tr>
<tr>
<td>_transport_pre_plugin()</td>
<td>BasicTransport</td>
<td>savu.core.transports.basic_transport</td>
</tr>
<tr>
<td>_transport_pre_plugin()</td>
<td>Hdf5Transport</td>
<td>savu.core.transports.hdf5_transport</td>
</tr>
<tr>
<td>_transport_pre_plugin() list_run()</td>
<td>BaseTransport</td>
<td>savu.core.transports.base_transport</td>
</tr>
<tr>
<td>_transport_pre_plugin()</td>
<td>BasicTransport</td>
<td>savu.core.transports.basic_transport</td>
</tr>
<tr>
<td>_transport_pre_plugin()</td>
<td>Hdf5Transport</td>
<td>savu.core.transports.hdf5_transport</td>
</tr>
<tr>
<td>_transport_pre_plugin() list_run()</td>
<td>BaseTransport</td>
<td>savu.core.transports.base_transport</td>
</tr>
<tr>
<td>_transport_process()</td>
<td>BaseTransport</td>
<td>savu.core.transports.base_transport</td>
</tr>
<tr>
<td>_transport_terminate_dataset()</td>
<td>BaseTransport</td>
<td>savu.core.transports.base_transport</td>
</tr>
<tr>
<td>_transport_terminate_dataset()</td>
<td>Hdf5Transport</td>
<td>savu.core.transports.hdf5_transport</td>
</tr>
<tr>
<td>_transport_update_plugin_list_run()</td>
<td>BaseTransport</td>
<td>savu.core.transports.base_transport</td>
</tr>
<tr>
<td>_transport_update_plugin_list_run()</td>
<td>Hdf5Transport</td>
<td>savu.core.transports.hdf5_transport</td>
</tr>
<tr>
<td>_unset_preview()</td>
<td>Preview</td>
<td>savu.core.utils</td>
</tr>
<tr>
<td>add_base()</td>
<td>(in module savu.core.utils)</td>
<td>22</td>
</tr>
<tr>
<td>add_base_class_with_instance()</td>
<td>BaseType</td>
<td>46</td>
</tr>
<tr>
<td>add_base_classes()</td>
<td>(in module savu.core.utils)</td>
<td>22</td>
</tr>
<tr>
<td>add_pattern()</td>
<td>(Data method)</td>
<td>30</td>
</tr>
<tr>
<td>add_syslog_log_handler()</td>
<td>(in module savu.core.utils)</td>
<td>22</td>
</tr>
</tbody>
</table>

**Index**

A

- add_base() (in module savu.core.utils), 22
- add_base_class_with_instance() (BaseType method), 46
- add_base_classes() (in module savu.core.utils), 22
- add_pattern() (Data method), 30
- add_syslog_log_handler() (in module savu.core.utils), 22

B

- add_user_log_handler() (in module savu.core.utils), 22
- add_user_log_level() (in module savu.core.utils), 22
- all_cpus_plugin (module), 80
- amend_axis_label_values() (Data method), 31
- axis_labels() (in module savu.data.data_structures.data_notes), 38
- band_pass (module), 82
- base_analysis (module), 71
- base_component_analysis (module), 76
- base_correction (module), 76
- base_data_args() (BaseType method), 46
- base_driver (module), 80
- base_image_saver (module), 104
- base_loader (module), 88
- base_multi_modal_loader (module), 92, 96
- base_ptycho (module), 97
- base_recon (module), 99
- base_saver (module), 103
- base_transport (module), 24, 52
- base_transport_data (module), 48, 65
- base_type (module), 46, 64
- BaseTransport (class in savu.core.transports.base_transport), 24
- BaseTransportData (class in savu.data.transport_data.base_transport_data), 48
- BaseType (class in savu.data.data_structures.data_types.base_type), 46
- basic_operations (module), 73
- basic_transport (module), 26, 52

C

- call_mpi__barrier() (MPI_setup method), 22
- check_is_map() (DataMapping_setup method), 32
- check_is_tomo() (DataMapping_setup method), 32
- Checkpointing (class in savu.core.checkpointing), 23
- chunking (class in savu.data.chunking), 27
- chunking (module), 27, 53
- clone_data_args() (BaseType method), 46
- clone_data_args() (ImageKey method), 45
- clone_data_args() (Map3Dto4Dh5 method), 46
- clone_data_args() (NoImageKey method), 45
- clone_data_args() (Replicate method), 47
clone_data_args() (StitchData method), 47
convert_indices() (Preview method), 43
cpu_plugin (module), 80
create_dataset() (DataCreate method), 34
create_next_instance() (BaseType method), 46

d
dark() (ImageKey method), 45
dark() (NoImageKey method), 45
dark_flat_field_correction (module), 77
dark_image_key_data() (DataWithDarksAndFlats method), 44
dark_mean() (DataWithDarksAndFlats method), 44
dark_mean() (DataWithDarksAndFlats method), 44
Data (class in savu.data.data_structures.data), 29
data (module), 29, 54
data_additions (module), 32, 56
data_create (module), 33, 56
data_notes (module), 36, 59, 69
data_removal (module), 102
data_with_darks_and_flats (module), 44, 63
DataCreate (class in savu.data.data_structures.data_create), 33
DataMapping (class in savu.data.data_structures.data_add_ons), 32
DataWithDarksAndFlats (class in savu.data.data_structures.data_types.data_plus_darks_and_flats), 44

delete() (MetaData method), 29
definger (module), 83
definger_simple (module), 84
docstring_parameter() (in module savu.core.utils), 22
docstring_parser (module), 66
dosna_transport_data (module), 49, 66
DosnaTransportData (class in savu.data.transport_data.dosna_transport_data), 49
dxchange_loader (module), 91

F
filter (module), 82
flat() (ImageKey method), 45
flat() (NoImageKey method), 45
flat_image_key_data() (DataWithDarksAndFlats method), 44
flat_mean() (DataWithDarksAndFlats method), 44
flat_mean() (StitchData method), 47
framework_citations (module), 28, 53

G
get() (MetaData method), 29
get_available_gpus() (in module savu.core.utils), 23
get_available_pattern_types() (in module savu.data.data_structures.utils), 44
get_axes() (DataMapping method), 32
get_axis_label_keys() (Data method), 31
get_axis_labels() (Data method), 31
get_checkpoint_plugin() (Checkpointing method), 24
get_clone_args() (BaseType method), 46
get_core_dimensions() (Data method), 31
get_core_dimensions() (PluginData method), 41
get_current_frame_idx() (PluginData method), 41
get_dark_flat_slice_list() (DataWithDarksAndFlats method), 44
get_data_dimension_by_axis_label() (Data method), 31
get_data_dimension_by_axis_label() (PluginData method), 41
get_data_obj() (Preview method), 43
get_data_patterns() (Data method), 31
get_dictionary() (MetaData method), 29
get_dtype() (DataCreate method), 36
get_frame_limit() (PluginData method), 41
get Framework_citation() (Data framework_citations), 28
get_image_key() (DataWithDarksAndFlats method), 45
get_index() (DataWithDarksAndFlats method), 45
get_itemsize() (Data method), 31
g’s() (Checkpointing method), 24
get_map_args() (BaseType method), 46
get_memory_usage_linux() (in module savu.core.utils), 23
get_motor_type() (DataMapping method), 32
get_motors() (DataMapping method), 32
get_name() (Data method), 31
get_padded_shape() (PluginData method), 41
get pattern() (PluginData method), 41
get_pattern_name() (PluginData method), 41
get_pattern_rank() (in module savu.data.data_structures.utils), 44
get_preview() (Data method), 31
get_previous_pattern() (Data method), 32
get Proc_idx() (Checkpointing method), 24
get_shape() (BaseType method), 46
get_shape() (Data method), 32
get_shape() (DataWithDarksAndFlats method), 45
get_shape() (Map3dto4dhS method), 46
get_shape() (NoImageKey method), 45
get_shape() (PluginData method), 41
get_shape() (Replicate method), 47
get_shape() (StitchData method), 47
get_shape_transfer() (PluginData method), 41
get_slice_dimensions() (PluginData method), 41
get_slice_dimensions() (Data method), 32
get_start_values() (Checkpointing method), 47
get_starts_stops_steps() (Preview method), 43
get_total_frames() (PluginData method), 41
get_trans_idx() (Checkpointing method), 24
GlobalData (class in savu.data.transport_data.slice_lists), 49
hdf5_tomo_saver (module), 104
hdf5_transport (module), 27, 53
hdf5_transport_data (module), 48, 49, 65, 66
Hdf5Transport (class in savu.core.transports.hdf5_transport), 27
Hdf5TransportData (class in savu.data.transport_data.hdf5_transport_data), 48
hilbert_filter (module), 87
histogram (module), 72
I08FluoLoader (module), 94
i13fluo_loader (module), 94
I13SpeckleLoader (module), 95
I14FluoLoader (module), 96
I18stxm_loader (module), 93–95
I22TomoLoader (module), 97
ica (module), 99
image_key() (in module savu.data.data_structures.data_notes), 39
ImageInterpolation (module), 87
ImageKey (class in savu.data.data_structures.data_types.data_plus_darks_and_flats), 45
import_class() (in module savu.core.utils), 23
is_time_to_checkpoint() (Checkpointing method), 24
iterative_plugin (module), 81
LocalData (class in savu.data.transport_data.slice_lists), 49
logfunction() (in module savu.core.utils), 23
logmethod() (in module savu.core.utils), 23
Map3dto4dh5 (class in savu.data.data_structures.data_types.map_3dto4d_h5), 46
map_3dto4d_h5 (module), 46, 64
map_input_args() (BaseType method), 46
map_input_args() (ImageKey method), 45
meta_data (module), 29, 53
MetaData (class in savu.data.meta_data), 29
min_and_max (module), 106
monitor_correction (module), 77, 79
MPI_setup (class in savu.core.transports_setup), 22
multi_nxtomo_loader (module), 90
multi_threaded_plugin (module), 81
N
no_process_plugin (module), 73
NoImageKey (class in savu.data.data_structures.data_types.data_plus_darks_and_flats), 45
nxfluo_loader (module), 92
nxmonitor_loader (module), 93
nxstxm_loader (module), 93–95
nxtomo_loader (module), 90
O
output_plugin_checkpoint() (Checkpointing method), 24
P
p2r_fly_scan_detector_loader (module), 93
pad_directions() (Padding method), 33
pad_frame_edges() (Padding method), 33
pad_mode() (Padding method), 33
pad_multi_frames() (Padding method), 33
Padding (class in savu.data.data_structures.data_add_ons), 32
patterns() (in module savu.data.data_structures.data_notes), 39
plugin (module), 66
plugin_data (module), 39, 60
plugin_data_setup() (PluginData method), 41
plugin_data_transfer_setup() (PluginData method), 42
plugin_datasets (module), 68
plugin_driver (module), 80, 81
PluginData (class in savu.data.data_structures.plugin_data), 39
poly_background_estimator (module), 83
Preview (class in savu.data.data_structures.preview), 42
preview (module), 42, 62
process_setup() (BaseTransport method), 26
projection_vertical_alignment (module), 70
Q
quantisation_filter (module), 85

Index 119