McSAS Documentation

Release 1.3.1

Brian R. Pauw

Jun 26, 2018

Contents

CHAPTER 1

McSAS Quick Usage guide

1.1 Introduction

This guide is intended as an aid to getting the first fits using McSAS.

For comprehensive details of what goes on under the hood, please refer as a baseline to the available publications. Additionally, the code is open source, and provides the best "documentation" of what actually takes place.

When publishing results using this code, the user is requested to cite either or both of the following works:

Bressler, I, Pauw, B. R, and Thuenemann, A., submitted to J. Appl. Cryst., arXiv:1412.1900

Pauw, B. R., Pedersen, J. S., Tardif, S., Takata, M. and Iversen, B. B.,

10. Appl. Cryst. 46 (2013), 365—371.

1.1.1 Scope of the code capabilities

The McSAS code at the moment can:

1. Fit supplied data to a variety of models, with absolute unit support.
2. Graphically show the distributions of selected parameters and associated parameter ranges.
3. Graphical output includes distribution population modes with uncertainties.
4. Output the fit, data, settings, and distributions for further processing.
5. Can be used with or without user interface, using command-line arguments.

1.2 -1. Starting McSAS

McSAS can be used on both Linux / Unix systems (including MacOS X) as well as personal computers running Windows.

On Unix- and Unix-like computers, McSAS can be started from any terminal by typing: $ /path/to/mcsas/main.py

On Windows, the compiled executable can be double-clicked to start.

1.3 0. Loading test data

A demonstration dataset has been simulated using SASFit, and output in a three-column file: "test-data/quickstartdemo1.csv".

The dataset can be loaded by right-clicking in the empty list in the "Data Files"-tab. When loaded, the list shows some information of the data: its length, content, q-limits, and scatterer radius limits calculated using the Q limits (theoretically, the spacing between Q-points also dictates the size limits, but the tendency of users to use much too narrowly-spaced Q-points would result in unworkable estimates).

Lastly, the "log"-window shows how many datapoints had an uncertainty estimate below 1% of the intensity. These datapoint uncertainty values have been adjusted to 1%, as that is a practically demonstrated limit of SAS data accuracy.

1.4 1. Configuring the algorithm

The algorithm uses several internal parameters. Some of these can be adjusted in the "Algorithm"-tab of the user interface. They are:

1. The convergence criterion. If the uncertainty estimates provided with the data are not accurate, or if the fitting model chosen is unsuitable or incompletely descriptive, the algorithm may not arrive at a final solution (a convergence criterion of 1 or below).
2. Number of repetitions from which uncertainties are determined. Set to 10 for the quick start.
3. Number of contributions. This can be adjusted to minimise the optimization time. The average optimisation time can be found in the "Timing" line on the graphical output.
4. Background (checkbox). When set, it will add a flat background contribution to the fit

If necessary, the other internal algorithm parameters can be changed through careful editing of the "mcsasparameters.json" parameter dictionary. This should not be needed for common use.

1.5 2. Configuring the Model

For the quick start, the model selected should be the "Sphere" model. After selecting this model, verify that the sphere radius is "Active", then move back to the "Data files"-tab, and double-click the "quickstartdemo1"-line to copy the sphere size estimates to the model.

1.6 3. Configuring the Post-fit Analysis

Select the already filled lines in the "Post-fit Analysis"-tab, and select "remove" from the right-click menu. Then add a new entry to the list by selecting "add range" from the right-click menu. In the emerging window, change the histogram X-axis scaling from "lin" to "log" and click "add".

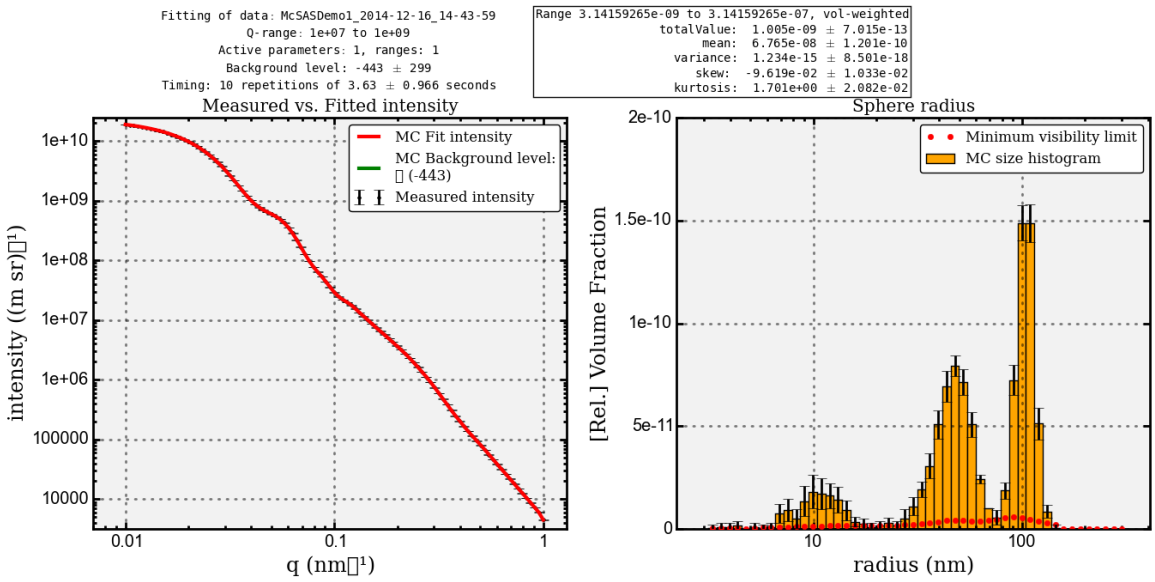
The settings are now complete.

1.7 4. Running the fit

Click the "start"-button. Optimisation takes 36 seconds on a 3.4 GHz intel i7 iMac (2012 model).

1.8 5. The result

The result window should pop up automatically and resemble the plot shown in *quickstartdemo1.pdf*



The left-hand plot shows the data in black with error bars, the MC fit in red, and a green line indicating the fitted background level (not shown as it approaches zero). The background value is furthermore indicated in the legend.

The right-hand plot shows the resulting volume-weighted size histogram, with uncertainties on the bars, and the red dashed line indicating the minimum level required for each bin to contribute a measurable amount to the scattering pattern (i.e. more than the uncertainty).

As is clear from the vertical axis on that plot, the partial volume fractions are unrealistic if the scattering contrast has not been set. One may also want to rebin the plot in fewer bins to reduce the relative uncertainties on the bins.

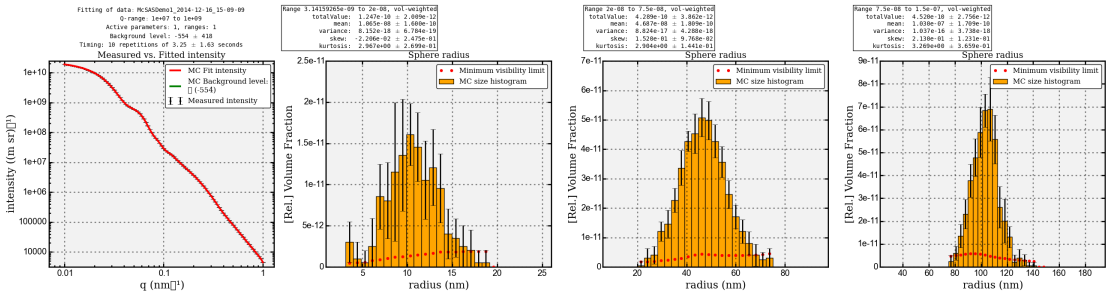
1.9 5. Getting peak parameters

Above the size distribution, the population statistics can be found suhc as the mean, the variance, skew and kurtosis (each with their own uncertainty estimates). These are valid for the entire range, and are calculated from the individual contributions (not from the histogram). If we want to find the values for the individual populations, we need to set the correct ranges in the Post-fit Analysis list:

1. 3.14 - 20, (binning e.g. 20 bins, linearly spaced, volume-weighted)
2. 20 - 75, (binning for example as above)
3. 75 - 150, (ibid.)

And then pressing "start" again. (This restarts the fitting procedure. A "Re-analyze"-option is in the works but was not finished at the time of writing)

This should give the following figure (as shown in *quickstartdemo1a.pdf*):



The mean of each population is slightly shifted upward as compared to the simulation parameters (given in paragraph 8). Note, however, that the simulation parameters define the *numeric* mean, whereas the ones shown in the histogram are the *volumetric* mean. Therefore, the latter are shifted up.

1.10 6. Further results

In the directory that the "quickstartdemo1.csv"-file was located, you will find several more files after a successful fit. These are the automatically saved results. They all contain a timestamp on when the fit was performed. Files with an asterisk are only present if the fit was successful. They are:

1. quickstartdemo1_[timestamp].pdf*: the autosaved plot
2. quickstartdemo1_[timestamp]_log.txt: the fitting log window.
3. quickstartdemo1_[timestamp]_contributions.pickle*: A pickled list of raw contributions.
4. quickstartdemo1_[timestamp]_hist[parameter].csv*: The histogram information of every range.
5. quickstartdemo1_[timestamp]_stats[parameter].csv*: The statistics information of every range
6. quickstartdemo1_[timestamp]_settings.cfg: The settings used in the fit.

1.11 7. What's next?

If you have the ability and interest in improving the code, please consider joining the development effort, which will work on including more shapes and adding slit-smearing options.

If you have more questions that are not answered in either 1) the paper, 2) the code, and 3) this document, feel free to send me an e-mail which you can find on the papers or http://mcsas.net/.

Good luck!

1.12 8. SASFit test data settings:

SASfit version 0.94.5, precompiled Mac OS X version. quickstartdemo1.csv:

1. Spheres, gaussian dist, N = 0.00105, S = 10, X0 = 100, eta = 1.
2. Spheres, gaussian dist, N = 0.0135, S = 10, X0 = 40, eta = 1.
3. Spheres, gaussian dist, N = 0.4, S = 3, X0 = 8, eta = 1.

CHAPTER 2

The Math Behind

2.1 SAXS

In *models.sphere* the form factor is defined as:

$$ff_{sph}(q,r) = \frac{3\ sin(qr) - qr\ cos(qr)}{(qr)^3}$$

$$v_{sph}(r) = \frac{4}{3}\ \pi\ r^3$$

$$v_{sph,abs}(r,\Delta\rho) = \Delta\rho^2\ v_{sph}(r)$$

Where *q* is the scattering vector loaded from the data file and possibly preprocessed, respectively filtered by defining min/max q or masking invalid values equal or below zero.

***r* denotes** the radius of the sphere set in the user interface (UI) or varied during optimization.

$\Delta\rho$ **denotes** the scattering length density difference constant of the model against the solution which is defined in the UI.

`Sphere.`**`formfactor`**`(`*dataset*`)`
Calculates the form factor of a sphere defined by:

$F(q,r) = \frac{3\ sin(qr) - qr \cdot cos(qr)}{(qr)^3}$

`Sphere.`**`volume`**`()`
Calculates the volume of a sphere defined by:

$v(r) = \frac{4\pi}{3} r^3$

`Sphere.`**`absVolume`**`()`
Calculates the volume of a sphere taking the scattering length density difference $\Delta\rho$ into account:

$v_{abs}(r,\Delta\rho) = v_{sph}(r) \cdot \Delta\rho^2$

`SASModel.`**`weight`**`()`
Calculates an intensity weighting used during fitting. It is based on the scatterers volume. It can be modified by a user-defined compensation exponent c. The default value is $c = \frac{2}{3}$

$w(r) = v(r)^{2c}$

SASModel.**calcIntensity**(*data*, *compensationExponent=None*)

Returns the intensity *I*, the volume v_{abs} and the intensity weights *w* for a single parameter contribution over all *q*:

$I(q, r) = F^2(q, r) \cdot w(r)$

CHAPTER 3

Source Code Documentation

Contents:

3.1 mcsas.mcsas package

3.1.1 Submodules

mcsas.mcsas.backgroundscalingfit module

class BackgroundScalingFit (*findBackground*, *positiveBackground*, **args*)
Bases: object

Chi-squared convergence calculation happens here. Optimizes the scaling and background factor to match *intCalc* closest to intObs. Returns an array with scaling factors. Input initial guess *sc* has to be a two-element array with the scaling and background.

Input arguments:

Parameters

- **intObs** – An array of *measured*, respectively *observed* intensities
- **intCalc** – An array of intensities which should be scaled to match *intObs*
- **intError** – An array of uncertainties to match *intObs*
- **sc** – A 2-element array of initial guesses for scaling factor and background
- **ver** – *(optional)* Can be set to 1 for old version, more robust but slow, default 2 for new version, 10x faster than version 1, requires decent starting values
- **outputIntensity** – *(optional)* Return the scaled intensity as third output argument, default: False
- **background** – *(optional)* Enables a flat background contribution, default: True

- **positiveBackground** – *(optional)* Fixes the background values to positive only, default: False

Returns (*sc*, *conval*): A tuple of an array containing the intensity scaling factor and background and the reduced chi-squared value.

static aGoFsAlpha(*dataMeas*, *dataErr*, *dataCalc*)
The alternative Goodness-of-Fit value without alpha, i.e. multiplied by alpha, according to [Henn 2016] (http://dx.doi.org/10.1107/S2053273316013206).

calc(*data*, *modelData*, *sc*, *ver=2*)

> **Warning:** method 'mcsas.backgroundscalingfit.BackgroundScalingFit.calc' undocumented

static chi(*sc*, *dataMeas*, *dataErr*, *dataCalc*)
Chi calculation, difference of measured and calculated signal.

static chiNoBg(*sc*, *dataMeas*, *dataErr*, *dataCalc*)
Chi calculation, difference of measured and calculated signal, scaling only, no backgrund.

static chiPosBg(*sc*, *dataMeas*, *dataErr*, *dataCalc*)
Chi calculation, difference of measured and calculated signal.

static chiSqr(*dataMeas*, *dataErr*, *dataCalc*)
Reduced Chi-squared calculation, size of parameter-space not taken into account; for data with known intError.

dataScaled(*data*, *sc*)
Returns the input data scaled by the provided factor and background level applied if requested.

fitLM(*dataMeas*, *dataErr*, *dataCalc*, *sc*)

> **Warning:** method 'mcsas.backgroundscalingfit.BackgroundScalingFit.fitLM' undocumented

fitSimplex(*dataMeas*, *dataErr*, *dataCalc*, *sc*)

> **Warning:** method 'mcsas.backgroundscalingfit.BackgroundScalingFit.fitSimplex' undocumented

signBackground(*b*)
Fixes the background to positive values if bool is set

mcsas.mcsas.mcsas module

class McSAS
Bases: *bases.algorithm.algorithmbase.AlgorithmBase*

Main class containing all functions required to do Monte Carlo fitting.

Required:

- ***data*: The dataset to fit.** Has to be an instance of :py:class:SASData

- ***model*: The scattering model object to assume.** It has to be an instance of `ScatteringModel`.

For more settings, see mcsas/mcsasparameters.json

Returns:

A McSAS object with the following Results stored in the *result* member attribute. These can be extracted using McSAS.result[<parameterIndexNumber>]['<Keyword>'] where the *parameterIndexNumber* indicates which shape parameter information is requested. E.g. an ellipsoid has 3: width, height and orientation. (Some information is only stored in *parameterIndexNumber = 0* (default)).

Keyword may be one of the following:

***fitMeasValMean*: 1D array (*common result*)** The fitted measVal, given as the mean of all numReps Results.

***fitX0*: 1D array (*common result*)** Corresponding q values (may be different than the input q if *X0Bounds* was used).

***fitMeasValStd*: array (*common result*)** Standard deviation of the fitted I(q), calculated as the standard deviation of all numReps results.

***contribs*: size array (numContribs x numReps) (*common result*)** Collection of numContribs contributions fitted to best represent the provided I(q) data. Contains the Results of each of *numReps* iterations. This can be used for rebinning without having to re-optimize.

***scalingFactors*: size array (2 x numReps) (*common result*)** Scaling and background values for each repetition. Used to display background level in data and fit plot.

***histogramXLowerEdge*: array** histogram bin left edge position (x-axis in histogram).

***histogramXMean*: array** Center positions for the size histogram bins (x-axis in histogram, used for errorbars).

***histogramXWidth*: array** histogram bin width (x-axis in histogram, defines bar plot bar widths).

***volumeHistogramYMean*: array** Volume-weighted particle size distribution values for all numReps Results (y-axis bar height).

***numberHistogramYMean*: array** Number-weighted analogue of the above *volumeHistogramYMean*.

***volumeHistogramRepetitionsY*: size array (self.histogramBins x numReps)** Volume-weighted particle size distribution bin values for each fit repetition (the mean of which is *volumeHistogramYMean*, and the sample standard deviation is *volumeHistogramYStd*).

***numberHistogramRepetitionsY*: size array (self.histogramBins x numReps)** Number-weighted particle size distribution bin values for each MC fit repetition.

***volumeHistogramYStd*: array** Standard deviations of the corresponding volume-weighted size distribution bins, calculated from *numReps* repetitions of the model fitting function.

***numberHistogramYStd*: array** Standard deviation for the number-weigthed distribution.

***volumeFraction*: size array (numContribs x numReps)** Volume fractions for each of numContribs contributions in each of *numReps* iterations.

***numberFraction*: size array (numContribs x numReps)** Number fraction for each contribution.

***totalVolumeFraction*: size array (numReps)** Total scatterer volume fraction for each of the *numReps* iterations.

***totalNumberFraction*: size array (numReps)** Total number fraction.

***minimumRequiredVolume*: size array (numContribs x numReps)** Minimum required volume fraction for each contribution to become statistically significant.

***minimumRequiredNumber*: size array (numContribs x numReps)** Number-weighted analogue to *minimumRequiredVolume*.

***volumeHistogramMinimumRequired*: size array (histogramXMean)** Array with the minimum required volume fraction per bin to become statistically significant. Used to display minimum required level in histogram.

***numberHistogramMinimumRequired*: size array (histogramXMean)** Number-weighted analogue to *volumeHistogramMinimumRequired*.

***scalingFactors*: size array (2 x numReps)** Scaling and background values for each repetition. Used to display background level in data and fit plot.

***totalVolumeFraction*: size array (numReps)** Total scatterer volume fraction for each of the *numReps* iterations.

***minimumRequiredVolume*: size array (numContribs x numReps)** Minimum required volube fraction for each contribution to become statistically significant.

***volumeHistogramMinimumRequired*: size array (histogramXMean)** Array with the minimum required volume fraction per bin to become statistically significant. Used to display minimum required level in histogram.

Creates instances from defined parameters and replaces the class attributes accordingly.

analyse()

This function runs the Monte Carlo optimisation a multitude (*numReps*) of times. If convergence is not achieved, it will try again for a maximum of *maxRetries* attempts.

calc(***kwargs*)

> **Warning:** method 'mcsas.mcsas.McSAS.calc' undocumented

data = None

classmethod factory()

> **Warning:** method 'mcsas.mcsas.McSAS.factory' undocumented

gen2DMeasVal()

This function is optionally run after the histogram procedure for anisotropic images, and will calculate the MC fit measVal in image form

histogram(*contribs=None*)

Takes the *contribs* result from the *McSAS.analyse()* function and calculates the corresponding volume- and number fractions for each contribution as well as the minimum observability limits. It will subsequently bin the Result across the range for histogramming purposes.

While the volume-weighted distribution will be in absolute units (providing volume fractions of material within a given size range), the number distributions have been normalized to 1.

Output a list of dictionaries with one dictionary per shape parameter:

***histogramXLowerEdge*: array** histogram bin left edge position (x-axis in histogram)

***histogramXMean*: array** Center positions for the size histogram bins (x-axis in histogram, used for errorbars)

***histogramXWidth*: array** histogram bin width (x-axis in histogram, defines bar plot bar widths)

***volumeHistogramYMean*: array** Volume-weighted particle size distribution values for all *numReps* Results (y-axis bar height)

***numberHistogramYMean*: array** Number-weighted analogue of the above *volumeHistogramYMean*

***volumeHistogramRepetitionsY*: size (histogramBins x numReps)** array Volume-weighted particle size distribution bin values for each MC fit repetition (whose mean is *volumeHistogramYMean*, and whose sample standard deviation is *volumeHistogramYStd*)

***numberHistogramRepetitionsY*: size (histogramBins x numReps)** array Number-weighted particle size distribution bin values for each MC fit repetition

***volumeHistogramYStd*: array** Standard deviations of the corresponding volume-weighted size distribution bins, calculated from *numReps* repetitions of the model fitting function

***numberHistogramYStd*: array** Standard deviation for the number-weigthed distribution

***volumeFraction*: size (numContribs x numReps) array** Volume fractions for each of numContribs contributions in each of numReps iterations

***numberFraction*: size (numContribs x numReps) array** Number fraction for each contribution

***totalVolumeFraction*: size (numReps) array** Total scatterer volume fraction for each of the *numReps* iterations

***totalNumberFraction*: size (numReps) array** Total number fraction

***minimumRequiredVolume*: size (numContribs x numReps) array** minimum required volume fraction for each contribution to become statistically significant.

***minimumRequiredNumber*: size (numContribs x numReps) array** number-weighted analogue to *minimumRequiredVolume*

***volumeHistogramMinimumRequired*: size (histogramXMean) array** array with the minimum required volume fraction per bin to become statistically significant. Used to display minimum required level in histogram.

***numberHistogramMinimumRequired*: size (histogramXMean) array** number-weighted analogue to *volumeHistogramMinimumRequired*

***scalingFactors*: size (2 x numReps) array** Scaling and background values for each repetition. Used to display background level in data and fit plot.

mcFit (*numContribs*, *outputMeasVal=False*, *outputDetails=False*, *nRun=None*)
Object-oriented, shape-flexible core of the Monte Carlo procedure. Takes optional arguments:

***outputMeasVal*:** Returns the fitted measVal besides the Result

***outputDetails*:** details of the fitting procedure, number of iterations and so on

***nRun*: "serial number" of run. Used to store results in parameters** in the right place

model = None

plot (*axisMargin=0.3*, *outputFilename=None*, *autoClose=False*)
Expects outputFilename to be of type gui.calc.OutputFilename.

result = None

array(*value*)

> **Warning:** function 'mcsas.mcsas.array' undocumented

mcsas.mcsas.mcsasdefaultcfg module

default for settings and info used for a McSAS run used by McSASCfg

class ExtendedEncoder(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: `json.encoder.JSONEncoder`

JSON encoder extended to deal with Unicode, arrays and cls descriptions

Constructor for JSONEncoder, with sensible defaults.

If skipkeys is false, then it is a TypeError to attempt encoding of keys that are not str, int, float or None. If skipkeys is True, such items are simply skipped.

If ensure_ascii is true, the output is guaranteed to be str objects with all incoming non-ASCII characters escaped. If ensure_ascii is false, the output can contain non-ASCII characters.

If check_circular is true, then lists, dicts, and custom encoded objects will be checked for circular references during encoding to prevent an infinite recursion (which would cause an OverflowError). Otherwise, no such check takes place.

If allow_nan is true, then NaN, Infinity, and -Infinity will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise, it will be a ValueError to encode such floats.

If sort_keys is true, then the output of dictionaries will be sorted by key; this is useful for regression tests to ensure that JSON serializations can be compared on a day-to-day basis.

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is `None` and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

If specified, default is a function that gets called for objects that can't otherwise be serialized. It should return a JSON encodable version of the object or raise a `TypeError`.

default(*obj*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

Parameter(**args*, ***kwargs*)

> **Warning:** function 'mcsas.mcsasdefaultcfg.Parameter' undocumented

class cInfo(***kwargs*)

Bases: `object`

This class contains all the information required to read, verify and write configuration parameters files.

initialise the defaults and populate the database with values where appropriate default parameter file can be provided using kwarg: paramDefFile = 'path/to/file' mcsasparameters.json should be in the same directory as this function

getPar(*key*)

> **Warning:** method 'mcsas.mcsasdefaultcfg.cInfo.getPar' undocumented

getParVal(*par*)

shortcut method for getting the value of a parameter

loadParams(*fname=None*)

writes the default definitions and bounds for the configuration parameters to self.parameters Can also be used to update existing parameters from supplied filename

parameterNames = []

parameters = None

parseConfig()

Runs through the entire settings, raising warnings where necessary

setParVal(*par*, *value*)

shortcut method for setting the value of a parameter only

writeConfig(*fname*)

writes the configuration to a settings file. Required input parameter is the filename to write to.

mcsas.mcsas.mcsasparameters module

class McSASParameters(*paramDefFile=None*)

Bases: *utils.propertynames.PropertyNames*

Defines the static parameters used for the fitting procedure:

- *model*: an instance of McSAS.model defining the fitting model
- *contribParamBounds*: Bounds of the active (fitting) parameter
- *qBounds*: limits in *q* between which the fit is applied
- ***psiBounds*: limits in azimuthal angle (2D patterns only) to which** the fit is applied
- *qMagnitude*: indicates the multiplier to scale q to m^-1
- *iMagnitude*: indicates the multiplier to scale I to (m sr)^-1
- ***histogramBins*: number of bins to use for size distribution** determination. Does not affect fit

- ***histogramXScale*: can be "log" or "linear", sets the horizontal** axis scaling. Does not affect fit
- ***histogramWeighting*: can be "volume"(recommended) or "number".** Determines whether to plot the volume-weighted size distribution (which is closest to what is measured in a scattering measurement) or a number-weighted size distribution.
- ***deltaRhoSquared* the value (in m^{-4}) of the squared electron** density contrast. Typically on the order of 10^{25} to 10^{30} for X-ray scattering measurements. To get a correct volume fraction, the intensity must be in reciprocal meters, as must q and the object size parameters must be adjusted accordingly.
- ***startFromMinimum*: may be depreciated: starts with an initial guess** consisting of minimum size values rather than random. For testing purposes only.
- ***maxRetries*: if a single optimisation fails, it will be retried** this integer of times.
- ***maskNegativeInt*: may be depreciated due to overlap with similar** functionality in the McSAS.data molule. Setting this will ignore datapoints with intensity values < 0
- *maskZeroInt*: similar to above, but for intensity values = 0

Most of them should be moved to McSAS as dynamic parameters of type *Parameter* which allows them to be configurable in the GUI. Some, esp. *histogramBins* and *histogramXScale* should be moved to a custom *FitParameter* class along with the *active* flag. (WIP)

Extension in progress: the class can take keyword-value pairs to overwrite (some of) the parameter values. Additionally, a default configuration file (json-style) can be provided using kwarg: `paramDefFile = 'path/to/file'` A (limited) set of custom parameters can be supplied in another configuration file that overwrite the defaults: `paramFile = 'path/to/file'`

The order is:

- default file, whose values are superseded by
- custom file, whose values are superseded by
- keyword-value pairs, which only set parameter *values*, or
- keyword-dict pairs, which sets parameter attributes as defined in the supplied dictionary

initialise the defaults and populate the database with values where appropriate default parameter file can be provided using kwarg: paramDefFile = 'path/to/file' relative to application root dir

contribParamBounds = ()

loadParameters(*filename*)

> **Warning:** method 'mcsas.mcsasparameters.McSASParameters.loadParameters' undocumented

model = None

paramDefFile = 'mcsas/mcsasparameters.json'

parameters = []

pickUnit(*unitClass=None*, *displayUnit=None*)
returns a unit object instance of the right class and displayUnit

Parameter(**args*, ***kwargs*)

Warning: function 'mcsas.mcsasparameters.Parameter' undocumented

mcsas.mcsas.plotting module

Defines the format of the final report on success of an MC fit.

class CoordinateFormat(*xname*, *xunit*, *yname*, *yunit*)
Bases: object

A Function object which sets up the particular formatting with axis name and associated unit at initialization time and formats plot coordinates given as (x,y) pair.

class PlotResults(*allRes*, *dataset*, *axisMargin=0.3*, *outputFilename=None*, *modelData=None*, *autoClose=False*, *logToFile=False*, *queue=None*)
Bases: object

This function plots the output of the Monte-Carlo procedure in two windows, with the left window the measured signal versus the fitted measVal (on double-log scale), and the righthand window the size distribution.

figInit(*nHists*, *figureTitle*, *nR=1*)
initialize figure and initialise axes using GridSpec. Each rangeinfo (nR) contains two rows and nHists + 1 columns. the top row axes are for placing text objects: settings and stats. The bottom row axes are for plotting the fits and the histograms TODO: add settings to window title? (next to figure_xy)

formatAlgoInfo()
Preformats the algorithm information ready for printing the colons are surrounded by string-marks, to force laTeX rendering

formatRangeInfo(*parHist*, *RI*, *weighti=0*)
Preformats the rangeInfo results ready for printing

plot1D(*dataset*, *fitX0*, *fitMeasVal*, *qAxis*)
plots 1D data and fit

classmethod plotGrid(*ax*)

Warning: method 'mcsas.plotting.PlotResults.plotGrid' undocumented

plotHist(*plotPar*, *parHist*, *hAxis*, *rangei*)
histogram plot

plotInfo(*InfoAxis*)
plots the range statistics in the small info axes above plots

plotPartial(*fitX0*, *fitMeasVal*, *fitSTD*, *qAxis*, *label='MC partial measVal'*)
plots 1D data and fit

plotStats(*parHist*, *rangei*, *fig*, *InfoAxis*)
plots the range statistics in the small info axes above plots

setAxis(*ah*)

Warning: method 'mcsas.plotting.PlotResults.setAxis' undocumented

class PlotSeriesStats
Bases: object

Simple 1D plotting of series statistics.

plot(*stats*)

> **Warning:** method 'mcsas.plotting.PlotSeriesStats.plot' undocumented

show()

> **Warning:** method 'mcsas.plotting.PlotSeriesStats.show' undocumented

getTextSize(*fig*, *fontProps*)
Returns the width and height of a character for the given font setup. This can be different on each platform.

3.1.2 Module contents

McSAS Core

3.2 mcsas.models package

3.2.1 Submodules

mcsas.models.cylindersisotropic module

class CylindersIsotropic
Bases: *bases.model.sasmodel.SASModel*

Form factor of cylinders previous version (length-fixed) checked against SASfit

absVolume()
Forwarding to usual volume() by default. Can be overridden to include SLD.

formfactor(*dataset*)
Calculates the Rayleigh function of this model. Reimplement this for new models.

parameters = (None, None, None, None, None, None)

shortName = 'SASfit Isotropic Cylinders'

volume()
Calculates the volume of this model, taking compensationExponent into account from input or preset parameters. Reimplement this for new models.

Parameter(**args*, ***kwargs*)

> **Warning:** function 'models.cylindersisotropic.Parameter' undocumented

mcsas.models.cylindersisotropicaspect module

class CylindersIsotropic
Bases: *bases.model.sasmodel.SASModel*

Form factor of cylinders which are radially isotropic (so not spherically isotropic!) !!!completed but not verified!!!

formfactor(*dataset*)
Calculates the Rayleigh function of this model. Reimplement this for new models.

parameters = (None, None, None, None)

shortName = 'Cylinders defined by aspect ratio'

volume()
Calculates the volume of this model, taking compensationExponent into account from input or preset parameters. Reimplement this for new models.

Parameter(**args*, ***kwargs*)

> **Warning:** function 'models.cylindersisotropicaspect.Parameter' undocumented

mcsas.models.cylindersradiallyisotropic module

class CylindersRadiallyIsotropic
Bases: *bases.model.sasmodel.SASModel*

Form factor of cylinders which are radially isotropic (so not spherically isotropic!) !!!completed but not verified!!!

absVolume()
Forwarding to usual volume() by default. Can be overridden to include SLD.

formfactor(*dataset*)
Calculates the Rayleigh function of this model. Reimplement this for new models.

parameters = (None, None, None, None, None)

shortName = 'Radially (in-plane) isotropic cylinders'

volume()
Calculates the volume of this model, taking compensationExponent into account from input or preset parameters. Reimplement this for new models.

Parameter(**args*, ***kwargs*)

> **Warning:** function 'models.cylindersradiallyisotropic.Parameter' undocumented

mcsas.models.cylindersradiallyisotropictilted module

This model is a special case of the radially isotropic cylinders, where the cylinders are also slightly tilted out of the plane parallel to the detector. This out of plane tilt is described using a Gaussian. The integration over this tilt angle is

done over several segments of the Gaussian PDF, with each segment occupying an equal cumulative probability. The centroid value used for the integration is the mass-weighted centre.

class CylindersRadiallyIsotropicTilted
Bases: *bases.model.sasmodel.SASModel*

Form factor of cylinders *UNFINISHED* which are radially isotropic (so not spherically isotropic!)

formfactor(*dataset*)
Calculates the Rayleigh function of this model. Reimplement this for new models.

parameters = (None, None, None, None, None, None)

shortName = 'Cylinders defined by aspect ratio'

volume()
Calculates the volume of this model, taking compensationExponent into account from input or preset parameters. Reimplement this for new models.

Parameter(**args*, ***kwargs*)

Warning: function 'models.cylindersradiallyisotropictilted.Parameter' undocumented

mcsas.models.ellipsoidalcoreshell module

class EllipsoidalCoreShell
Bases: *bases.model.sasmodel.SASModel*

Form factor for an ellipsoidal core shell structure as defined in the SASfit manual (par. 3.2.3) Tested 2014-01-21 against SASfit function with good agreement.

absVolume()
Forwarding to usual volume() by default. Can be overridden to include SLD.

formfactor(*dataset*)
Calculates the Rayleigh function of this model. Reimplement this for new models.

parameters = (None, None, None, None, None, None, None)

shortName = 'Core-Shell Ellipsoid'

volume()
Calculates the volume of this model, taking compensationExponent into account from input or preset parameters. Reimplement this for new models.

Parameter(**args*, ***kwargs*)

Warning: function 'models.ellipsoidalcoreshell.Parameter' undocumented

mcsas.models.ellipsoidsisotropic module

class EllipsoidsIsotropic
Bases: *bases.model.sasmodel.SASModel*

Form factor for a spheroidal structure with semi-axes a = b, c. c can be set to be an aspect ratio with respect to a tested with Ellipsoid II from SASfit 20140626

absVolume()
Forwarding to usual volume() by default. Can be overridden to include SLD.

formfactor(*dataset*)
Calculates the Rayleigh function of this model. Reimplement this for new models.

parameters = (None, None, None, None, None, None)

shortName = 'Isotropic Ellipsoids'

volume()
Calculates the volume of this model, taking compensationExponent into account from input or preset parameters. Reimplement this for new models.

Parameter(**args*, ***kwargs*)

> **Warning:** function 'models.ellipsoidsisotropic.Parameter' undocumented

mcsas.models.gaussianchain module

class GaussianChain
Bases: *bases.model.sasmodel.SASModel*

Form factor of flexible polymer chains which are not selfavoiding and obey Gaussian statistics after *[Debye47]*

See also: http://sasfit.sf.net/manual/Gaussian_Chain#Gauss_2

I_0 = (bp - (k * Rg^2) * eta_s)^2 with k = 1 nm. k * Rg^2 = volume approximation

fixTestParams = functools.partial(<function GaussianChain.fixTestParams>, <class 'mode

formfactor(*dataset*)
Calculates the Rayleigh function of this model. Reimplement this for new models.

parameters = (None, None, None, None)

shortName = 'Gaussian Chain'

volume()
Calculates the volume of this model, taking compensationExponent into account from input or preset parameters. Reimplement this for new models.

Parameter(**args*, ***kwargs*)

> **Warning:** function 'models.gaussianchain.Parameter' undocumented

test()

> **Warning:** function 'models.gaussianchain.test' undocumented

mcsas.models.kholodenko module

class Kholodenko
Bases: *bases.model.sasmodel.SASModel*

Form factor of a worm-like structure after *[Kholodenko93]*

formfactor(*dataset*)
Calculates the Rayleigh function of this model. Reimplement this for new models.

parameters = (None, None, None)

shortName = 'Kholodenko Worm'

volume()
Calculates the volume of this model, taking compensationExponent into account from input or preset parameters. Reimplement this for new models.

calcPcs(*u*)

> **Warning:** function 'models.kholodenko.calcPcs' undocumented

core(*z*, *qValue*, *kuhnLength*, *x*)

> **Warning:** function 'models.kholodenko.core' undocumented

coreIntegral(*qValue*, *kuhnLength*, *x*)

> **Warning:** function 'models.kholodenko.coreIntegral' undocumented

test()

> **Warning:** function 'models.kholodenko.test' undocumented

mcsas.models.lmadensesphere module

class LMADenseSphere
Bases: *bases.model.sasmodel.SASModel*

Form factor of a sphere convoluted with a structure factor, equations 15-17 from Pedersen, J. Appl. Cryst. 27 (1994), 595–608. Correct eqn given in Kinning and Thomas, Macromolecules 17 (1984) 1712. Internally set parameters are volume fraction of the hard spheres, and the multiplication factor /mf/ for an additional stand-off distance between the hard spheres: Rh=mf*R where Rh is the hard-sphere radius ("interaction radius") used in the structure factor, R is the radius of the sphere, and mf is the multiplication factor.

absVolume()
Forwarding to usual volume() by default. Can be overridden to include SLD.

canSmear = True

formfactor(*dataset*)
Calculates the Rayleigh function of this model. Reimplement this for new models.

parameters = (None, None, None, None)

shortName = 'LMADenseSphere'

volume()
Calculates the volume of this model, taking compensationExponent into account from input or preset parameters. Reimplement this for new models.

Parameter(**args*, ***kwargs*)

> **Warning:** function 'models.lmadensesphere.Parameter' undocumented

mcsas.models.sphere module

Parameter(**args*, ***kwargs*)

> **Warning:** function 'models.sphere.Parameter' undocumented

class Sphere
Bases: *bases.model.sasmodel.SASModel*

Form factor of a sphere

absVolume()
Calculates the volume of a sphere taking the scattering length density difference $\Delta\rho$ into account:

$v_{abs}(r, \Delta\rho) = v_{sph}(r) \cdot \Delta\rho^2$

canSmear = True

formfactor(*dataset*)
Calculates the form factor of a sphere defined by:

$F(q, r) = \frac{3\ sin(qr) - qr \cdot cos(qr)}{(qr)^3}$

parameters = (None, None)

shortName = 'Sphere'

surface()
Calculates the surface of a sphere defined by:

$s(r) = 4\pi r^2$

volume()
Calculates the volume of a sphere defined by:

$v(r) = \frac{4\pi}{3} r^3$

test()

> **Warning:** function 'models.sphere.test' undocumented

mcsas.models.sphericalcoreshell module

Parameter(**args*, ***kwargs*)

> **Warning:** function 'models.sphericalcoreshell.Parameter' undocumented

class SphericalCoreShell
Bases: *bases.model.sasmodel.SASModel*

Form factor for a spherical core shell structure as defined in the SASfit manual (par. 3.1.4, Spherical Shell III). One modification is the ability to specify SLD for core, shell and solvent, identical to the notation used in the Core-shell ellipsoid. Compared wiht a SASfit-generated model (both with and without distribution)

absVolume()
Forwarding to usual volume() by default. Can be overridden to include SLD.

formfactor(*dataset*)
Calculates the Rayleigh function of this model. Reimplement this for new models.

parameters = (None, None, None, None, None)

shortName = 'Core-Shell Sphere'

volume()
Calculates the volume of this model, taking compensationExponent into account from input or preset parameters. Reimplement this for new models.

3.2.2 Module contents

3.3 mcsas.gui package

3.3.1 Subpackages

mcsas.gui.bases package

Subpackages

mcsas.gui.bases.mainwindow package

Submodules

mcsas.gui.bases.mainwindow.mainwindow module

class MainWindow(*appversion*, *parent=None*)
Bases: QMainWindow, *gui.bases.mainwindow.ui_mainwindow.Ui_MainWindow*, *gui.bases.mixins.appsettings.AppSettings*

Main window base class.

Provides functionality for storing and loading application settings, managing widgets.

closeEvent(*event*)

> **Warning:** method 'gui.bases.mainwindow.mainwindow.MainWindow.closeEvent' undocumented

getCommandlineArguments()
Get command line arguments, if any.

onStartupSignal

restoreSettings()
Load defaults for settings if missing and available.

show()

> **Warning:** method 'gui.bases.mainwindow.mainwindow.MainWindow.show' undocumented

storeSettings()

> **Warning:** method 'gui.bases.mainwindow.mainwindow.MainWindow.storeSettings' undocumented

mcsas.gui.bases.mainwindow.mainwindow_rc module

qCleanupResources()

> **Warning:** function 'gui.bases.mainwindow.mainwindow_rc.qCleanupResources' undocumented

qInitResources()

> **Warning:** function 'gui.bases.mainwindow.mainwindow_rc.qInitResources' undocumented

mcsas.gui.bases.mainwindow.ui_mainwindow module

class Ui_MainWindow
Bases: `object`

retranslateUi(*MainWindow*)

> **Warning:** method 'gui.bases.mainwindow.ui_mainwindow.Ui_MainWindow.retranslateUi' undocumented

setupUi(*MainWindow*)

> **Warning:** method 'gui.bases.mainwindow.ui_mainwindow.Ui_MainWindow.setupUi' undocumented

Module contents

mcsas.gui.bases.mixins package

Submodules

mcsas.gui.bases.mixins.appsettings module

class AppSettings
Bases: `object`

appSettings

setRootGroup()
Resets any QSettings group(s) currently set.

mcsas.gui.bases.mixins.contextmenuwidget module

class ContextMenuWidget
Bases: `object`

Menu states are mutual exclusive states of available context menu actions.

Menu states map to boolean methods of this object. The names of a state equal names of methods of this object otherwise the menu state is useless. updateMenu() tests the states, i.e. boolean methods for a true return value. The first one specifies the menu to be build. If none matches, the 'default' state will be used.

Menu states are defined by updateMenuState(). The provided list of entry names has to be a partial set of those returned by menuEntries(). Entries may be defined by addMenuEntry().

TODO: Instead of multiple inheritance, convert it to a standalone object. Problems: need to store bound methods as callbacks, to get state on update() The widget this menu applies to needs to be stored only once at init.

action(*name*)

> **Warning:** method 'gui.bases.mixins.contextmenuwidget.ContextMenuWidget.action' undocumented

addMenuEntry(*name=None*, *text=None*, *toolTip=None*, *shortCut=None*, *checkable=False*, *checked=False*, *callbacks=None*, *menuStates=None*)
Argument 'callbacks' is supposed to be a list of methods which will be connected to the QAction.triggered signal.

addMenuEntryAction(*name*, *action*, *menuStates=None*)
Common menu states: default and all. 'all' includes the default state. Everything else expects a method of the same name which evaluates to True or False. It is used to show or hide the respective context menu action. Actions bound to the same state appear grouped in the menu. The underlying ordered dict preserves the order of states.

addMenuSeparator(*menuStates=None*)

> **Warning:** method 'gui.bases.mixins.contextmenuwidget.ContextMenuWidget.addMenuSeparator' undocumented

addToMenuState(*stateName*, **entryNames*)

> **Warning:** method 'gui.bases.mixins.contextmenuwidget.ContextMenuWidget.addToMenuState' undocumented

allMenuStates = '*'

menuEntries(*stateName*)

> **Warning:** method 'gui.bases.mixins.contextmenuwidget.ContextMenuWidget.menuEntries' undocumented

removeMenuEntries(*stateName*)

> **Warning:** method 'gui.bases.mixins.contextmenuwidget.ContextMenuWidget.removeMenuEntries' undocumented

updateMenu(*widget=None*)

> **Warning:** method 'gui.bases.mixins.contextmenuwidget.ContextMenuWidget.updateMenu' undocumented

escapeAmp(*text*)

> **Warning:** function 'gui.bases.mixins.contextmenuwidget.escapeAmp' undocumented

mcsas.gui.bases.mixins.dropwidget module

class DropWidget
Bases: `object`

Drag&Drop support for widgets which inherit from this.

dragEnterEvent(*ev*)

> **Warning:** method 'gui.bases.mixins.dropwidget.DropWidget.dragEnterEvent' undocumented

dropEvent(*ev*)

> **Warning:** method 'gui.bases.mixins.dropwidget.DropWidget.dropEvent' undocumented

mcsas.gui.bases.mixins.titlehandler module

class TitleHandler(*title*)

Bases: *bases.dataset.titlemixin.TitleMixin*

registerUpdateFunc(*func*)

> **Warning:** method 'gui.bases.mixins.titlehandler.TitleHandler.registerUpdateFunc' undocumented

classmethod setup(*parent*, *title*)

Gets a title and the widget this title belongs to.

update(*obj*)

> **Warning:** method 'gui.bases.mixins.titlehandler.TitleHandler.update' undocumented

Module contents

Submodules

mcsas.gui.bases.datalist module

class DataItem(*data*)

Bases: `QTreeWidgetItem`

Generates a QTreeWidgetItem from arbitrary python objects. Storing those objects separately.

data(**args*)

> **Warning:** method 'gui.bases.datalist.DataItem.data' undocumented

dataId()

> **Warning:** method 'gui.bases.datalist.DataItem.dataId' undocumented

getItemProperty(*value*)
For a value, returns this items getter/setter methods according to value type.

static hash32(*data*)
Avoids OverFlowError at setData() with PySide on MacOS.

isRemovable

isTopLevelItem()

> **Warning:** method 'gui.bases.datalist.DataItem.isTopLevelItem' undocumented

listIndex()
Index of this items top most parent in the treewidget.

remove()
Removes the item from its treewidget or parent item.

setAlignment(*alignment*)

> **Warning:** method 'gui.bases.datalist.DataItem.setAlignment' undocumented

setChanged(*column*)

> **Warning:** method 'gui.bases.datalist.DataItem.setChanged' undocumented

setClicked(*column*)

> **Warning:** method 'gui.bases.datalist.DataItem.setClicked' undocumented

update()
Updates this item according to eventually changed data object

wasClickedAndChanged()
Tests if this item was previously clicked and changed in the UI.

class DataList(*parent=None*, *title=None*, *withBtn=True*, *nestedItems=True*)
Bases: `QWidget`, *gui.bases.mixins.dropwidget.DropWidget*, *gui.bases.mixins.contextmenuwidget.ContextMenuWidget*

Manages all loaded spectra.

```
>>> from utilsgui import DialogInteraction, DisplayException
>>> from spectralist import SpectraList
>>> sl = DialogInteraction.instance(SpectraList)
```

Test available actions >>> [str(action.text()) for action in sl.listWidget.actions()] [‘load spectra’, ‘remove’, ‘’, ‘save matrices’, ‘select all’] >>> sl.listWidget.count() 0

Test methods on empty list >>> sl.updateSpectra() >>> sl.removeSelectedSpectra() >>> [sl.getMatrix(i) for i in -1,0,1] [None, None, None] >>> DialogInteraction.query(DisplayException, sl.saveMatrix, ... slot = ‘accept’) >>> sl.selectionChangedSlot()

add(*data*)

> **Warning:** method ‘gui.bases.datalist.DataList.add’ undocumented

clear()

> **Warning:** method ‘gui.bases.datalist.DataList.clear’ undocumented

currentSelection()

> **Warning:** method ‘gui.bases.datalist.DataList.currentSelection’ undocumented

data(*indexOrItem=None*, *selectedOnly=False*)
Returns the list of data for a given list index or list widget item. If none is specified return the data of all items or the data of selected items only, if desired.

expandAll()

> **Warning:** method ‘gui.bases.datalist.DataList.expandAll’ undocumented

fitColumnsToContents(**args*)

> **Warning:** method ‘gui.bases.datalist.DataList.fitColumnsToContents’ undocumented

hasSelection()

> **Warning:** method ‘gui.bases.datalist.DataList.hasSelection’ undocumented

isEmpty()

> **Warning:** method 'gui.bases.datalist.DataList.isEmpty' undocumented

isNotEmpty()

> **Warning:** method 'gui.bases.datalist.DataList.isNotEmpty' undocumented

isRemovableSelected()
True, if there is at least one item selected which may be removed

itemDoubleClicked(*item*, *column*)

> **Warning:** method 'gui.bases.datalist.DataList.itemDoubleClicked' undocumented

itemUpdate(*item*, *column*)
Reimplement to update item if changed by user in GUI

itemsHaveChildren()

> **Warning:** method 'gui.bases.datalist.DataList.itemsHaveChildren' undocumented

leaveEvent(*event*)

> **Warning:** method 'gui.bases.datalist.DataList.leaveEvent' undocumented

loadData(*sourceList=None*, *processSourceFunc=None*, *showProgress=True*, *alignment=None*, ***kwargs*)
Loads a list of data source items.

processSourceFunc is expected to be a function which gets individual elements of sourceList as argument. It returns an arbitrary data item which is then added to this data list widget.

Reimplement it in child classes and it will be called on load button and add action signal.

This method handles exceptions and progress indication.

Test loading a single spectra >>> import utils >>> from tests import TestData >>> from utilsgui import DialogInteraction, UiSettings, fileDialogType >>> from chemsettings import ChemSettings >>> from datafiltersgui import DataFiltersGui >>> from spectralist import SpectraList >>> cs = DialogInteraction.instance(ChemSettings) >>> dfg = DialogInteraction.instance(DataFiltersGui) >>> sl = DialogInteraction.instance(SpectraList, settings = cs) >>> utils.LastPath.path = TestData.spectra(0) >>> DialogInteraction.query(fileDialogType(), sl.loadData, ... slot = 'accept') >>> sl.updateSpectra() >>> utils.LastPath.path = utils.getTempFileName() >>> matrixfiles = DialogInteraction.query(fileDialogType(), sl.saveMatrix, ... slot = 'accept') >>> len(matrixfiles) 1 >>> matrixfiles

Verify written matrix data with existent matrix export >>> TestData.verifyMatrix(TestData.spectra(0), ... matrixfiles[0]) True

removeItems(*indexList*)
Deletes items specified in the given list of indices.

removeSelected()

> **Warning:** method 'gui.bases.datalist.DataList.removeSelected' undocumented

reraiseLast()
Reraise the last error if any and display an error message dialog.

selectAll()
Selects all items in the list if not all are selected. Clears the selection if all items in the list already are selected.

selectionChanged()

> **Warning:** method 'gui.bases.datalist.DataList.selectionChanged' undocumented

setCurrentIndex(*index*)

> **Warning:** method 'gui.bases.datalist.DataList.setCurrentIndex' undocumented

setHeader(*labels=None*)

> **Warning:** method 'gui.bases.datalist.DataList.setHeader' undocumented

setupUi()
Reimplement this in child classes for custom UI configuration.

sigEditingFinished

sigEmpty

sigReceivedUrls

sigRemovedData

sigSelectedData

sigUpdatedData

topLevelItems()

> **Warning:** method 'gui.bases.datalist.DataList.topLevelItems' undocumented

updateData(*selectedOnly=False*, *showProgress=True*, *updateFunc=None*, *prepareFunc=None*, *stopFunc=None*, ***kwargs*)
Calls the provided function on all data items.

The object returned by prepareFunc() is forwarded as optional argument to updateFunc(dataItem, optionalArguments = None).

updateItems()

> **Warning:** method 'gui.bases.datalist.DataList.updateItems' undocumented

mcsas.gui.bases.dockwidget module

class DockWidget(*parent*, *childWidgetType*, **args*, ***kwargs*)
Bases: `QDockWidget`

Widget for docking in a QMainWindow environment.

child

closeEvent(*event*)

> **Warning:** method 'gui.bases.dockwidget.DockWidget.closeEvent' undocumented

onVisibilityChange(*visible*)

> **Warning:** method 'gui.bases.dockwidget.DockWidget.onVisibilityChange' undocumented

mcsas.gui.bases.logwidget module

class LogWidget(*parent=None*, *appversion=None*)
Bases: `QTextBrowser`, *gui.bases.mixins.contextmenuwidget.ContextMenuWidget*

Simple TextEdit which can save its contents to file.

Fill it with text. >>> from gui.utils.dialoginteraction import DialogInteraction >>> from gui.widgets.logwidget import LogWidget >>> testdata = 'blablubbanblubb' >>> te = DialogInteraction.instance(LogWidget) >>> te.append(testdata) >>> te.contents() == testdata True

Call save to file dialog and save. >>> fn = str(DialogInteraction.query(fileDialogType(), te.saveToFile, ... slot = 'accept'))

Verify written data and remove test file. >>> import os >>> data = None >>> if fn is not None: ... data = open(fn).read() ... os.remove(fn) >>> data == testdata True

PlainTextEdit Tests: Simple TextEdit which can save its contents to file.

Fill it with text. >>> from utilsgui import DialogInteraction, fileDialogType >>> from plaintextedit import PlainTextEdit >>> testdata = 'blablubbanblubb' >>> te = DialogInteraction.instance(PlainTextEdit) >>> te.appendPlainText(testdata) >>> te.toPlainText() == testdata True

Call save to file dialog and save. >>> fn = str(DialogInteraction.query(fileDialogType(), te.saveToFile, ... slot = 'accept'))

Verify written data and remove test file. >>> import os >>> data = None >>> if fn is not None: ... data = open(fn).read() ... os.remove(fn) >>> data == testdata True

addWatchDir(*path*)
Adds a directory to check for *plotlog* messages from multiprocessing.

append(*text*)
Appends a new line of text.

appversion

clear()

> **Warning:** method 'gui.bases.logwidget.LogWidget.clear' undocumented

contents()

> **Warning:** method 'gui.bases.logwidget.LogWidget.contents' undocumented

isCopyAvailable()

> **Warning:** method 'gui.bases.logwidget.LogWidget.isCopyAvailable' undocumented

isEmpty()

> **Warning:** method 'gui.bases.logwidget.LogWidget.isEmpty' undocumented

onCloseSlot()
Workaround to exit a program during calculation. Logging a message processes Qt events (see append()). It makes sure this slot is called if connected to a application close signal. For example, emitted by the main window on closeEvent().

saveToFile(*filename=None*)

> **Warning:** method 'gui.bases.logwidget.LogWidget.saveToFile' undocumented

scrollToBottom()

> **Warning:** method 'gui.bases.logwidget.LogWidget.scrollToBottom' undocumented

scrollToTop()

> **Warning:** method 'gui.bases.logwidget.LogWidget.scrollToTop' undocumented

setCopyAvailable(*yes*)

> **Warning:** method 'gui.bases.logwidget.LogWidget.setCopyAvailable' undocumented

title = None

url2href(*text*)
http://daringfireball.net/2010/07/improved_regex_for_matching_urls

mcsas.gui.bases.settingswidget module

class SettingsWidget(*parent=None*)
Bases: QWidget

Provides access to user provided application settings.

Call get('<objectname>') to get an input widget value.

connectInputWidgets(*widget*)

> **Warning:** method 'gui.bases.settingswidget.SettingsWidget.connectInputWidgets' undocumented

get(*key*, *defaultValue=None*)
Retrieves the values for a given key name.

static getEditingFinishedSignal(*widget*)

> **Warning:** method 'gui.bases.settingswidget.SettingsWidget.getEditingFinishedSignal' undocumented

getInputWidget(*datatype*)

> **Warning:** method 'gui.bases.settingswidget.SettingsWidget.getInputWidget' undocumented

getValue(*widget*, *defaultValue=None*)
Retrieves the values for a given widget.

getWidget(*key*)

> **Warning:** method 'gui.bases.settingswidget.SettingsWidget.getWidget' undocumented

set(*key*, *value*)

> **Warning:** method 'gui.bases.settingswidget.SettingsWidget.set' undocumented

setValue(*widget*, *value*)

> **Warning:** method 'gui.bases.settingswidget.SettingsWidget.setValue' undocumented

sigValueChanged

sigValuesChanged

mcsas.gui.bases.settingswidget_test module

class TestSettings(*parent=None*)

Bases: *gui.bases.settingswidget.SettingsWidget*

setupUi(*dummy*)

> **Warning:** method 'gui.bases.settingswidget_test.TestSettings.setupUi' undocumented

testFloatingPointInputBox()

> **Warning:** function 'gui.bases.settingswidget_test.testFloatingPointInputBox' undocumented

testIntegerInputBox()

> **Warning:** function 'gui.bases.settingswidget_test.testIntegerInputBox' undocumented

testTextualInputBox()

> **Warning:** function 'gui.bases.settingswidget_test.testTextualInputBox' undocumented

w

Module contents

mcsas.gui.utils package

Subpackages

mcsas.gui.utils.appversion package

Submodules

mcsas.gui.utils.appversion.appversion module

class AppVersion(*programName=None*, *versionNumber=None*, *organizationName=None*, *organizationDomain=None*, *defaultSettings=None*)

Bases: `object`

Stores version meta data.

defaultSettings()

> **Warning:** method 'gui.utils.appversion.appversion.AppVersion.defaultSettings' undocumented

classmethod isValid(*other*)

> **Warning:** method 'gui.utils.appversion.appversion.AppVersion.isValid' undocumented

name()

> **Warning:** method 'gui.utils.appversion.appversion.AppVersion.name' undocumented

number()

> **Warning:** method 'gui.utils.appversion.appversion.AppVersion.number' undocumented

organizationDomain()

> **Warning:** method 'gui.utils.appversion.appversion.AppVersion.organizationDomain' undocumented

organizationName()

> **Warning:** method 'gui.utils.appversion.appversion.AppVersion.organizationName' undocumented

static updateFile(*module*, *newversion*)
Updates the version string within a given module. Replaces the version string in the source code textually. Assumes there is an AppVersion constructor call in the module.

mcsas.gui.utils.appversion.qappversion module

class QAppVersion(**args*, ***kwargs*)
Bases: *gui.utils.appversion.appversion.AppVersion*

Set QCoreApplication properties based on version meta data.

settingsKey()
Version dependent settings key.

Module contents

Submodules

mcsas.gui.utils.dialoginteraction module

class DialogInteraction(*config*)
Bases: QThread

Simulates user interaction on blocking widgets.

Calls a specified slot on all widgets of a given type after a given delay. Please see tests in other classes for examples.

config: A *dict* containing argument/value pairs. For valid arguments, please see _propertyNames.

classmethod application()

> **Warning:** method 'gui.utils.dialoginteraction.DialogInteraction.application' undocumented

classmethod instance(*instanceType*, **args*, ***kwargs*)

> **Warning:** method 'gui.utils.dialoginteraction.DialogInteraction.instance' undocumented

classmethod query(*queryType*, *blockingFunc*, **args*, ***kwargs*)

> **Warning:** method 'gui.utils.dialoginteraction.DialogInteraction.query' undocumented

classmethod queryInstance(*instanceType*, **args*, ***kwargs*)

> **Warning:** method 'gui.utils.dialoginteraction.DialogInteraction.queryInstance' undocumented

run()

> **Warning:** method 'gui.utils.dialoginteraction.DialogInteraction.run' undocumented

mcsas.gui.utils.displayexception module

class DisplayException(*exception*, *level='critical'*, *fmt=None*)

Bases: QMessageBox

Displays a message box with the text of a provided exception. *level*: one of 'question', 'info', 'warning', 'critical'

```
>>> from utilsgui import DisplayException, DialogInteraction
>>> from utils import AppError
>>> de = DialogInteraction.queryInstance(DisplayException,
...                                      AppError("test"),
...                                      button = 'default')
>>> type(de).__name__
'DisplayException'
>>> de.result() == 0
True
```

classmethod classAndMethodName()

Retrieves some meta info to decribe the error further.

mcsas.gui.utils.filedialog module

File dialogs and convenience functions.

fileDialog(*parent*, *labeltext*, *path*, *directory=False*, *readOnly=True*)

Opens a dialog to select one or more files for reading.

Alternative to native file dialogs.

fileDialogType()

> **Warning:** function 'gui.utils.filedialog.fileDialogType' undocumented

getOpenFiles(*parent*, *labeltext*, *path*, *filefilter=None*, *multiple=True*)

> **Warning:** function 'gui.utils.filedialog.getOpenFiles' undocumented

getSaveDirectory(*parent*, *labeltext*, *path*)

> **Warning:** function 'gui.utils.filedialog.getSaveDirectory' undocumented

getSaveFile(*parent*, *labeltext*, *path*, *filefilter*)

> **Warning:** function 'gui.utils.filedialog.getSaveFile' undocumented

makeFilter(*filterList*)

> **Warning:** function 'gui.utils.filedialog.makeFilter' undocumented

mcsas.gui.utils.progressdialog module

class ProgressDialog(*parent=None*, *title=None*, *count=0*)
Bases: `QProgressDialog`

A progress dialog to visualize calculation status of the program.

Tests the ProgressDialog with 10 steps, cancels in between. >>> import time >>> from utilsgui import ProgressDialog, DialogInteraction >>> pd = DialogInteraction.queryInstance(ProgressDialog, ... slot = 'cancel', ... count = 11) >>> for i in range(0, pd.maximum()): ... dummy = pd.update() ... time.sleep(0.2) >>> pd.wasCanceled() True

Tests again, without cancel this time. >>> pd.reset() >>> for i in range(0, pd.maximum()): ... dummy = pd.update() ... time.sleep(0.2) >>> pd.wasCanceled() False

update()
Updates progress status and returns True if canceled, False otherwise.

mcsas.gui.utils.signal module

tryDisconnect(*sig*, *slot*)
Tries to disconnect signal *sig* from slot *slot*.

mcsas.gui.utils.translate module

tr(*s*)

> **Warning:** function 'gui.utils.translate.tr' undocumented

Module contents

processEventLoop()

> **Warning:** function 'gui.utils.processEventLoop' undocumented

3.3.2 Submodules

mcsas.gui.algorithmwidget module

class AlgorithmWidget(*parent*, *algorithm*, *appSettings*)

Bases: *gui.bases.settingswidget.SettingsWidget*, *gui.bases.mixins.appsettings.AppSettings*

algorithm

Retrieves AlgorithmBase object containing all parameters for this settings.

blockSigValueChanged()

> **Warning:** method 'gui.algorithmwidget.AlgorithmWidget.blockSigValueChanged' undocumented

static clearLayout(*layout*, *newParent=None*)

Removes all widgets from the given layout and reparents them if *newParent* is a sub class of QWidget

inputWidgets

Returns all existing input names (for store/restore).

keys

makeSetting(*param*, *activeBtns=False*)

Creates an input widget for the provided Parameter and configures it appropriately.

makeWidgets(**args*)

> **Warning:** method 'gui.algorithmwidget.AlgorithmWidget.makeWidgets' undocumented

onBackendUpdate()

> **Warning:** method 'gui.algorithmwidget.AlgorithmWidget.onBackendUpdate' undocumented

static removeWidgets(*widget*)

Removes all widgets from the layout of the given widget.

resizeEvent(*resizeEvent*)

Resizes widget based on available width.

resizeWidgets(*targetWidth*)

> **Warning:** method 'gui.algorithmwidget.AlgorithmWidget.resizeWidgets' undocumented

restoreSession(*section=None*)

> **Warning:** method 'gui.algorithmwidget.AlgorithmWidget.restoreSession' undocumented

sigBackendUpdated

storeSession(*section=None*)
Stores current UI configuration to persistent application settings.

uiWidgets
May return a list of input widgets compatible but not associated to a parameter, e.g. for UI configuration. To be overridden in subclasses.

unblockSigValueChanged()

> **Warning:** method 'gui.algorithmwidget.AlgorithmWidget.unblockSigValueChanged' undocumented

updateAll()
Called in MainWindow on calculation start.

updateParam(*param*, *emitBackendUpdated=True*)
Write UI settings back to the algorithm. Processes all input widgets which belong to a certain parameter.

updateUi()
Update input widgets according to possibly changed backend data.

updateWidget(*widget*, *emitBackendUpdated=True*)
Write UI settings back to the algorithm. Gets the parameter associated with a certain widget and processes all related widgets as well.

class SettingsGridWidget(*parent*, *algorithm*, *appSettings=None*, *showParams=None*)
Bases: *gui.algorithmwidget.AlgorithmWidget*

Base class for displaying simple input boxes of various settings arranged on a grid dynamically based on the width of the window.

Additional arguments: *showParams* is a list of parameter names to show in this widget. If not specified it shows all available parameters by default.

resizeWidgets(*targetWidth*)
Creates a new layout with appropriate row/column count.

isNotNone(*lst*)

> **Warning:** function 'gui.algorithmwidget.isNotNone' undocumented

rearrangeWidgets(*layout*, *widgets*, *targetWidth*)

> **Warning:** function 'gui.algorithmwidget.rearrangeWidgets' undocumented

mcsas.gui.calc module

class Calculator

Bases: *utils.hdf.HDFMixin*

algo

hdfLoad(*filehandle*)

load a calculator configuration

hdfWrite(*hdf*)

write a calculator configuration.

indent = ' '

isStopped()

> **Warning:** method 'gui.calc.Calculator.isStopped' undocumented

model

modelActiveParams()

> **Warning:** method 'gui.calc.Calculator.modelActiveParams' undocumented

modelParams()

> **Warning:** method 'gui.calc.Calculator.modelParams' undocumented

nolog = False

postProcess()

> **Warning:** method 'gui.calc.Calculator.postProcess' undocumented

prepare()

Resets series data. Supposed to be called before each run of multiple __call__() invokations.

stop()

> **Warning:** method 'gui.calc.Calculator.stop' undocumented

class OutputFilename(*dataset*, *createDir=True*)
Bases: object

Generates output filenames with a common time stamp and logs appropriate messages.

basename

filename(*kind=None*, *extension='.txt'*)
Creates a file name from data base name, its directory and the current timestamp. It's created once so that all output files have the same base name and timestamp.

filenameVerbose(*kind*, *descr*, *extension='.txt'*)
Returns the file name as in filename() and logs a descriptive message containing the full file name which is usually click-able.

outDir

timestamp

cfgwrite(*self*, *fp*)
Write an .ini-format representation of the configuration state.

mcsas.gui.datawidget module

class DataWidget(*parent*, *appSettings*)
Bases: QWidget, *gui.bases.mixins.appsettings.AppSettings*

buildUi(*dataobj*)

> **Warning:** method 'gui.datawidget.DataWidget.buildUi' undocumented

clearUi()

> **Warning:** method 'gui.datawidget.DataWidget.clearUi' undocumented

makeConfigUi(*config*)

> **Warning:** method 'gui.datawidget.DataWidget.makeConfigUi' undocumented

onBackendUpdate()

> **Warning:** method 'gui.datawidget.DataWidget.onBackendUpdate' undocumented

onDataSelected(*dataobj*)

> **Warning:** method 'gui.datawidget.DataWidget.onDataSelected' undocumented

onEmptyDataList()
Forgets which data settings were already restored.

restoreSession()

> **Warning:** method 'gui.datawidget.DataWidget.restoreSession' undocumented

sigConfig

storeSession()

> **Warning:** method 'gui.datawidget.DataWidget.storeSession' undocumented

mcsas.gui.filelist module

class FileList(*parent=None*, *title=None*, *withBtn=True*, *nestedItems=True*)
Bases: *gui.bases.datalist.DataList*

configFromLast()
Get the data config of the last item in the list.

itemDoubleClicked(*item*, *column*)

> **Warning:** method 'gui.filelist.FileList.itemDoubleClicked' undocumented

loadData(*fileList=None*)
Loads a list of data source items.

processSourceFunc is expected to be a function which gets individual elements of sourceList as argument. It returns an arbitrary data item which is then added to this data list widget.

Reimplement it in child classes and it will be called on load button and add action signal.

This method handles exceptions and progress indication.

Test loading a single spectra >>> import utils >>> from tests import TestData >>> from utilsgui import DialogInteraction, UiSettings, fileDialogType >>> from chemsettings import ChemSettings >>> from datafiltersgui import DataFiltersGui >>> from spectralist import SpectraList >>> cs = DialogInteraction.instance(ChemSettings) >>> dfg = DialogInteraction.instance(DataFiltersGui) >>> sl = DialogInteraction.instance(SpectraList, settings = cs) >>> utils.LastPath.path = TestData.spectra(0) >>> DialogInteraction.query(fileDialogType(), sl.loadData, ... slot = 'accept') >>> sl.updateSpectra() >>> utils.LastPath.path = utils.getTempFileName() >>> matrixfiles = DialogInteraction.query(fileDialogType(), sl.saveMatrix, ... slot = 'accept') >>> len(matrixfiles) 1 >>> matrixfiles

Verify written matrix data with existent matrix export >>> TestData.verifyMatrix(TestData.spectra(0), ... matrixfiles[0]) True

setDataConfig(*dataConfig*)
Propagates the given DataConfig to all DataObj in the list. Makes sure that all data sets have the same configuration finally. Disable this in order to have individual per-data-set configuration.

setupUi()
Reimplement this in child classes for custom UI configuration.

sigSphericalSizeRange

mcsas.gui.liststyle module

makeAlternatingRowColorsTransparent(*widget*)

> **Warning:** function 'gui.liststyle.makeAlternatingRowColorsTransparent' undocumented

setBackgroundStyleSheet(*widget*, *imgpath*)

> **Warning:** function 'gui.liststyle.setBackgroundStyleSheet' undocumented

mcsas.gui.mainwindow module

class MainWindow(*parent=None*, *args=None*)
Bases: *gui.bases.mainwindow.mainwindow.MainWindow*

calc()

> **Warning:** method 'gui.mainwindow.MainWindow.calc' undocumented

calculator
Returns a calculator object.

closeEvent(*closeEvent*)

> **Warning:** method 'gui.mainwindow.MainWindow.closeEvent' undocumented

fileDialog()

> **Warning:** method 'gui.mainwindow.MainWindow.fileDialog' undocumented

initUi()

> **Warning:** method 'gui.mainwindow.MainWindow.initUi' undocumented

keyPressEvent(*keyEvent*)

> **Warning:** method 'gui.mainwindow.MainWindow.keyPressEvent' undocumented

onCloseSignal

onStartStopClick(*checked*)

> **Warning:** method 'gui.mainwindow.MainWindow.onStartStopClick' undocumented

restoreSettings()
Load defaults for settings if missing and available.

setupUi(**args*)

> **Warning:** method 'gui.mainwindow.MainWindow.setupUi' undocumented

storeSettings()

> **Warning:** method 'gui.mainwindow.MainWindow.storeSettings' undocumented

class ToolBox(**args*, ***kwargs*)
Bases: `QToolBox`

QToolBox containing the widgets for user settings. Used to propagate resize events to child widgets to enable responsive behaviour. On MacOS, fixes failed detection of size changes in child widget due to scroll area.

resizeEvent(*event*)

> **Warning:** method 'gui.mainwindow.ToolBox.resizeEvent' undocumented

eventLoop(*args*)
Starts the UI event loop and get command line parser arguments.

mcsas.gui.modelwidget module

class ModelWidget(*parent*, *calculator*, **args*)
Bases: *gui.algorithmwidget.AlgorithmWidget*

algorithm
Retrieves AlgorithmBase object containing all parameters for this settings.

model
Retrieves AlgorithmBase object containing all parameters for this settings.

onDataSelected(*dataobj*)
Gets the data which is currently selected in the UI and rebuilds the model selection box based on compatible models.

restoreSession(*model=None*)
Load last known user settings from persistent app settings.

selectModel(*model*)
model: string containing the name of the model to select. Calls _selectModelSlot() via signal.

setSphericalSizeRange(*minVal*, *maxVal*)

> **Warning:** method 'gui.modelwidget.ModelWidget.setSphericalSizeRange' undocumented

setStatsWidget(*statsWidget*)
Sets the statistics widget to use for updating ranges.

storeSession(*section=None*)
Stores current UI configuration to persistent application settings.

mcsas.gui.optimizationwidget module

class OptimizationWidget(**args*)
Bases: *gui.algorithmwidget.AlgorithmWidget*

onDataSelected(*dataobj*)
Sets defaults for certain types of DataConfig selected, respectively fixes some values.

resizeWidgets(*targetWidth*)
Creates a new layout with appropriate row/column count.

restoreSession(**args*, ***kwargs*)

> **Warning:** method 'gui.optimizationwidget.OptimizationWidget.restoreSession' undocumented

storeSession(**args*, ***kwargs*)
Stores current UI configuration to persistent application settings.

uiWidgets
May return a list of input widgets compatible but not associated to a parameter, e.g. for UI configuration. To be overridden in subclasses.

mcsas.gui.qt module

mcsas.gui.rangelist module

class RangeDialog(*parent=None*, *model=None*)
Bases: QDialog

Creates a modal dialog window to ask the user for a range to be added.

output()

> **Warning:** method 'gui.rangelist.RangeDialog.output' undocumented

class RangeList(*calculator=None*, *appSettings=None*, ***kwargs*)
Bases: *gui.bases.datalist.DataList*, *gui.bases.mixins.appsettings.AppSettings*

append(*histList*)

> **Warning:** method 'gui.rangelist.RangeList.append' undocumented

editEntry()

> **Warning:** method 'gui.rangelist.RangeList.editEntry' undocumented

itemUpdate(*item*, *column*)
Reimplement to update item if changed by user in GUI

loadData(*ranges=None*)
Overridden base class method for adding entries to the list.

mouseDoubleClickEvent(*event*)
Shows RangeDialog on double click.

onRemoval(*removedHistograms*)

> **Warning:** method 'gui.rangelist.RangeList.onRemoval' undocumented

recalc()

> **Warning:** method 'gui.rangelist.RangeList.recalc' undocumented

restoreSession()
Load last known user settings from persistent app settings.

setupUi()
Reimplement this in child classes for custom UI configuration.

storeSession()

> **Warning:** method 'gui.rangelist.RangeList.storeSession' undocumented

updateHistograms()
Called after UI update by sigBackendUpdated from an AlgorithmWidget.

getItemIndex(*comboBox*, *text*)

> **Warning:** function 'gui.rangelist.getItemIndex' undocumented

mcsas.gui.scientrybox module

class SciEntryBox(*parent=None*)
Bases: QLineEdit

fmt

indicateCorrectness(*isValid*)

> **Warning:** method 'gui.scientrybox.SciEntryBox.indicateCorrectness' undocumented

static numberFormat(*decimals=None*)

> **Warning:** method 'gui.scientrybox.SciEntryBox.numberFormat' undocumented

setMaximum(*value*)
Work around issues regarding round-off errors by the 'g' format type when comparing numbers from user input with calculation results.

setMinimum(*value*)
Work around issues regarding round-off errors by the 'g' format type when comparing numbers from user input with calculation results.

setPrefix(*value*)

> **Warning:** method 'gui.scientrybox.SciEntryBox.setPrefix' undocumented

setRange(*lo*, *hi*)

> **Warning:** method 'gui.scientrybox.SciEntryBox.setRange' undocumented

setValue(*value*)

> **Warning:** method 'gui.scientrybox.SciEntryBox.setValue' undocumented

toolTipFmt = 'A value between {lo} and {hi} (including).'

classmethod updateToolTip(*widget*, *decimals=None*)

> **Warning:** method 'gui.scientrybox.SciEntryBox.updateToolTip' undocumented

value()

> **Warning:** method 'gui.scientrybox.SciEntryBox.value' undocumented

class SciEntryValidator(*parent=None*)

Bases: QDoubleValidator

Assumes the associated QLineEdit is provided as parent object in the constructor.

fixup(*input*)

Restricts the value to the valid range defined by setTop() and setBottom(). Limits the precision as well.

validate(*input*, *pos*)

> **Warning:** method 'gui.scientrybox.SciEntryValidator.validate' undocumented

mcsas.gui.settingsgroup module

class AdvancedSettings(**args*, ***kwargs*)

Bases: *gui.settingsgroup.SettingsGroup*, QGroupBox

showAdvanced(*show=False*)

> **Warning:** method 'gui.settingsgroup.AdvancedSettings.showAdvanced' undocumented

updateWidgets(*widget=None*)

> **Warning:** method 'gui.settingsgroup.AdvancedSettings.updateWidgets' undocumented

class DefaultSettings(**args*, ***kwargs*)

Bases: *gui.settingsgroup.SettingsGroup*, QWidget

class SettingsGroup(**args*, ***kwargs*)
Bases: object

rearrangeWidgets(*targetWidth*)

> **Warning:** method 'gui.settingsgroup.SettingsGroup.rearrangeWidgets' undocumented

mcsas.gui.version module

3.3.3 Module contents

3.4 mcsas.log package

3.4.1 Submodules

mcsas.log.log module

Interface and convenience methods for general logging.

addHandler(*handler*)
Set up a new handler and add it for logging.

formatter()
Date and time format for logging, ISO 8601:2004

removeHandler(*handler*)

> **Warning:** function 'log.log.removeHandler' undocumented

replaceHandler(*handler*)

> **Warning:** function 'log.log.replaceHandler' undocumented

replaceStdOutErr(*sout=None*, *serr=None*)
Replaces stdout/err with calls to logging.info/error.

timestamp()
Current local time in seconds.

timestampFormat()
Format for current local time, suitable for file names. >>> timestampFormat() '%Y-%m-%d_%H-%M-%S'

timestampFormatted(*ts=None*)
Current local time. >>> timestamp() == time.strftime("%Y-%m-%d_%H-%M-%S") True

mcsas.log.sink module

Interface and convenience methods for general logging.

class Sink

Bases: object

buf = None

flush()

> **Warning:** method 'log.sink.Sink.flush' undocumented

process(*msg*, *func*)

> **Warning:** method 'log.sink.Sink.process' undocumented

class StdErrSink

Bases: *log.sink.Sink*

write(*msg*)

> **Warning:** method 'log.sink.StdErrSink.write' undocumented

class StdOutSink

Bases: *log.sink.Sink*

write(*msg*)

> **Warning:** method 'log.sink.StdOutSink.write' undocumented

mcsas.log.widgethandler module

class WidgetHandler(*widget*)

Bases: logging.Handler

A logging.Handler which appends messages to some widget.append().

```
>>> import logging
>>> from PyQt4.QtGui import QPlainTextEdit
>>> from utilsgui import DialogInteraction
>>> from logdock import LogHandler
>>> pte = DialogInteraction.instance(QPlainTextEdit)
>>> handler = LogHandler(pte)
>>> msg = "testest"
>>> handler.emit(logging.LogRecord("", 0, "", 0, msg, (), None))
```

(continues on next page)

(continued from previous page)

```
>>> str(pte.toPlainText()) == msg
True
```

emit(*record*)
Do whatever it takes to actually log the specified logging record.

This version is intended to be implemented by subclasses and so raises a NotImplementedError.

widget

3.4.2 Module contents

getWidgetHandlers()
Returns all active WidgetHandlers for logging.

3.5 mcsas.utils package

3.5.1 Submodules

mcsas.utils.binning module

- *binningArray()*: Can be used to do n-by-n pixel binning of 2D detector images. The returned uncertainty is the larger of either the binned uncertainty or the sample standard deviation in the bin.
- *binning1d()*: bins the data and propagates errors, or calculates errors if not initially provided
- *binningWeighted1d()*: Weighted binning, where the intensities of a pixel are divided between the two neighbouring bins depending on the distances to the centres. If error provided is empty, the standard deviation of the intensities in the bins are computed.

array(*value*)

> **Warning:** function 'utils.binning.array' undocumented

binning1d(*q*, *intensity*, *error=None*, *numBins=200*, *stats='std'*)
An unweighted binning routine. The intensities are sorted across bins of equal size. If provided error is empty, the standard deviation of the intensities in the bins are computed.

binningArray(*q*, *psi*, *intensity*, *error*, *s=2*)
This function applies a simple s-by-s binning routine on images. It calculates new error based on old error superseded by standard deviation in a bin.

binningWeighted1d(*q*, *intensity*, *error=None*, *numBins=200*, *stats='se'*)
Implementation of the binning routine written in Matlab. The intensities are divided across the q-range in bins of equal size. The intensities of a pixel are divided between the two neighbouring bins depending on the distances to the centres. If error provided is empty, the standard deviation of the intensities in the bins are computed.

Usage:

```
qbin, ibin, ebin = binning_weighted_1d(q, intensity, error = [],
                                       numBins = 200, stats = 'se')
```

Optional input arguments:

***numBins*: integer indicating the number of bins to divide the intensity** over. Alternatively, this can be an array of equidistant bin centres. If you go this route, depending on the range, not all intensity may be counted.

***stats*: Can be set to 'auto'. This takes the maximum error between** supplied Poisson statistics error-based errors or the standard error.

Written by Brian R. Pauw, 2011, released under BSD open source license.

mcsas.utils.classproperty module

class classproperty
Bases: property

Subclass property to make classmethod properties possible.

Use it like this:

```
@classproperty
@classmethod
def var(cls):
    <code>
```

Getters only, see http://stackoverflow.com/questions/128573/using-property-on-classmethods

mcsas.utils.devtools module

Definitions used during development only. This module is supposed to not being required in a release package.

DBG(**args*, ***kwargs*)

> **Warning:** function 'utils.devtools.DBG' undocumented

DBGF(**args*)

> **Warning:** function 'utils.devtools.DBGF' undocumented

mcsas.utils.error module

Some Error classes.

exception AppError(*msg=''*)
Bases: Exception

classmethod getMessage(*msg=''*)

> **Warning:** method 'utils.error.AppError.getMessage' undocumented

msg = ''
General error with descriptive message to be forwarded and shown to the user in a message box.

exception EmptySelection(*msg=''*)
Bases: *utils.error.AppError*

exception FileError(*msg*, *fn*)
Bases: *utils.error.AppError*

exception InitError(*msg=''*)
Bases: *utils.error.AppError*

exception LoadError(*msg=''*)
Bases: *utils.error.AppError*

exception VerboseError
Bases: *utils.error.AppError*

Verbose Exception containing class and method where it was raised. A cache makes sure VerboseErrors with the same name can be matched.

classmethod new(*name*, *msg=''*)
Creates an exception dynamically. Remembers previous ones for testing

classmethod prefix(*msg*)
Retrieves some meta info to decribe the error further.

mcsas.utils.findmodels module

For use in gui/modelwidget.py, to help find valid calculation models

class FindModels(**searchPaths*)
Bases: `object`

Finds all methods of type ScatteringModel in the subdirectories starting from searchPath. searchPath defaults to the root mcsas pwd + "models". returns a list of full paths, and a list of associated model names

classmethod candidateFiles(**searchPaths*)

> **Warning:** method 'utils.findmodels.FindModels.candidateFiles' undocumented

classmethod getSearchPaths()

> **Warning:** method 'utils.findmodels.FindModels.getSearchPaths' undocumented

classmethod libraryPath()

> **Warning:** method 'utils.findmodels.FindModels.libraryPath' undocumented

classmethod rootName()

> **Warning:** method 'utils.findmodels.FindModels.rootName' undocumented

classproperty(*func*)

> **Warning:** function 'utils.findmodels.classproperty' undocumented

findFiles(*searchPath*, *extension*)
generator for files ending in .py code from: http://stackoverflow.com/questions/2186525

reorder(*indata*, *priorityKeys*)

> **Warning:** function 'utils.findmodels.reorder' undocumented

mcsas.utils.hdf module

Helper functions for HDF5 functionality

HDFCleanup(*infile*)
Unused space is reclaimed by making a copy of the contents in the current hdf 5 file object, and moves the copy in place of the original. if the input argument supplied is a file name instead of an HDF5 object, the method returns nothing. Else, the method returns the new HDF5 object

class HDFMixin
Bases: `object`

classmethod hdfLoad()
Restores an instance of this type from a given HDF file location or group.

hdfStore(*filename*, *rootLocation=None*)
Writes itself to an HDF file at the given position or group.

hdfWrite(*hdf*)
To be overridden by sub classes to store themselves in an HDF structure. *hdf*: a HDFWriter instance.

class HDFWriter(*hdfHandle*, *rootLocation=None*)
Bases: `object`

Represents an open HDF file location in memory and keeps track of the current address/name for reading or writing. Once this object looses scope, its data is actually written to file.

location

log(*msg*)

> **Warning:** method 'utils.hdf.HDFWriter.log' undocumented

classmethod open(*filename*, *rootLocation=None*)

> **Warning:** method 'utils.hdf.HDFWriter.open' undocumented

writeAttribute(*key*, *value*)

> **Warning:** method 'utils.hdf.HDFWriter.writeAttribute' undocumented

writeAttributes(***kwargs*)

> **Warning:** method 'utils.hdf.HDFWriter.writeAttributes' undocumented

writeDataset(*name*, *data*)

> **Warning:** method 'utils.hdf.HDFWriter.writeDataset' undocumented

writeMember(*obj*, *memberName*)

> **Warning:** method 'utils.hdf.HDFWriter.writeMember' undocumented

writeMembers(*obj*, **members*)

> **Warning:** method 'utils.hdf.HDFWriter.writeMembers' undocumented

getCallerInfo(*referenceType=None*, *stackOffset=0*)

referenceType: Stop the search for a frame when this type for a local 'self' is found. *stackOffset*: grab that frame counted from the last instead of search

mcsas.utils.lastpath module

General utilities without GUI dependencies.

class LastPath

Bases: `object`

Stores a file system path for use in file open dialogs.

How to test this platform independent? >>> from utils import LastPath, getHomeDir >>> LastPath.path == getHomeDir() True >>> LastPath.path = '.' >>> LastPath.path == '.' True

classmethod get()

> **Warning:** method 'utils.lastpath.LastPath.get' undocumented

classmethod set(*lastpath*)
Accepts a directory path or a file path. Determines the directory itself in the latter case.

getHomeDir()

> **Warning:** function 'utils.lastpath.getHomeDir' undocumented

mcsas.utils.loadstore module

- *pickleLoad()*: Reads in pickled data from a file (by filename)
- *pickleStore()*: write a block or dictionary to a file (by filename)

pickleLoad(*filename*)
Loads data from a pickle file

pickleStore(*filename*, *somedata*)
Writes python object to a file.

mcsas.utils.mixedmethod module

This module implements a mixedmethod() decorator for class definitions. Basic idea found here: http://www.daniweb.com/software-development/python/code/406393/mixedmethod-mixes-class-and-instance-method-into-one

Usually, a class method works for an instance only. If decorated with @classmethod it works for the class/type and for the instance but ignores all instance data and works with the class data only. Methods in a class decorated with @mixedmethod work with class/type and instances in the same way BUT if the underlying object is a class/type it works on the class/type data. If the underlying object is an instance it works on the instance data. Thus, for instances modifications apply to that individual instance only. For classes modifications apply on the class level which applies changes to all instances created from that.

class mixedmethod(*func*)
Bases: `object`

This decorator mutates a function defined in a class into a 'mixed' class and instance method. Usage:

```
class Spam:
    @mixedmethod
    def egg(selforcls, *args, **kwargs):
        # selforcls is the instance when called on an instance
        # selforcls is the class when called on the class
        pass
```

mcsas.utils.parameter module

FitParameter(**args*, ***kwargs*)

Warning: function 'utils.parameter.FitParameter' undocumented

class FitParameterBase

Bases: *bases.algorithm.parameter.ParameterBase*

Deriving parameters for curve fitting from bases.algorithm.parameter to introduce more specific fit related attributes.

activeRange = functools.partial(<function FitParameterBase.activeRange>, <class 'utils

activeVal = functools.partial(<function FitParameterBase.activeVal>, <class 'utils.para

activeValues = functools.partial(<function _makeGetter.<locals>.getter>, <class 'utils

displayActiveRange = functools.partial(<function FitParameterBase.displayActiveRange>,

hdfStoreAsMember()

Warning: method 'utils.parameter.FitParameterBase.hdfStoreAsMember' undocumented

histograms = functools.partial(<function _makeGetter.<locals>.getter>, <class 'utils.pa

isActive = functools.partial(<function _makeGetter.<locals>.getter>, <class 'utils.para

setActive = functools.partial(<function FitParameterBase.setIsActive>, <class 'utils.pa

setActiveRange = functools.partial(<function FitParameterBase.setActiveRange>, <class

setActiveVal = functools.partial(<function FitParameterBase.setActiveVal>, <class 'uti

setActiveValues = functools.partial(<function _makeSetter.<locals>.setter>, <class 'ut

setDisplayActiveRange = functools.partial(<function FitParameterBase.setDisplayActiveRa

setHistograms = functools.partial(<function _makeSetter.<locals>.setter>, <class 'utils

setIsActive = functools.partial(<function FitParameterBase.setIsActive>, <class 'utils

setValueRange = functools.partial(<function FitParameterBase.setValueRange>, <class 'u

class FitParameterBoolean

Bases: *utils.parameter.FitParameterBase*, *bases.algorithm.parameter.ParameterBoolean*

class FitParameterFloat

Bases: *utils.parameter.FitParameterNumerical*, *bases.algorithm.parameter.ParameterFloat*

class FitParameterLog

Bases: *utils.parameter.FitParameterBase*, *bases.algorithm.parameter.ParameterLog*

class FitParameterNumerical

Bases: *utils.parameter.FitParameterBase*, *bases.algorithm.parameter.ParameterNumerical*

generate(*lower=None*, *upper=None*, *count=1*)

Returns a list of valid parameter values within given bounds. Accepts vectors of individual bounds for lower and upper limit. This allows for inequality parameter constraints. lower, upper: arrays for lower and upper bounds

class FitParameterString
Bases: *utils.parameter.FitParameterBase*, *bases.algorithm.parameter.ParameterString*

class Histogram(*param*, *lower*, *upper*, *binCount=50*, *xscale=None*, *yweight=None*, *autoFollow=True*)
Bases: *bases.dataset.dataset.DataSet*, *bases.dataset.dataset.DisplayMixin*

Stores histogram related settings of a parameter. The results too, eventually(?). yes, please. Stores&calculates rangeInfo() results for all available weighting options.

Creates an histogram with default bin count, will be updated later.

autoFollow

binCount

bins

calc(*contribs*, *paramIndex*, *fractions*)

> **Warning:** method 'utils.parameter.Histogram.calc' undocumented

cdf

classmethod displayData()
Properties used for UI display.

classmethod displayDataDescr()
Descriptive text of fields for UI display.

hdfWrite(*hdf*)

> **Warning:** method 'utils.parameter.Histogram.hdfWrite' undocumented

classmethod integralProps()
All properties needed to properly serialize and restore this histogram. Same order expected by the constructor.

lower

lowerDisplay
Lower limit in display units including the unit text.

moments

observability

param

paramName

updateRange()
Restricts histogram range according to changed parameter range if needed. Checks histogram range against parameter limits.

upper

upperDisplay
Upper limit in display units including the unit text.

xLowerEdge

xMean

xWidth

xrange

xscale

static xscaling(*index=None*)

> **Warning:** method 'utils.parameter.Histogram.xscaling' undocumented

yweight

static yweighting(*index=None*)

> **Warning:** method 'utils.parameter.Histogram.yweighting' undocumented

class Histograms

Bases: `list`

Manages a set of user configured histograms for evaluation after monte-carlo run.

append(*object*) → None – append object to end

> **Warning:** method 'utils.parameter.Histograms.append' undocumented

calc(**args*)

> **Warning:** method 'utils.parameter.Histograms.calc' undocumented

clear() → None – remove all items from L

> **Warning:** method 'utils.parameter.Histograms.clear' undocumented

hdfWrite(*hdf*)

> **Warning:** method 'utils.parameter.Histograms.hdfWrite' undocumented

updateRanges()

Updates ranges of all histograms.

class Moments(*contribs*, *paramIndex*, *valueRange*, *fraction*, *algo=None*)

Bases: object

static fieldNames()

Returns the field names in the same order as str()

fields

Tuple of member data incl. uncertainty for export.

intensity

kurtosis

mean

skew

total

varName = '_kurtosis'

variance

Parameter(**args*, ***kwargs*)

> **Warning:** function 'utils.parameter.Parameter' undocumented

class VectorResult(*vecResult*)

Bases: object

Stores multiple populations of a single result data vector. Calculates statistics at initialization.

full

mean

std

classproperty(*func*)

> **Warning:** function 'utils.parameter.classproperty' undocumented

isActiveFitParam(*param*)

Checks any type of parameter for activeness. Shorter than that below or a try/except clause.

isFitParam(*param*)

> **Warning:** function 'utils.parameter.isFitParam' undocumented

mcsas.utils.pickleinstancemethods module

mcsas.utils.propertynames module

class PropertyNames
Bases: object

classmethod propNames()

> **Warning:** method 'utils.propertynames.PropertyNames.propNames' undocumented

classmethod properties()
Returns all attributes configured in this class.

mcsas.utils.tests module

Utils for testing something.

assertName(*newName*, *errorType*, *noWhitespace=False*)

> **Warning:** function 'utils.tests.assertName' undocumented

isCallable(*obj*)

> **Warning:** function 'utils.tests.isCallable' undocumented

isFrozen()

> **Warning:** function 'utils.tests.isFrozen' undocumented

isInteger(*obj*)

> **Warning:** function 'utils.tests.isInteger' undocumented

isLinux()

> **Warning:** function 'utils.tests.isLinux' undocumented

isList(*obj*)

Warning: function 'utils.tests.isList' undocumented

isMac()

Warning: function 'utils.tests.isMac' undocumented

isMap(*obj*)

Warning: function 'utils.tests.isMap' undocumented

isNonEmptyString(*obj*)

Warning: function 'utils.tests.isNonEmptyString' undocumented

isNumber(*obj*)

Warning: function 'utils.tests.isNumber' undocumented

isSet(*obj*)

Warning: function 'utils.tests.isSet' undocumented

isString(*obj*)

Warning: function 'utils.tests.isString' undocumented

isWindows()

Warning: function 'utils.tests.isWindows' undocumented

testfor(*condition*, *exception*, *errorMessage=''*)

Warning: function 'utils.tests.testfor' undocumented

mcsas.utils.units module

Defines methods for using and manipulating units of variables. Some default magnitude-name dictionaries are provided, but the user can supply their own dictionary if required. Default unit to translate to must be set. Required keyword arguments:

- ***magnitudedict*: a dictionary of magnitude - name pairs. Names must be** unicode strings.
- *simagnitudename*: the si magnitude name.

Example usage:

```
>>> rUnit = Length("nm")
>>> rUnit.siMagnitudeName
u'm'
>>> rUnit.displayMagnitudeName
u'nm'
>>> rUnit.magnitudeConversion
1e-09
```

or:

```
>>> rUnit.toSi(32)
3.2e-08
```

Selecting a default:

```
>>> qUnit = ScatteringVector(u"cm^-1")
>>> qUnit.magnitudeConversion
100.0
```

class Angle(*magnitudeName=None*)
Bases: *utils.units.Unit*

Set up a unit of measurement. The provided magnitude name is used by default in the UI. Using SI units if unspecified.

class Area(*magnitudeName=None*)
Bases: *utils.units.Unit*

Set up a unit of measurement. The provided magnitude name is used by default in the UI. Using SI units if unspecified.

class DynamicViscosity(*magnitudeName=None*)
Bases: *utils.units.Unit*

Set up a unit of measurement. The provided magnitude name is used by default in the UI. Using SI units if unspecified.

class Fraction(*magnitudeName=None*)
Bases: *utils.units.Unit*

Set up a unit of measurement. The provided magnitude name is used by default in the UI. Using SI units if unspecified.

class Length(*magnitudeName=None*)
Bases: *utils.units.Unit*

Set up a unit of measurement. The provided magnitude name is used by default in the UI. Using SI units if unspecified.

class NoUnit(*magnitudeName=None*)
Bases: *utils.units.Unit*

Set up a unit of measurement. The provided magnitude name is used by default in the UI. Using SI units if unspecified.

class SLD(*magnitudeName=None*)
Bases: *utils.units.Unit*

Set up a unit of measurement. The provided magnitude name is used by default in the UI. Using SI units if unspecified.

class ScatteringIntensity(*magnitudeName=None*)
Bases: *utils.units.Unit*

Set up a unit of measurement. The provided magnitude name is used by default in the UI. Using SI units if unspecified.

class ScatteringVector(*magnitudeName=None*)
Bases: *utils.units.Unit*

Set up a unit of measurement. The provided magnitude name is used by default in the UI. Using SI units if unspecified.

class Temperature(*magnitudeName=None*)
Bases: *utils.units.Unit*

test case for special conversions. Done by redefining toSI and toDisplay. Implemented units are given in _magnitudeMap.

Set up a unit of measurement. The provided magnitude name is used by default in the UI. Using SI units if unspecified.

magnitudeConversion
Scaling factor to move from display magnitude to si units. Required display argument:

displaymagnitudename : The name of the magnitude to convert from

Optional display argument:

simagnitudename [The name of the magnitude to convert to.] Defaults to self.siMagnitudeName

Returns: *float* : A scaling factor for display unit to scale to si unit.

toDisplay(*value*)

> **Warning:** method 'utils.units.Temperature.toDisplay' undocumented

toSi(*value*)

> **Warning:** method 'utils.units.Temperature.toSi' undocumented

class Time(*magnitudeName=None*)
Bases: *utils.units.Unit*

Set up a unit of measurement. The provided magnitude name is used by default in the UI. Using SI units if unspecified.

class Unit(*magnitudeName=None*)

Bases: object

Set up a unit of measurement. The provided magnitude name is used by default in the UI. Using SI units if unspecified.

availableMagnitudeNames

displayMagnitude

displayMagnitudeName

hdfLoad(*hdf*)

should go something like this, here, hold my beer.

hdfWrite(*hdf*)

> **Warning:** method 'utils.units.Unit.hdfWrite' undocumented

static invName(*unitString*)

Adds an $^{-1}$ sign or removes it if already present

classmethod magnitude(*name*)

Returns a (numerical) magnitude matching a magnitude name

magnitudeConversion

Scaling factor to move from display magnitude to si units. Required display argument:

displaymagnitudename : The name of the magnitude to convert from

Optional display argument:

simagnitudename [The name of the magnitude to convert to.] Defaults to self.siMagnitudeName

Returns: *float* : A scaling factor for display unit to scale to si unit.

magnitudeMapping = {}

classmethod name()

> **Warning:** method 'utils.units.Unit.name' undocumented

siMagnitude = ''

siMagnitudeName = ''

toDisplay(*value*)

> **Warning:** method 'utils.units.Unit.toDisplay' undocumented

toSi(*value*)

> **Warning:** method 'utils.units.Unit.toSi' undocumented

class Volume(*magnitudeName=None*)
Bases: *utils.units.Unit*

Set up a unit of measurement. The provided magnitude name is used by default in the UI. Using SI units if unspecified.

3.5.2 Module contents

classname(*obj*)

> **Warning:** function 'utils.classname' undocumented

classproperty(*func*)

> **Warning:** function 'utils.classproperty' undocumented

clip(*value*, *minv*, *maxv*)
Expects a range tuple or list consisting of lower and upper limits.

fixFilename(*filename*)
Works around Windows file path length limitation of 260 chars.

hashNumpyArray(*arr*)

> **Warning:** function 'utils.hashNumpyArray' undocumented

mcopen(*fn*, *mode*, *encoding='utf8'*)

> **Warning:** function 'utils.mcopen' undocumented

3.6 mcsas.datafile package

3.6.1 Submodules

mcsas.datafile.arrayfile module

class ArrayFile(*filename*, ***kwargs*)
Bases: *datafile.asciifile.AsciiFile*

A data file containing a single array of data, mostly.

classmethod fileFilter()
classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: def f(cls, arg1, arg2, ...): ... f = classmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

getDataObj()
Creates and returns the appropriate DataObj instance for this file type.

parseLines(*asciiLines*, ***kwargs*)
Parses lines of an ASCII file in order to extract a single array of numbers. Reimplement this in subclasses for different behaviour.

rawArray

classproperty(*func*)

Warning: function 'datafile.arrayfile.classproperty' undocumented

np_array(*value*)

Warning: function 'datafile.arrayfile.np_array' undocumented

mcsas.datafile.asciifile module

class AsciiFile(*filename*, ***kwargs*)
Bases: *datafile.datafile.DataFile*

A generic ascii data file.

classmethod appendFile(*filename*, *data*, ***kwargs*)
like writeFile but appends data to an existing file

classmethod appendHeaderLine(*filename*, *header*)
writes a single-line header to a file consisting of a string or tuple of strings to be joined

classmethod formatData(*data*, ***kwargs*)

Warning: method 'datafile.asciifile.AsciiFile.formatData' undocumented

classmethod formatRow(*row*, ***kwargs*)

> **Warning:** method 'datafile.asciifile.AsciiFile.formatRow' undocumented

classmethod formatValue(*value*)

> **Warning:** method 'datafile.asciifile.AsciiFile.formatValue' undocumented

newline = '\n'

parseLines(*asciiLines*, ***kwargs*)
Parses lines of an ASCII file in order to extract a single array of numbers. Reimplement this in subclasses for different behaviour.

readArray(*asciiLines*, *dataType=<class 'float'>*, *startLine=0*, *endLine=None*, ***kwargs*)
Reads a numpy.array from a specified segment (startLine, endLine) of a line buffer given by asciiLines. Stops at lines incompatible to previous lines read due to different number of fields or incompatible data type. Returns the last line successfully parsed and the populated numpy.array.

readFile(***kwargs*)
Gets a proper file name and returns file data. May modify the instance. To be reimplemented.

readTuple(*fields*, *dataType=<class 'float'>*, ***kwargs*)
Converts each field to the requested datatype. Raises an error if it is incompatible, the line is skipped in that case.

separator = ' '

valueFormat = '{0: 14.6E}'

classmethod writeFile(*filename*, *data*, ***kwargs*)
Gets a proper file name and numpy array and writes it to file. Reimplement this.

classmethod writeHeaderLine(*filename*, *header*)
writes a single-line header to a file consisting of a string or tuple of strings to be joined

np_array(*value*)

> **Warning:** function 'datafile.asciifile.np_array' undocumented

mcsas.datafile.datafile module

class DataFile(*filename*, ***kwargs*)
Bases: `object`

Base class for handling data files. Can be initialized with a file (name) to read or with a data array.

Test error behaviour >>> from utils import DataFile, getTempFileName, getTempFile, FileNotFound >>> fn = getTempFileName() >>> try: DataFile.checkFilename(fn) ... except FileNotFound, e: str(e).find(fn) > 0 True

Prepare test data file >>> fd = getTempFile() >>> l = ['123 234n', '1,23 43.4rn', '2.3; 34,4n', ... '21.2 42 2n', '23,2 3.4 n'] >>> fd.writelines(l) >>> fd.close()

Test data parsing >>> df = DataFile() >>> df.loadFile(fd.name) [('123', '234'), ('1.23', '43.4'), ('2.3', '34.4'), ('21.2', '42', '2'), ('23.2', '3.4')]

Remove test data finally >>> import os >>> if os.path.isfile(fd.name): ... os.remove(fd.name)

classmethod extensions()

> **Warning:** method 'datafile.datafile.DataFile.extensions' undocumented

fileFilter

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

> **class C:** def f(cls, arg1, arg2, ...): ... f = classmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

filename

Absolute path name of this file.

getDataObj()

Creates and returns the appropriate DataObj instance for this file type.

name

The plain name of the file with path and extension stripped.

readFile(***kwargs*)

Gets a proper file name and returns file data. May modify the instance. To be reimplemented.

static sanitizeReadFilename(*filename*)

Checks provided filename for plausibility and updates LastPath.

classmethod sanitizeWriteFilename(*filename*)

Checks and sets the file name to write to.

setFilename(*filename*)

Checks provided filename for plausibility and updates LastPath

write(*filename*, ***kwargs*)

> **Warning:** method 'datafile.datafile.DataFile.write' undocumented

classmethod writeData(*filename*, *data*, ***kwargs*)

Convenience method to write data directly to file.

classmethod writeFile(*filename*, *data*, ***kwargs*)

Gets a proper file name and numpy array and writes it to file. Reimplement this.

classproperty(*func*)

> **Warning:** function 'datafile.datafile.classproperty' undocumented

mcsas.datafile.nxcansasfile module

class NXSHeader(*dataCount*, *description=None*)
Bases: `object`

line(*index*)
Returns the specified line of the header as string.

classmethod maxLines()

> **Warning:** method 'datafile.nxcansasfile.NXSHeader.maxLines' undocumented

class NXcanSASFile(*filename*, ***kwargs*)
Bases: *datafile.datafile.DataFile*

A NXcanSAS file, which is a NeXus-conform HDF5 file for storing corrected SAS data.

dataRoot

classmethod fileFilter()
classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

> **class C:** def f(cls, arg1, arg2, ...): ... f = classmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod getDataObj()
Creates and returns the appropriate DataObj instance for this file type.

rawArray

readFile(***kwargs*)
Gets a proper file name and returns file data. May modify the instance. To be reimplemented.

readItem(*element*)

> **Warning:** method 'datafile.nxcansasfile.NXcanSASFile.readItem' undocumented

classproperty(*func*)

> **Warning:** function 'datafile.nxcansasfile.classproperty' undocumented

mcsas.datafile.pdhfile module

class PDHFile(*filename*, ***kwargs*)
Bases: *datafile.arrayfile.ArrayFile*

classmethod fileFilter()
classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: def f(cls, arg1, arg2, ...): ... f = classmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod formatData(*data*, *description=None*)

> **Warning:** method 'datafile.pdhfile.PDHFile.formatData' undocumented

parseLines(*asciiLines*, ***kwargs*)
Parses lines of an ASCII file in order to extract a single array of numbers. Reimplement this in subclasses for different behaviour.

class PDHHeader(*dataCount*, *description=None*)
Bases: `object`

line(*index*)
Returns the specified line of the header as string.

classmethod maxLines()

> **Warning:** method 'datafile.pdhfile.PDHHeader.maxLines' undocumented

classproperty(*func*)

> **Warning:** function 'datafile.pdhfile.classproperty' undocumented

3.6.2 Module contents

class DataFile(*filename*, ***kwargs*)
Bases: `object`

Base class for handling data files. Can be initialized with a file (name) to read or with a data array.

Test error behaviour >>> from utils import DataFile, getTempFileName, getTempFile, FileNotFound >>> fn = getTempFileName() >>> try: DataFile.checkFilename(fn) ... except FileNotFound, e: str(e).find(fn) > 0 True

Prepare test data file >>> fd = getTempFile() >>> l = ['123 234n', '1,23 43.4rn', '2.3; 34,4n', ... '21.2 42 2n', '23,2 3.4 n'] >>> fd.writelines(l) >>> fd.close()

Test data parsing >>> df = DataFile() >>> df.loadFile(fd.name) [('123', '234'), ('1.23', '43.4'), ('2.3', '34.4'), ('21.2', '42', '2'), ('23.2', '3.4')]

Remove test data finally >>> import os >>> if os.path.isfile(fd.name): ... os.remove(fd.name)

classmethod extensions()

> **Warning:** method 'datafile.DataFile.extensions' undocumented

fileFilter
classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: def f(cls, arg1, arg2, ...): ... f = classmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

filename
Absolute path name of this file.

getDataObj()
Creates and returns the appropriate DataObj instance for this file type.

name
The plain name of the file with path and extension stripped.

readFile(***kwargs*)
Gets a proper file name and returns file data. May modify the instance. To be reimplemented.

static sanitizeReadFilename(*filename*)
Checks provided filename for plausibility and updates LastPath.

classmethod sanitizeWriteFilename(*filename*)
Checks and sets the file name to write to.

setFilename(*filename*)
Checks provided filename for plausibility and updates LastPath

write (*filename*, ***kwargs*)

> **Warning:** method 'datafile.DataFile.write' undocumented

classmethod writeData (*filename*, *data*, ***kwargs*)
Convenience method to write data directly to file.

classmethod writeFile (*filename*, *data*, ***kwargs*)
Gets a proper file name and numpy array and writes it to file. Reimplement this.

class AsciiFile (*filename*, ***kwargs*)
Bases: *datafile.datafile.DataFile*

A generic ascii data file.

classmethod appendFile (*filename*, *data*, ***kwargs*)
like writeFile but appends data to an existing file

classmethod appendHeaderLine (*filename*, *header*)
writes a single-line header to a file consisting of a string or tuple of strings to be joined

classmethod formatData (*data*, ***kwargs*)

> **Warning:** method 'datafile.AsciiFile.formatData' undocumented

classmethod formatRow (*row*, ***kwargs*)

> **Warning:** method 'datafile.AsciiFile.formatRow' undocumented

classmethod formatValue (*value*)

> **Warning:** method 'datafile.AsciiFile.formatValue' undocumented

newline = '\n'

parseLines (*asciiLines*, ***kwargs*)
Parses lines of an ASCII file in order to extract a single array of numbers. Reimplement this in subclasses for different behaviour.

readArray (*asciiLines*, *dataType=<class 'float'>*, *startLine=0*, *endLine=None*, ***kwargs*)
Reads a numpy.array from a specified segment (startLine, endLine) of a line buffer given by asciiLines. Stops at lines incompatible to previous lines read due to different number of fields or incompatible data type. Returns the last line successfully parsed and the populated numpy.array.

readFile (***kwargs*)
Gets a proper file name and returns file data. May modify the instance. To be reimplemented.

readTuple (*fields*, *dataType=<class 'float'>*, ***kwargs*)
Converts each field to the requested datatype. Raises an error if it is incompatible, the line is skipped in that case.

separator = ' '

valueFormat = '{0: 14.6E}'

classmethod writeFile (*filename*, *data*, ***kwargs*)
Gets a proper file name and numpy array and writes it to file. Reimplement this.

classmethod writeHeaderLine (*filename*, *header*)
writes a single-line header to a file consisting of a string or tuple of strings to be joined

class ArrayFile (*filename*, ***kwargs*)
Bases: *datafile.asciifile.AsciiFile*

A data file containing a single array of data, mostly.

classmethod fileFilter ()
classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: def f(cls, arg1, arg2, ...): ... f = classmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

getDataObj ()
Creates and returns the appropriate DataObj instance for this file type.

parseLines (*asciiLines*, ***kwargs*)
Parses lines of an ASCII file in order to extract a single array of numbers. Reimplement this in subclasses for different behaviour.

rawArray

class PDHFile (*filename*, ***kwargs*)
Bases: *datafile.arrayfile.ArrayFile*

classmethod fileFilter ()
classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: def f(cls, arg1, arg2, ...): ... f = classmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod formatData(*data*, *description=None*)

> **Warning:** method 'datafile.PDHFile.formatData' undocumented

parseLines(*asciiLines*, ***kwargs*)
Parses lines of an ASCII file in order to extract a single array of numbers. Reimplement this in subclasses for different behaviour.

class PDHHeader(*dataCount*, *description=None*)
Bases: `object`

line(*index*)
Returns the specified line of the header as string.

classmethod maxLines()

> **Warning:** method 'datafile.PDHHeader.maxLines' undocumented

getFileFilter()
Returns the file filter text of all available data file formats which can be used with a file selection dialog UI.

3.7 mcsas.dataobj package

3.7.1 Submodules

mcsas.dataobj.dataconfig module

class CallbackRegistry
Bases: `object`

callback(*what*, **args*, ***kwargs*)

> **Warning:** method 'dataobj.dataconfig.CallbackRegistry.callback' undocumented

callbackSlots

register(*what*, **func*)

> **Warning:** method 'dataobj.dataconfig.CallbackRegistry.register' undocumented

class DataConfig
Bases: *bases.algorithm.algorithmbase.AlgorithmBase*, *dataobj.dataconfig.CallbackRegistry*

callbackSlots

hdfWrite(*hdf*)

> **Warning:** method 'dataobj.dataconfig.DataConfig.hdfWrite' undocumented

is2d

onUpdatedX0(*x0*)
Sets available range of loaded data.

onUpdatedX1(*x1*)

> **Warning:** method 'dataobj.dataconfig.DataConfig.onUpdatedX1' undocumented

parameters = (None, None, None, None, None, None, None, None)

sampleName

showParams
A list of parameter names which defines the parameters and their ordering shown in a UI. To be overridden in sub classes.

updateFMasks()

> **Warning:** method 'dataobj.dataconfig.DataConfig.updateFMasks' undocumented

updateFuMin()

> **Warning:** method 'dataobj.dataconfig.DataConfig.updateFuMin' undocumented

updateX0Limits()

> **Warning:** method 'dataobj.dataconfig.DataConfig.updateX0Limits' undocumented

updateX0Unit(*newUnit*)
Sets the unit of the x0 vector.

updateX1Limits()

> **Warning:** method 'dataobj.dataconfig.DataConfig.updateX1Limits' undocumented

updateX1Unit(*newUnit*)

> **Warning:** method 'dataobj.dataconfig.DataConfig.updateX1Unit' undocumented

Parameter(**args*, ***kwargs*)

> **Warning:** function 'dataobj.dataconfig.Parameter' undocumented

funcNotInFuncList(*f*, *flst*)
Custom predicate for comparing bounded methods: Duplicate only if instance ID and method name match.

mcsas.dataobj.dataconfig_test module

assertDefaults(*dc*)

> **Warning:** function 'dataobj.dataconfig_test.assertDefaults' undocumented

testCallbacks()

> **Warning:** function 'dataobj.dataconfig_test.testCallbacks' undocumented

testLimits()

> **Warning:** function 'dataobj.dataconfig_test.testLimits' undocumented

testSerialize()

> **Warning:** function 'dataobj.dataconfig_test.testSerialize' undocumented

testSetter()

> **Warning:** function 'dataobj.dataconfig_test.testSetter' undocumented

mcsas.dataobj.dataobj module

Represents input data associated with a measurement.

class DataObj(***kwargs*)
Bases: abc.NewBase

General container for data loaded from file. It offers specialised methods to derive information from the provided data.

accumulate(*others*)

> **Warning:** method 'dataobj.dataobj.DataObj.accumulate' undocumented

config

configType
Returns a compatible DataConfig type.

count

f
The measurement vector.

filename

hasUncertainties
Returns True if this data set has an error bar for its intensities.

hdfWrite(*hdf*)

> **Warning:** method 'dataobj.dataobj.DataObj.hdfWrite' undocumented

initConfig()
Initializes a new data configuration and sets the sample name which is used to differentiate different data objects of the same type later on.

is2d
Returns true if this dataset contains two-dimensional data with psi information available.

modelType
Returns a compatible ScatteringModel type.

sampleName

seriesKey
The Name of the DataObj property to use as series key, hard-coded for now, assuming it exists. It allows to let the user chose from a generated list of properties (todo).

seriesKeyName
Returns the docstring of the property defined by self.seriesKeyProp.

seriesKeyValue
Returns the value of the property defined by self.seriesKeyProp.

setConfig(*config=None*)
Set the configuration of this data object if the type matches.

setFilename(*fn*)
Stores the absolute path to this data file. Should be reviewed when data sets can be created from several files.

classmethod sourceName()
Returns the name of the measurement method.

updateConfig()
Updates the config object based on this data set. All callbacks are run right after this method in setConfig().

x0
First sampling vector.

x1
Second sampling vector.

x2
Third sampling vector.

classproperty(*func*)

> **Warning:** function 'dataobj.dataobj.classproperty' undocumented

mcsas.dataobj.datavector module

A class describing a vector with limits, units, mask and uncertainties

class DataVector(*name*, *raw*, *rawU=None*, *unit=None*)
Bases: object

a class for combining aspects of a particular vector of data. This is intended only as a storage container without additional functionality.

binnedData

binnedDataU

hdfWrite(*hdf*)

> **Warning:** method 'dataobj.datavector.DataVector.hdfWrite' undocumented

limit

limsString

name

rawData

rawDataU

sanitized

sanitizedU

siData

siDataU

unit

validIndices

mcsas.dataobj.sasconfig module

class GaussianSmearing
Bases: *dataobj.sasconfig.SmearingConfig*

inputValid()

> **Warning:** method 'dataobj.sasconfig.GaussianSmearing.inputValid' undocumented

parameters = (None,)

setIntPoints(*q*)
Sets smearing profile integration points for trapezoidal slit. Top (umbra) of trapezoid has full width xt, bottom of trapezoid (penumbra) has full width. Since the smearing function is assumed to be symmetrical, the integration parameters are calculated in the interval [0, xb/2]

showParams
A list of parameter names which defines the parameters and their ordering shown in a UI. To be overridden in sub classes.

updatePLimits(*pLimit*)

> **Warning:** method 'dataobj.sasconfig.GaussianSmearing.updatePLimits' undocumented

updatePUnit(*newUnit*)

> **Warning:** method 'dataobj.sasconfig.GaussianSmearing.updatePUnit' undocumented

updateQLimits(*qLimit*)

> **Warning:** method 'dataobj.sasconfig.GaussianSmearing.updateQLimits' undocumented

updateQUnit(*newUnit*)

> **Warning:** method 'dataobj.sasconfig.GaussianSmearing.updateQUnit' undocumented

updateSmearingLimits(*q*)

> **Warning:** method 'dataobj.sasconfig.GaussianSmearing.updateSmearingLimits' undocumented

Parameter(**args*, ***kwargs*)

> **Warning:** function 'dataobj.sasconfig.Parameter' undocumented

class SASConfig(**args*, ***kwargs*)

Bases: *dataobj.dataconfig.DataConfig*

hdfWrite(*hdf*)

> **Warning:** method 'dataobj.sasconfig.SASConfig.hdfWrite' undocumented

onUpdatedX0(*x0*)

Sets available range of loaded data.

onUpdatedX1(*x1*)

> **Warning:** method 'dataobj.sasconfig.SASConfig.onUpdatedX1' undocumented

prepareSmearing(*q*)

> **Warning:** method 'dataobj.sasconfig.SASConfig.prepareSmearing' undocumented

shortName = 'SAS data configuration'

showParams

A list of parameter names which defines the parameters and their ordering shown in a UI. To be overridden in sub classes.

smearing

updateX0Unit(*newUnit*)

Sets the unit of the x0 vector.

updateX1Unit(*newUnit*)

> **Warning:** method 'dataobj.sasconfig.SASConfig.updateX1Unit' undocumented

class SmearingConfig

Bases: *bases.algorithm.algorithmbase.AlgorithmBase*

Abstract base class, can't be instantiated.

Creates instances from defined parameters and replaces the class attributes accordingly.

hdfWrite(*hdf*)

> **Warning:** method 'dataobj.sasconfig.SmearingConfig.hdfWrite' undocumented

parameters = (None, None, None)

prepared

qOffset

shortName = 'SAS smearing configuration'

updatePLimits(*pLimit*)

> **Warning:** method 'dataobj.sasconfig.SmearingConfig.updatePLimits' undocumented

updatePUnit(*newUnit*)

> **Warning:** method 'dataobj.sasconfig.SmearingConfig.updatePUnit' undocumented

updateQLimits(*qLimit*)

> **Warning:** method 'dataobj.sasconfig.SmearingConfig.updateQLimits' undocumented

updateQUnit(*newUnit*)

> **Warning:** method 'dataobj.sasconfig.SmearingConfig.updateQUnit' undocumented

updateSmearingLimits(*q*)

> **Warning:** method 'dataobj.sasconfig.SmearingConfig.updateSmearingLimits' undocumented

weights

class TrapezoidSmearing

Bases: *dataobj.sasconfig.SmearingConfig*

halfTrapzPDF(*x*, *c*, *d*)

> **Warning:** method 'dataobj.sasconfig.TrapezoidSmearing.halfTrapzPDF' undocumented

inputValid()

> **Warning:** method 'dataobj.sasconfig.TrapezoidSmearing.inputValid' undocumented

onUmbraUpdate()
Value in umbra will not exceed available q.

parameters = (None, None)

setIntPoints(*q*)
sets smearing profile integration points for trapezoidal slit. Top (umbra) of trapezoid has full width xt, bottom of trapezoid (penumbra) has full width. Since the smearing function is assumed to be symmetrical, the integration parameters are calculated in the interval [0, xb/2]

showParams
A list of parameter names which defines the parameters and their ordering shown in a UI. To be overridden in sub classes.

updatePLimits(*pLimit*)

> **Warning:** method 'dataobj.sasconfig.TrapezoidSmearing.updatePLimits' undocumented

updatePUnit(*newUnit*)

> **Warning:** method 'dataobj.sasconfig.TrapezoidSmearing.updatePUnit' undocumented

updateQLimits(*qLimit*)

> **Warning:** method 'dataobj.sasconfig.TrapezoidSmearing.updateQLimits' undocumented

updateQUnit(*newUnit*)

> **Warning:** method 'dataobj.sasconfig.TrapezoidSmearing.updateQUnit' undocumented

updateSmearingLimits(*q*)

> **Warning:** method 'dataobj.sasconfig.TrapezoidSmearing.updateSmearingLimits' undocumented

mcsas.dataobj.sasconfig_test module

assertDefaults(*sc*)

Warning: function 'dataobj.sasconfig_test.assertDefaults' undocumented

testSerialize()

Warning: function 'dataobj.sasconfig_test.testSerialize' undocumented

mcsas.dataobj.sasdata module

Represents data associated with a measurement by small angle scattering (SAS). Some examples and tests.

```
>>> import numpy
>>> testdata = numpy.random.rand(4,4)
>>> testtitle = "some title"
>>> from sasdata import SASData
```

Testing >>> first = SASData(testtitle, testdata) >>> first.title == testtitle True >>> numpy.all(first.rawArray == testdata) True

class SASData(***kwargs*)

Bases: *dataobj.dataobj.DataObj*

Represents one set of data from a unique source (a file, for example).

configType

Returns a compatible DataConfig type.

count

dataContent

shows the content of the loaded data: Q, I, IErr, etc

classmethod displayData()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: def f(cls, arg1, arg2, ...): ... f = classmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod displayDataDescr()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: def f(cls, arg1, arg2, ...): ... f = classmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

modelType
Returns a compatible ScatteringModel type.

p
Q-Vector at which the intensities are measured. Provided for convenience use within models.

pLimsString
Properly formatted q-limits for UI label text.

q
Q-Vector at which the intensities are measured. Provided for convenience use within models.

qLimsString
Properly formatted q-limits for UI label text.

rUnit

shannonChannelEst()

> **Warning:** method 'dataobj.sasdata.SASData.shannonChannelEst' undocumented

shannonChannelEstText

classmethod sourceName()
The type of data source for UI label text.

sphericalSizeEst()

> **Warning:** method 'dataobj.sasdata.SASData.sphericalSizeEst' undocumented

sphericalSizeEstText

updateConfig()
Updates the config object based on this data set. All callbacks are run right after this method in setConfig().

classproperty(*func*)

> **Warning:** function 'dataobj.sasdata.classproperty' undocumented

3.7.2 Module contents

class DataObj(**kwargs*)
Bases: abc.NewBase

General container for data loaded from file. It offers specialised methods to derive information from the provided data.

accumulate(*others*)

> **Warning:** method 'dataobj.DataObj.accumulate' undocumented

config

configType
Returns a compatible DataConfig type.

count

f
The measurement vector.

filename

hasUncertainties
Returns True if this data set has an error bar for its intensities.

hdfWrite(*hdf*)

> **Warning:** method 'dataobj.DataObj.hdfWrite' undocumented

initConfig()
Initializes a new data configuration and sets the sample name which is used to differentiate different data objects of the same type later on.

is2d
Returns true if this dataset contains two-dimensional data with psi information available.

modelType
Returns a compatible ScatteringModel type.

sampleName

seriesKey
The Name of the DataObj property to use as series key, hard-coded for now, assuming it exists. It allows to let the user chose from a generated list of properties (todo).

seriesKeyName
Returns the docstring of the property defined by self.seriesKeyProp.

seriesKeyValue
Returns the value of the property defined by self.seriesKeyProp.

setConfig(*config=None*)
Set the configuration of this data object if the type matches.

setFilename(*fn*)
Stores the absolute path to this data file. Should be reviewed when data sets can be created from several files.

classmethod sourceName()
Returns the name of the measurement method.

updateConfig()
Updates the config object based on this data set. All callbacks are run right after this method in setConfig().

x0
First sampling vector.

x1
Second sampling vector.

x2
Third sampling vector.

class SASData(**kwargs*)
Bases: *dataobj.dataobj.DataObj*

Represents one set of data from a unique source (a file, for example).

configType
Returns a compatible DataConfig type.

count

dataContent
shows the content of the loaded data: Q, I, IErr, etc

classmethod displayData()
classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: def f(cls, arg1, arg2, ...): ... f = classmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod displayDataDescr()
classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: def f(cls, arg1, arg2, ...): ... f = classmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

modelType
Returns a compatible ScatteringModel type.

p
Q-Vector at which the intensities are measured. Provided for convenience use within models.

pLimsString
Properly formatted q-limits for UI label text.

q
Q-Vector at which the intensities are measured. Provided for convenience use within models.

qLimsString
Properly formatted q-limits for UI label text.

rUnit

shannonChannelEst()

> **Warning:** method 'dataobj.SASData.shannonChannelEst' undocumented

shannonChannelEstText

classmethod sourceName()
The type of data source for UI label text.

sphericalSizeEst()

> **Warning:** method 'dataobj.SASData.sphericalSizeEst' undocumented

sphericalSizeEstText

updateConfig()
Updates the config object based on this data set. All callbacks are run right after this method in setConfig().

class DataConfig
Bases: *bases.algorithm.algorithmbase.AlgorithmBase*, *dataobj.dataconfig.CallbackRegistry*

callbackSlots

hdfWrite(*hdf*)

> **Warning:** method 'dataobj.DataConfig.hdfWrite' undocumented

is2d

onUpdatedX0(*x0*)
Sets available range of loaded data.

onUpdatedX1(*x1*)

> **Warning:** method 'dataobj.DataConfig.onUpdatedX1' undocumented

parameters = (None, None, None, None, None, None, None, None)

sampleName

showParams
A list of parameter names which defines the parameters and their ordering shown in a UI. To be overridden in sub classes.

updateFMasks()

> **Warning:** method 'dataobj.DataConfig.updateFMasks' undocumented

updateFuMin()

> **Warning:** method 'dataobj.DataConfig.updateFuMin' undocumented

updateX0Limits()

> **Warning:** method 'dataobj.DataConfig.updateX0Limits' undocumented

updateX0Unit(*newUnit*)
Sets the unit of the x0 vector.

updateX1Limits()

> **Warning:** method 'dataobj.DataConfig.updateX1Limits' undocumented

updateX1Unit(*newUnit*)

> **Warning:** method 'dataobj.DataConfig.updateX1Unit' undocumented

3.8 mcsas.bases package

3.8.1 Subpackages

mcsas.bases.algorithm package

Submodules

mcsas.bases.algorithm.algorithmbase module

class AlgorithmBase
Bases: `object`

Base class for all data filtering algorithms.

Creates instances from defined parameters and replaces the class attributes accordingly.

classmethod factory(**args*, ***kwargs*)

> **Warning:** method 'bases.algorithm.algorithmbase.AlgorithmBase.factory' undocumented

hdfWrite(*hdf*)

> **Warning:** method 'bases.algorithm.algorithmbase.AlgorithmBase.hdfWrite' undocumented

classmethod makeDefault()

> **Warning:** method 'bases.algorithm.algorithmbase.AlgorithmBase.makeDefault' undocumented

classmethod name()

> **Warning:** method 'bases.algorithm.algorithmbase.AlgorithmBase.name' undocumented

param = functools.partial(<function AlgorithmBase.param>, <class 'bases.algorithm.algo

paramCount = functools.partial(<function AlgorithmBase.paramCount>, <class 'bases.algo

params = functools.partial(<function AlgorithmBase.params>, <class 'bases.algorithm.al

classmethod setName(*name*)

> **Warning:** method 'bases.algorithm.algorithmbase.AlgorithmBase.setName' undocumented

setParam = functools.partial(<function AlgorithmBase.setParam>, <class 'bases.algorith

classmethod setParams(**parameters*)
Expects a list of ParameterBase classes/types and sets them as class attributes to this class. They will become instances later, please see __init__()

showParams
A list of parameter names which defines the parameters and their ordering shown in a UI. To be overridden in sub classes.

update(*other*)
Copy parameter values from another algorithm of the same type.

exception AlgorithmError
Bases: `Exception`

exception AlgorithmNameError
Bases: *bases.algorithm.algorithmbase.AlgorithmError*

exception AlgorithmParameterError
Bases: *bases.algorithm.algorithmbase.AlgorithmError*

classproperty(*func*)

> **Warning:** function 'bases.algorithm.algorithmbase.classproperty' undocumented

mcsas.bases.algorithm.algorithmbase_test module

class DummyAlgo
Bases: *bases.algorithm.algorithmbase.AlgorithmBase*

dummy(*v*)

> **Warning:** method 'bases.algorithm.algorithmbase_test.DummyAlgo.dummy' undocumented

parameters = (None, None)

shortName = 'Dummy'

Parameter(**args*, ***kwargs*)

> **Warning:** function 'bases.algorithm.algorithmbase_test.Parameter' undocumented

np_array(*value*)

> **Warning:** function 'bases.algorithm.algorithmbase_test.np_array' undocumented

testParam()
Algorithm without parameters allowed

testParam1()

> **Warning:** function 'bases.algorithm.algorithmbase_test.testParam1' undocumented

testSerialize()

> **Warning:** function 'bases.algorithm.algorithmbase_test.testSerialize' undocumented

testTypeVsInstance()

> **Warning:** function 'bases.algorithm.algorithmbase_test.testTypeVsInstance' undocumented

mcsas.bases.algorithm.numbergenerator module

class NumberGenerator
Bases: object

Base class for number generators. Generates numbers in the interval [0, 1]. Scaling is supposed to happen elsewhere.

classmethod get(*count=1*)

> **Warning:** method 'bases.algorithm.numbergenerator.NumberGenerator.get' undocumented

classmethod hdfWrite(*hdf*)

> **Warning:** method 'bases.algorithm.numbergenerator.NumberGenerator.hdfWrite' undocumented

class RandomExponential
Bases: *bases.algorithm.numbergenerator.NumberGenerator*

classmethod get(*count=1*)

> **Warning:** method 'bases.algorithm.numbergenerator.RandomExponential.get' undocumented

lower = 0.0

upper = 1.0

class RandomExponential1
Bases: *bases.algorithm.numbergenerator.RandomExponential*

Alias class for RandomExponential

class RandomExponential2
Bases: *bases.algorithm.numbergenerator.RandomExponential*

Picks values with inverse logarithmic probability over)0, 1(, as if it were spanning two decades.

upper = 2.0

class RandomExponential3
Bases: *bases.algorithm.numbergenerator.RandomExponential*

Picks values with inverse logarithmic probability over)0, 1(, as if it were spanning three decades.

upper = 3.0

class RandomUniform
Bases: *bases.algorithm.numbergenerator.NumberGenerator*

classmethod get(*count=1*)

> **Warning:** method 'bases.algorithm.numbergenerator.RandomUniform.get' undocumented

class RandomXorShiftUniform

Bases: *bases.algorithm.numbergenerator.NumberGenerator*

Implemented according to xorshift1024* at http://xorshift.di.unimi.it

```
>>> from bases.algorithm.numbergenerator import RandomXorShiftUniform
>>> RandomXorShiftUniform.getSeed()
>>> RandomXorShiftUniform.setSeed()
>>> RandomXorShiftUniform.next()
>>> RandomXorShiftUniform.get()
>>> RandomXorShiftUniform.get(3)
```

classmethod dtype()

> **Warning:** method 'bases.algorithm.numbergenerator.RandomXorShiftUniform.dtype' undocumented

classmethod get(*count=1*)

> **Warning:** method 'bases.algorithm.numbergenerator.RandomXorShiftUniform.get' undocumented

classmethod getSeed()

Generate seed using numpy.

classmethod next()

> **Warning:** method 'bases.algorithm.numbergenerator.RandomXorShiftUniform.next' undocumented

p = None

s = None

classmethod setSeed(*seedData=None*)

> **Warning:** method 'bases.algorithm.numbergenerator.RandomXorShiftUniform.setSeed' undocumented

class RandomXorShiftUniformTest(*methodName='runTest'*)

Bases: unittest.case.TestCase

Tests RandomXorShiftUniform output against reference C implementation. The full path to the executable has to be specified. Call it like this:

```
nosetests bases.algorithm.numbergenerator
```

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

classmethod generateTests()

> **Warning:** method 'bases.algorithm.numbergenerator.RandomXorShiftUniformTest.generateTests' undocumented

getRef()

> **Warning:** method 'bases.algorithm.numbergenerator.RandomXorShiftUniformTest.getRef' undocumented

setUp()
Hook method for setting up the test fixture before exercising it.

tearDown()
Hook method for deconstructing the test fixture after testing it.

test0()

> **Warning:** method 'bases.algorithm.numbergenerator.RandomXorShiftUniformTest.test0' undocumented

test1()

> **Warning:** method 'bases.algorithm.numbergenerator.RandomXorShiftUniformTest.test1' undocumented

test2()

> **Warning:** method 'bases.algorithm.numbergenerator.RandomXorShiftUniformTest.test2' undocumented

test3()

> **Warning:** method 'bases.algorithm.numbergenerator.RandomXorShiftUniformTest.test3' undocumented

test4()

> **Warning:** method 'bases.algorithm.numbergenerator.RandomXorShiftUniformTest.test4' undocumented

lshift(*x*, *s*)

> **Warning:** function 'bases.algorithm.numbergenerator.lshift' undocumented

rshift(*x*, *s*)

> **Warning:** function 'bases.algorithm.numbergenerator.rshift' undocumented

mcsas.bases.algorithm.parameter module

This module defines a generic parameter class for algorithms. It contains meta information which allows for automated UI building. Create sub classes by calling factory() in this module. It creates a new sub class type which inherits ParameterBase:

```
>>> from parameter import factory as paramFactory
>>> ParamType = paramFactory("radius", 1.3, valueRange = (0, 2))
```

Created a new type RadiusParameter:

```
>>> print(ParamType)
<class 'parameter.RadiusParameter'>
```

Using methods on instances work as usual:

```
>>> p = ParamType()
>>> p.name()
'radius'
>>> p.value()
1.3
```

Update the instance:

```
>>> p.setValue(2.4)
>>> p.value()
2.4
```

Changing class default: >>> ParamType.setValue(3.5) >>> ParamType.value() 3.5

Existing instance keep their values: >>> p.value() 2.4

New instances get the updated defaults: >>> q = ParamType() >>> q.value() 3.5

Parameter attributes are accessible on type/class as well as on the instance. Updating an attribute of an instance changes just that individual instance whereas updating an attribute of the type changes that attribute in general for all new instances to be created which is behaves like a default value.

exception DecimalsError
Bases: *bases.algorithm.parameter.ParameterError*

exception DefaultValueError
Bases: *bases.algorithm.parameter.ParameterError*

exception DisplayValuesError
Bases: *bases.algorithm.parameter.ParameterError*

class ParameterBase
Bases: object

Base class for algorithm parameters providing additional information to ease automated GUI building.

classmethod addAttributes(*dictionary*, **names*, ***namesAndValues*)
Sets an *ordered* list of attributes. Initializes the private variable to None and sets a default getter method for each name provided. Additionally, sets *attributeNames* to return all attribute names.

classmethod attributeNames()
Returns an ordered list of attribute names considering multiple inheritance and maintaining its order.

attributes = functools.partial(<function ParameterBase.attributes>, <class 'bases.algo

copy()

> **Warning:** method 'bases.algorithm.parameter.ParameterBase.copy' undocumented

displayName = functools.partial(<function _makeGetter.<locals>.getter>, <class 'bases.

displayValue = functools.partial(<function ParameterBase.displayValue>, <class 'bases.

classmethod dtype()

> **Warning:** method 'bases.algorithm.parameter.ParameterBase.dtype' undocumented

formatDisplayName = functools.partial(<function ParameterBase.formatDisplayName>, <cla

generate(*lower=None*, *upper=None*, *count=1*)
Returns a list of valid parameter values within given bounds. Accepts vectors of individual bounds for lower and upper limit. This allows for inequality parameter constraints. lower, upper: arrays for lower and upper bounds

classmethod get(*key*, *default=None*)
metagetter to get an attribute parameter

hdfStoreAsMember()

> **Warning:** method 'bases.algorithm.parameter.ParameterBase.hdfStoreAsMember' undocumented

hdfWrite(*hdf*)

> **Warning:** method 'bases.algorithm.parameter.ParameterBase.hdfWrite' undocumented

classmethod isDataType(*value*)

> **Warning:** method 'bases.algorithm.parameter.ParameterBase.isDataType' undocumented

name = functools.partial(<function _makeGetter.<locals>.getter>, <class 'bases.algorit

onValueUpdate = functools.partial(<function _makeGetter.<locals>.getter>, <class 'base

classmethod set(*key*, *value*)
metasetter to set an attribute value

setAttributes = functools.partial(<function ParameterBase.setAttributes>, <class 'base

setDisplayName = functools.partial(<function ParameterBase.setDisplayName>, <class 'ba

setDisplayValue = functools.partial(<function ParameterBase.setDisplayValue>, <class '

classmethod setName(*name*)
Changing the name is allowed for the class/type only, not for instances.

setOnValueUpdate = functools.partial(<function _makeSetter.<locals>.setter>, <class 'ba

setValue = functools.partial(<function ParameterBase.setValue>, <class 'bases.algorithm

value = functools.partial(<function _makeGetter.<locals>.getter>, <class 'bases.algori

class ParameterBoolean
Bases: *bases.algorithm.parameter.ParameterBase*

classmethod dtype()

> **Warning:** method 'bases.algorithm.parameter.ParameterBoolean.dtype' undocumented

exception ParameterError
Bases: Exception

class ParameterFloat
Bases: *bases.algorithm.parameter.ParameterNumerical*

decimals = functools.partial(<function _makeGetter.<locals>.getter>, <class 'bases.alg

displayMagnitudeName = functools.partial(<function ParameterFloat.displayMagnitudeName

displayValue = functools.partial(<function ParameterFloat.displayValue>, <class 'bases

displayValueRange = functools.partial(<function ParameterFloat.displayValueRange>, <cl

classmethod dtype()

> **Warning:** method 'bases.algorithm.parameter.ParameterFloat.dtype' undocumented

hdfStoreAsMember()

> **Warning:** method 'bases.algorithm.parameter.ParameterFloat.hdfStoreAsMember' undocumented

classmethod isDataType(*value*)
ParameterNumerical is a fallback for all number not being float.

setDecimals = functools.partial(<function ParameterFloat.setDecimals>, <class 'bases.a

setDisplayValue = functools.partial(<function ParameterFloat.setDisplayValue>, <class

setSuffix = functools.partial(<function ParameterFloat.setSuffix>, <class 'bases.algor

setUnit = functools.partial(<function _makeSetter.<locals>.setter>, <class 'bases.algo

suffix = functools.partial(<function ParameterFloat.suffix>, <class 'bases.algorithm.p

toDisplay = functools.partial(<function ParameterFloat.toDisplay>, <class 'bases.algor

toSi = functools.partial(<function ParameterFloat.toSi>, <class 'bases.algorithm.param

unit = functools.partial(<function _makeGetter.<locals>.getter>, <class 'bases.algorit

exception ParameterGeneratorError
Bases: *bases.algorithm.parameter.ParameterError*

class ParameterLog
Bases: *bases.algorithm.parameter.ParameterFloat*

Used to select an UI input widget with logarithmic behaviour.

exception ParameterNameError
Bases: *bases.algorithm.parameter.ParameterError*

class ParameterNumerical
Bases: *bases.algorithm.parameter.ParameterBase*

clip = functools.partial(<function ParameterNumerical.clip>, <class 'bases.algorithm.p

displayValues = functools.partial(<function ParameterNumerical.displayValues>, <class

classmethod dtype()

> **Warning:** method 'bases.algorithm.parameter.ParameterNumerical.dtype' undocumented

generate(*lower=None*, *upper=None*, *count=1*)
Returns a list of valid parameter values within given bounds. Accepts vectors of individual bounds for lower and upper limit. This allows for inequality parameter constraints. lower, upper: arrays for lower and upper bounds

generator = functools.partial(<function _makeGetter.<locals>.getter>, <class 'bases.al

hdfStoreAsMember()

> **Warning:** method 'bases.algorithm.parameter.ParameterNumerical.hdfStoreAsMember' undocumented

classmethod isDataType(*value*)
ParameterNumerical is a fallback for all number not being float.

max = functools.partial(<function ParameterNumerical.max>, <class 'bases.algorithm.para

min = functools.partial(<function ParameterNumerical.min>, <class 'bases.algorithm.para

setDisplayValues = functools.partial(<function ParameterNumerical.setDisplayValues>, <

setGenerator = functools.partial(<function ParameterNumerical.setGenerator>, <class 'ba

setStepping = functools.partial(<function ParameterNumerical.setStepping>, <class 'base

setSuffix = functools.partial(<function ParameterNumerical.setSuffix>, <class 'bases.a

setValue = functools.partial(<function ParameterNumerical.setValue>, <class 'bases.alg

setValueRange = functools.partial(<function ParameterNumerical.setValueRange>, <class

stepping = functools.partial(<function _makeGetter.<locals>.getter>, <class 'bases.alg

suffix = functools.partial(<function _makeGetter.<locals>.getter>, <class 'bases.algor

valueRange = functools.partial(<function ParameterNumerical.valueRange>, <class 'bases

class ParameterString
Bases: *bases.algorithm.parameter.ParameterBase*

String-based parameter class. The default value should be the first item in the _valueRange list.

classmethod dtype()

> **Warning:** method 'bases.algorithm.parameter.ParameterString.dtype' undocumented

classmethod isDataType(*value*)

> **Warning:** method 'bases.algorithm.parameter.ParameterString.isDataType' undocumented

classmethod setValueRange(*newRange*)

> **Warning:** method 'bases.algorithm.parameter.ParameterString.setValueRange' undocumented

valueRange = functools.partial(<function ParameterString.valueRange>, <class 'bases.al

exception SteppingError
Bases: *bases.algorithm.parameter.ParameterError*

exception SuffixError
Bases: *bases.algorithm.parameter.ParameterError*

exception ValueRangeError
Bases: *bases.algorithm.parameter.ParameterError*

classproperty(*func*)

> **Warning:** function 'bases.algorithm.parameter.classproperty' undocumented

factory(*name*, *value*, *paramTypes=None*, ***kwargs*)

Generates a new Parameter type derived from one of the predefined base classes choosen by the supplied value: Providing a string value results in a type derived from ParameterBase, providing an integer value produces a ParameterNumerical type and a float value results in a ParameterFloat type. Alternatively, a class type cls can be provided which is used as base class for the resulting Parameter class type. Make sure in this case, all attributes mandatory for this base type are provided too.

- *name*: short name of the new parameter without spaces
- *value*: default value from which the type is derived if cls is not given

Optional arguments:

- *paramTypes*: tuple of available parameter types instead of the default
- *cls*: forces a certain Parameter type.
- ***description*: Updates the __doc__ attribute. May be displayed in the UI** somewhere.

generateValues(*numberGenerator*, *defaultRange*, *lower*, *upper*, *count*)

> **Warning:** function 'bases.algorithm.parameter.generateValues' undocumented

mcsas.bases.algorithm.parameter_test module

class Dummy

Bases: object

dummyFunc()

> **Warning:** method 'bases.algorithm.parameter_test.Dummy.dummyFunc' undocumented

Parameter(**args*, ***kwargs*)

> **Warning:** function 'bases.algorithm.parameter_test.Parameter' undocumented

testParameterBaseCopy()

> **Warning:** function 'bases.algorithm.parameter_test.testParameterBaseCopy' undocumented

testParameterCompare()

> **Warning:** function 'bases.algorithm.parameter_test.testParameterCompare' undocumented

testParameterFloat()

> **Warning:** function 'bases.algorithm.parameter_test.testParameterFloat' undocumented

testParameterFloatCopy()

> **Warning:** function 'bases.algorithm.parameter_test.testParameterFloatCopy' undocumented

testParameterName()

> **Warning:** function 'bases.algorithm.parameter_test.testParameterName' undocumented

testParameterNumerical()

> **Warning:** function 'bases.algorithm.parameter_test.testParameterNumerical' undocumented

testParameterNumericalCopy()

> **Warning:** function 'bases.algorithm.parameter_test.testParameterNumericalCopy' undocumented

testParameterNumericalDisplayValues()

> **Warning:** function 'bases.algorithm.parameter_test.testParameterNumericalDisplayValues' undocumented

testParameterNumericalStepping()

> **Warning:** function 'bases.algorithm.parameter_test.testParameterNumericalStepping' undocumented

testParameterNumericalSuffix()

> **Warning:** function 'bases.algorithm.parameter_test.testParameterNumericalSuffix' undocumented

testParameterNumericalValueRange()

> **Warning:** function 'bases.algorithm.parameter_test.testParameterNumericalValueRange' undocumented

testParameterSerialize()

> **Warning:** function 'bases.algorithm.parameter_test.testParameterSerialize' undocumented

Module contents

Parameter(**args*, ***kwargs*)

> **Warning:** function 'bases.algorithm.Parameter' undocumented

mcsas.bases.dataset package

Submodules

mcsas.bases.dataset.dataset module

class DataSet(***kwargs*)

Bases: *bases.dataset.titlemixin.TitleMixin*, *bases.dataset.rawarraymixin.RawArrayMixin*

Container base class for all kinds of data to be passed around in the UI. Knows its originally loaded (raw) data array and its title to be shown in a UI.

class DisplayMixin(***kwargs*)

Bases: object

Provides additional data to display in a list or tree view.

displayData

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

> **class C:** def f(cls, arg1, arg2, ...): ... f = classmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

displayDataDescr
classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: def f(cls, arg1, arg2, ...): ... f = classmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

isRemovable
Returns if this data set may be removed (e.g. from data lists in a GUI)

class ResultMixin(***kwargs*)
Bases: `object`

result
Supposed to return a list of *Result* types.

classproperty(*func*)

> **Warning:** function 'bases.dataset.dataset.classproperty' undocumented

mcsas.bases.dataset.rawarraymixin module

class RawArrayMixin(*rawArray=None*, ***kwargs*)
Bases: `object`

Memorizes the original data before processing it eventually.

classmethod isValidInput(*rawArray*)

> **Warning:** method 'bases.dataset.rawarraymixin.RawArrayMixin.isValidInput' undocumented

rawArray

setRawArray(*rawArray*)

> **Warning:** method 'bases.dataset.rawarraymixin.RawArrayMixin.setRawArray' undocumented

valid

mcsas.bases.dataset.titlemixin module

class TitleMixin(*title=None*, ***kwargs*)
Bases: object

Manages a title of this object for display in a GUI.

title
Data object title

Module contents

mcsas.bases.model package

Submodules

mcsas.bases.model.modeldata module

class ModelData(*cumInt*, *vset*, *wset*, *sset*, *numParams*)
Bases: object

chisqrInt
Make the model intensity comparable to the measured intensity. The difference of both will be calculated in BackgroundScalingFit in order to perform the chi-square test.

cumInt
Returns the cumulated model intensity or signal.

hdfWrite(*hdf*)

> **Warning:** method 'bases.model.modeldata.ModelData.hdfWrite' undocumented

numParams
Returns the number of active (fitted) parameters.

sset
Returns the associated set of surfaces.

volumeFraction(*scaling*)
Returns the volume fraction based on the provided scaling factor to match this model data to the measured data. Assumes that the weights 'self.wset' contain the scatterer volume squared.

vset
Returns the associated set of volumes.

wset
Returns the associated set of weights.

class SASModelData(*cumInt*, *vset*, *wset*, *sset*, *numParams*)
Bases: *bases.model.modeldata.ModelData*

mcsas.bases.model.sasmodel module

class SASModel

Bases: *bases.model.scatteringmodel.ScatteringModel*

calcIntensity(*data*, *compensationExponent=None*)

Returns the intensity *I*, the volume v_{abs} and the intensity weights *w* for a single parameter contribution over all *q*:

$I(q,r) = F^2(q,r) \cdot w(r)$

canSmear = False

getQ(*dataset*)

This is a function that returns Q. In case of smearing, dataset itself is a 2D matrix of Q-values. When smearing is not enabled, dataset.q contains a 1D vector of q.

I do realize that this is not a good way of doing things. This should be replaced at a given point in time by a better solution within sasdata.

modelDataType()

Returns the appropriate ModelData class for this type of model.

weight()

Calculates an intensity weighting used during fitting. It is based on the scatterers volume. It can be modified by a user-defined compensation exponent *c*. The default value is $c = \frac{2}{3}$

$w(r) = v(r)^{2c}$

mcsas.bases.model.scatteringmodel module

class ScatteringModel

Bases: *bases.algorithm.algorithmbase.AlgorithmBase*

Creates instances from defined parameters and replaces the class attributes accordingly.

absVolume()

Forwarding to usual volume() by default. Can be overridden to include SLD.

activeParamCount = functools.partial(<function ScatteringModel.activeParamCount>, <cla

activeParamNames = functools.partial(<function ScatteringModel.activeParamNames>, <cla

activeParams = functools.partial(<function ScatteringModel.activeParams>, <class 'base

calc(*data*, *pset*, *compensationExponent=None*)

Calculates the total intensity and scatterer volume contributions using the current model. *pset* number columns equals the number of active parameters. Returns a ModelData object for a certain type of measurement.

calcIntensity(*data*, *compensationExponent=None*)

Calculates the model intensity which is later compared to the data. Returns a tuple containing an array of the calculated intensities for the grid provided with the data and the volume of a single particle based on the model parameters. Has to be implemented in derived classes specific to a certain type of measurement.

fixTestParams = functools.partial(<function ScatteringModel.fixTestParams>, <class 'ba

formfactor(*dataset*)

Calculates the Rayleigh function of this model. Reimplement this for new models.

generateParameters(*count=1*)
Generates a set of parameters for this model using the predefined Parameter.generator. Allows for different random number distributions.

getModelData(*cumInt*, *vset*, *wset*, *sset*)

> **Warning:** method 'bases.model.scatteringmodel.ScatteringModel.getModelData' undocumented

classmethod getParametersFromFilename(*filename*)
Derives model parameters for testing from reference data file.

hdfWrite(*hdf*)

> **Warning:** method 'bases.model.scatteringmodel.ScatteringModel.hdfWrite' undocumented

modelDataType()
Returns the appropriate ModelData class for this type of model.

surface()
Returns the surface area of a single scatterer. Used for the surface weighted distribution histogram. Returns 0 by default. Reimplement this for a model.

classmethod test(*filename*)
Regression test of a scattering model. File names are expected to contain the parameter values which produce the provided intensity. Otherwise implement fixTestParams() for the particular model.

- *filename*: Name of the file in cls.testDataDir to test against.
- ***cls.testRelErr*: Acceptable mean of relative error against reference** intensity. Default: 1e-5
- ***cls.testVolExp*: Volume compensation exponent, sets the amount of** volume contribution the intensity is scaled by.
- ***cls.testDataDir*: Directory of test data relative to program dir.** Default: "testdata"

update = functools.partial(<function ScatteringModel.update>, <class 'bases.model.scat

updateParamBounds(*bounds*)

> **Warning:** method 'bases.model.scatteringmodel.ScatteringModel.updateParamBounds' undocumented

volume()
Calculates the volume of this model, taking compensationExponent into account from input or preset parameters. Reimplement this for new models.

weight()
A weighting function for the form factor. With SAXS, it is usually the volume squared.

mcsas.bases.model.scatteringmodel_test module

Module contents

3.8.2 Module contents

3.9 mcsas.main module

getScriptPath()
Returns the full path to the current script file which calls this function.

main(*argv=None*)

> **Warning:** function 'main.main' undocumented

makeAbsolutePath(*relpath*)

> **Warning:** function 'main.makeAbsolutePath' undocumented

3.10 McSAS

3.11 mcsas.cxfreeze module

3.11.1 Overview

Creates a standalone program package for a particular platform to be run by restricted users without installing any additional packages.

This script is executable and has to be run on the platform for which a package shall be created. Please follow the instructions below for each particular platform.

Common Package Dependencies:

- Python 2.7
- Qt 4.8 + PySide
- **NumPy and SciPy** In order to work around freeze failures with newer versions it is recommended to stick with Numpy 1.7 and SciPy 1.12 which was tested successfully.
- matplotlib

In addition to the dependencies of the MCSAS package listed above the cx_Freeze package is used for freezing the python source code structure into a standalone package.

Working with Source Code Repositories

In order to download the latest source code repositories of packages such as MCSAS or cx_Freeze a client to Git and Mercurial is required. There are several available, for both Mac OS X and Windows the SourceTree program is recommended.

Windows

A self-contained archive consisting of `MCSAS.exe` and all necessary libraries and files is created by the following command executed in the MCSAS folder:

```
> python cxfreeze.py build_exe
```

Requirements

On a fresh installation of Windows 7 the following packages are required:

- Python 2.7.9
- PySide 1.2.1
- NumPy 1.7.1
- SciPy 0.12.0
- matplotlib 1.4.2 and its requirements:
  - **Six 1.9.0** Install it on the command line by:

    ```
    pip install six-1.9.0-py2.py3-none-any.whl
    ```

  - dateutil 2.4.0
  - pyparsing 2.0.3
- cx_Freeze 4.3.4
- pywin32 219
- h5py HDF5 support, install one of the precompiled Windows packages, such as h5py-2.4.0.win32-py2.7.exe

Mac OS X

After installing the required packages below a disk image file (.dmg) consisting of the application bundle is created by:

```
$ /usr/local/bin/python2 cxfreeze.py bdist_dmg
```

Alternatively, for testing purposes the bundle can be created without packaging into a disk image by:

```
$ /usr/local/bin/python2 cxfreeze.py bdist_mac
```

Requirements

On a fresh installation of OS X 10.8 the following packages are required:

- **Xcode command line tools: for build essentials such as a compiler** (xcode461_cltools_10_86938245a.dmg)
- Python 2.7.9
- Qt 4.8.6
- PySide 1.2.1 / Qt 4.8
- NumPy 1.7.1
- SciPy 0.12.0
- **matplotlib 1.4.2** Install it on the command line by:

```
$ /usr/local/bin/pip install matplotlib-1.4.2-*.whl
```

- h5py HDF5 support, install HDF5 from source first:

```
$ cd hdf5-src
$ ./configure --prefix=/usr/local
$ make && sudo make install
$
$ pip2 install h5py
```

- a modified cx_Freeze 4.3.4 with local modifications for successful app freezing on OS X

  Download the source and install it on the command line by:

```
$ hg clone https://bitbucket.org/ibressler/cx_freeze
$ cd cx_freeze
$ hg co 4.x
$ /usr/local/bin/python2 setup.py install
```

Ubuntu/Linux

Similar to the procedure on Windows a self-contained archive containing all necessary libraries and files is created by:

```
$ python cxfreeze.py build_exe
```

Requirements

On a fresh installation of Ubuntu Linux 14.04 LTS the following packages are required:

- `apt-get install git build-essential python-setuptools python-dev liblapack-dev libfreetype6-dev tk-dev`
- PySide 1.2.4
- NumPy 1.7.2
- SciPy 0.12.1
- matplotlib 1.4.2
- cx_Freeze 4.3.4

- future 0.16.1
- h5py 2.2.1

3.11.2 Internals

class Archiver
Bases: object

archive(*targetPath*)
Creates an archive from the given absolute target directory path. The archive file will have the base name of the last directory of the given path.

execName

getLogFilename()

> **Warning:** method 'cxfreeze.Archiver.getLogFilename' undocumented

class Archiver7z(*filetype='7z'*)
Bases: *cxfreeze.Archiver*

archive(*targetPath*)
Creates an archive from the given absolute target directory path. The archive file will have the base name of the last directory of the given path.

class ArchiverZip
Bases: *cxfreeze.Archiver*

archive(*targetPath*)
Expects an absolute target directory path

includeModels(*includeFilesLst*)

> **Warning:** function 'cxfreeze.includeModels' undocumented

sanitizeVersionNumber(*number*)
Removes non-digits to be compatible with pywin32

3.12 mcsas.autobuild module

3.13 mcsas.mcsas_test module

getExpectedData(*filename*)

> **Warning:** function 'mcsas_test.getExpectedData' undocumented

getSettings (*testfn*, *expectedfn*)
Test settings for mcsas routine. Using number of repetitions and contributions from expected test data to improve comparability.

getTestData (*filename*)

> **Warning:** function 'mcsas_test.getTestData' undocumented

isEqualFloat (*a*, *b*, *tol=1e-16*)
Return TRUE if both float arrays can be considered as equal. *tol* Tolerance in relative mean difference Supposed to be symmetrical.

storeResultData (*filename*, *result*)

> **Warning:** function 'mcsas_test.storeResultData' undocumented

test ()
Testing the algorithm in 1D. Atm, we just test as much as possible. Testing post-processing routines should be separated later as it's deterministic and though easier to test.

3.14 Indices and tables

- genindex
- modindex
- search

CHAPTER 4

How to generate the documentation

4.1 Requirements

- Python, of course
- Sphinx package
- For Latex/PDF generation, there should be a latex environment installed

4.2 Generate a PDF document

```
cd <mcsas>/doc
make latexpdf
```

The resulting `McSAS.pdf` can be found in `<mcsas>/doc/_build/latex/`.

4.3 Generate HTML pages

```
cd <mcsas>/doc
make html
```

The entry point `index.html` can be found in `<mcsas>/doc/_build/html/`.

4.4 Update Source Code Documentation

```
cd <mcsas>/doc
make clean
make apidoc
```

This command automatically generates sphinx documentation files for all source code files in the directory. It assumes the current working directory is the McSAS root directory.

CHAPTER 5

Indices and tables

- genindex
- search

Bibliography

[Debye47] P. Debye, Mollecular-weight determination by light scattering, Journal of Physical and Colloid Chemistry, 51:18–32, 1947.

[Kholodenko93] A. L. Kholodenko. Analytical calculation of the scattering function for polymers of arbitrary flexibility using the dirac propagator. Macromolecules, 26:4179–4183, 1993.

Python Module Index

b

c

d

g

l

m

p

u

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y