## Contents

1. **Contents**
   1.1 Required packages .................................................. 3
   1.2 GSAS-II GUI ............................................................... 4
   1.3 GSASIiobj: Data objects ............................................. 4
   1.4 GSAS-II Utility Modules ............................................. 28
   1.5 GSAS-II GUI Support Routines ..................................... 64
   1.6 GSAS-II GUI Submodules ........................................... 89
   1.7 GSAS-II Structure Submodules ..................................... 110
   1.8 GSASIImapvars: Parameter constraints ........................... 121
   1.9 GSASIIimage: Image calc module ................................ 131
   1.10 GSASIImath: computation module ................................ 134
   1.11 GSASIindex: Cell Indexing Module .............................. 150
   1.12 GSASIIplot: plotting routines ................................... 151
   1.13 GSASII powder calculation module .............................. 159
   1.14 GSAS-II Small Angle Scattering Submodules .................. 165
   1.15 GSASIIscriptable: Scripting Interface ......................... 169
   1.16 GSAS-II Misc Scripts ................................................ 211
   1.17 GSAS-II Import Modules ........................................... 213
   1.18 GSAS-II Export Modules ........................................... 227

**Python Module Index** ................................................... 235

**Index** ........................................................................ 237
The following documentation is intended for those wishing to code within the GSAS-II framework, those planning to understand how GSAS-II works, or for people wishing to develop scripting applications using the API (GSASIIscriptable).

For information on obtaining or learning to use GSAS-II, please see the information on the GSAS-II home page: https://subversion.xray.aps.anl.gov/trac/pyGSAS and pages referenced there.
1.1 Required packages

GSAS-II requires a standard Python interpreter to be installed, as well as several separately-developed packages. GSAS-II is being developed using Python 2.7, 3.6 and 3.7. At this point we think that most sections of the code have been exercised in Python 2 and 3, but bugs are still expected (please report them). Our goal is to keep the code compliant with both Python 2.7 and 3.x for the immediate future.

Note that the packages listed below are not distributed as part of the Python standard library. We use the free Anaconda Python (https://www.anaconda.com/) distribution (and provide installers based on that), but there are many other fine distributions, such as Enthought Inc.’s Canopy and Python(x,y), see here: https://www.python.org/downloads/alternatives/. We do some testing using the older Enthought Python Distribution (EPD); this is known to have some problems with reading CIFs and encourage updating from that.

1.1.1 GUI Requirements

When using the GSAS-II graphical user interface (GUI), the following Python extension packages are required:

- wxPython (http://wxpython.org/docs/api/). Note that GSAS-II has been tested with wxPython 2.8.x, 3.0.x and 4.0.x. We encourage use of 3.0 with Python 2.7 and 4.x with Python 3.x.
- NumPy (http://docs.scipy.org/doc/numpy/reference/),
- SciPy (http://docs.scipy.org/doc/scipy/reference/),
- matplotlib (http://matplotlib.org/contents.html) and
- PyOpenGL (http://pyopengl.sourceforge.net/documentation). Note: a copy of this is distributed with GSAS-II (at present) and will be installed if the Python setuptools package is present.

Several packages are used in sections of the code, but are not required. If these packages are not present, warning messages may be generated if they would be needed, but the vast bulk of GSAS-II will function normally.

- Pillow (https://pillow.readthedocs.org) or PIL (http://www.pythonware.com/products/pil/). This is used to save and read certain types of images.
• h5py is the HDF5 interface and hdf5 is the support package. These packages are (not surprisingly) required to import images from HDF5 files. If these libraries are not present, the HDF5 importer(s) will not appear in the import menu and a warning message appears on GSAS-II startup.

• imageio is used to make movies.

• svn: When using Anaconda we also encourage installation of the svn (subversion) conda package. This is not actually part of Python and can be installed directly into your system’s configuration. It is used by GSAS-II to download updates to our code.

1.1.2 Scripting Requirements

When using the GSAS-II scripting interface (GSASIIscriptable), only the following Python extension packages are required:

• NumPy (http://docs.scipy.org/doc/numpy/reference/),
• SciPy (http://docs.scipy.org/doc/scipy/reference/).

Note that some sections of the code may require matplotlib (http://matplotlib.org/contents.html), Pillow (https://pillow.readthedocs.org) (or PIL, http://www.pythonware.com/products/pil/), or h5py + hdf5 to function but scripts will load and run without these.

1.2 GSAS-II GUI

This is the script to start the GSAS-II graphical user interface (GUI). This script imports GSASIIpath, which does some minor initialization and then (before any wxPython calls can be made) creates a wx.App application. A this point GSASIIpath.SetBinaryPath() is called to establish the directory where GSAS-II binaries are found. If the binaries are not installed or are incompatible with the OS/Python packages, the user is asked if they should be updated from the subversion site. The wxPython app is then passed to GSASIIdataGUI.GSASIImain(), which creates the GSAS-II GUI and finally the event loop is started.

class GSASII.G2App (*args, **kwargs)

    Used to create a wx python application for the GUI for Mac. Customized to implement drop of GPX files onto app.

    ClearStartup()

        Call this after app startup complete because a Drop event is posted when GSAS-II is initially started.

1.3 GSASIIobj: Data objects

This module defines and/or documents the data structures used in GSAS-II, as well as provides misc. support routines.

1.3.1 Constraints Tree Item

Constraints are stored in a dict, separated into groups. Note that parameter are named in the following pattern, p:h:<var>:n, where p is the phase number, h is the histogram number <var> is a variable name and n is the parameter number. If a parameter does not depend on a histogram or phase or is unnumbered, that number is omitted. Note that the contents of each dict item is a List where each element in the list is a constraint definition objects. The constraints in this form are converted in GSASIIstrIO.ProcessConstraints() to the form used in GSASIImapvars. The keys in the Constraints dict are:
<table>
<thead>
<tr>
<th>key</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hist</td>
<td>This specifies a list of constraints on histogram-related parameters, which will be of form :h:&lt;var&gt;:n.</td>
</tr>
<tr>
<td>HAP</td>
<td>This specifies a list of constraints on parameters that are defined for every histogram in each phase and are of form p:h:&lt;var&gt;:n.</td>
</tr>
<tr>
<td>Phase</td>
<td>This specifies a list of constraints on phase parameters, which will be of form p::&lt;var&gt;:n.</td>
</tr>
<tr>
<td>Global</td>
<td>This specifies a list of constraints on parameters that are not tied to a histogram or phase and are of form ::&lt;var&gt;:n</td>
</tr>
</tbody>
</table>

Each constraint is defined as an item in a list. Each constraint is of form:

```
[[<mult1>, <var1>], [<mult2>, <var2>], ..., <fixedval>, <varyflag>, <constype>]
```

Where the variable pair list item containing two values [<mult>, <var>], where:

- `<mult>` is a multiplier for the constraint (float)
- `<var>` a `G2VarObj` object (previously a str variable name of form ‘p:h:name[:at]’

Note that the last three items in the list play a special role:

- `<fixedval>` is the fixed value for a `constant equation` (`constype=c`) constraint or is None. For a `New variable` (`constype=f`) constraint, a variable name can be specified as a str (used for externally generated constraints)
- `<varyflag>` is True or False for `New variable` (`constype=f`) constraints or is None. This indicates if this variable should be refined.
- `<constype>` is one of four letters, ‘e’, ‘c’, ‘h’, ‘f’ that determines the type of constraint:
  - ‘e’ defines a set of equivalent variables. Only the first variable is refined (if the appropriate refine flag is set) and all other equivalent variables in the list are generated from that variable, using the appropriate multipliers.
  - ‘c’ defines a constraint equation of form, \( m_1 \times var_1 + m_2 \times var_2 + ... = c \)
  - ‘h’ defines a variable to hold (not vary). Any variable on this list is not varied, even if its refinement flag is set. Only one [mult, var] pair is allowed in a hold constraint and the mult value is ignored. This is of particular value when needing to hold one or more variables where a single flag controls a set of variables such as, coordinates, the reciprocal metric tensor or anisotropic displacement parameter.
  - ‘f’ defines a new variable (function) according to relationship newvar = \( m_1 \times var_1 + m_2 \times var_2 + ... \)

### 1.3.2 Covariance Tree Item

The Covariance tree item has results from the last least-squares run. They are stored in a dict with these keys:
### GSAS-II Developers Documentation, Release version 4249

#### 1.3.3 Phase Tree Items

Phase information is stored in the GSAS-II data tree as children of the Phases item in a dict with keys:

<table>
<thead>
<tr>
<th>key</th>
<th>sub-key</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>newCellDict</strong></td>
<td>dict</td>
<td>with lattice parameters computed by <code>GSASIIstrMath.GetNewCellParms()</code></td>
</tr>
<tr>
<td><strong>title</strong></td>
<td>str</td>
<td>Name of gpx file(?)</td>
</tr>
<tr>
<td><strong>variables</strong></td>
<td>list</td>
<td>Values for all N refined variables (list of float values, length N, ordered to match varyList)</td>
</tr>
<tr>
<td><strong>sig</strong></td>
<td>list</td>
<td>Uncertainty values for all N refined variables (list of float values, length N, ordered to match varyList)</td>
</tr>
<tr>
<td><strong>varyList</strong></td>
<td>list</td>
<td>List of directly refined variables (list of str values, length N)</td>
</tr>
<tr>
<td><strong>newAtomDict</strong></td>
<td>dict</td>
<td>with atom position values computed in <code>GSASIIstrMath.ApplyXYZshifts()</code></td>
</tr>
<tr>
<td><strong>Rvals</strong></td>
<td>dict</td>
<td>R-factors, GOF, Marquardt value for last refinement cycle</td>
</tr>
<tr>
<td>Nobs</td>
<td>int</td>
<td>Number of observed data points</td>
</tr>
<tr>
<td>Rwp</td>
<td>float</td>
<td>overall weighted profile R-factor (%)</td>
</tr>
<tr>
<td>chisq</td>
<td>float</td>
<td>sum[w*(Iobs-Icalc)**2] for all data note this is not the reduced chi squared</td>
</tr>
<tr>
<td>lamMax</td>
<td>float</td>
<td>Marquardt value applied to Hessian diagonal</td>
</tr>
<tr>
<td>GOF</td>
<td>float</td>
<td>The goodness-of-fit, aka square root of the reduced chi squared.</td>
</tr>
<tr>
<td>covMatrix</td>
<td>np.array</td>
<td>The (NxN) covariance matrix</td>
</tr>
</tbody>
</table>

---

Continued on next page
<table>
<thead>
<tr>
<th>key</th>
<th>sub-key</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>vdWRadii</td>
<td></td>
<td>dict of van der Waals radii for each atom type</td>
</tr>
<tr>
<td>Color</td>
<td></td>
<td>Colors for atoms (list of (r,b,g) triplets)</td>
</tr>
<tr>
<td>AtomTypes</td>
<td></td>
<td>List of atom types</td>
</tr>
<tr>
<td>AtomMass</td>
<td></td>
<td>List of masses for atoms</td>
</tr>
<tr>
<td>doPawley</td>
<td></td>
<td>Flag for Pawley intensity extraction (bool)</td>
</tr>
<tr>
<td>NoAtoms</td>
<td></td>
<td>Number of atoms per unit cell of each type (dict)</td>
</tr>
<tr>
<td>Pawley dmin</td>
<td></td>
<td>maximum Q (as d-space) to use for Pawley extraction (float)</td>
</tr>
<tr>
<td>BondRadii</td>
<td></td>
<td>Default radius for each atom used to compute interatomic distances (list of floats)</td>
</tr>
<tr>
<td>AngleRadii</td>
<td></td>
<td>Default radius for each atom used to compute interatomic angles (list of floats)</td>
</tr>
<tr>
<td>DisAgICtls</td>
<td></td>
<td>Dict with distance/angle search controls, which has keys ‘Name’, ‘AtomTypes’, ‘BondRadii’, ‘AngleRadii’ which are as above except are possibly edited. Also contains ‘Factors’, which is a 2 element list with a multiplier for bond and angle search range [typically (0.85,0.85)].</td>
</tr>
<tr>
<td>ranId</td>
<td></td>
<td>unique random number Id for phase (int)</td>
</tr>
<tr>
<td>pld</td>
<td></td>
<td>Phase Id number for current project (int).</td>
</tr>
<tr>
<td>Atoms</td>
<td></td>
<td>Atoms in phase as a list of lists. The outer list is for each atom, the inner list contains varying items depending on the type of phase, see the Atom Records description. (list of lists)</td>
</tr>
<tr>
<td>Drawing</td>
<td></td>
<td>Display parameters (dict)</td>
</tr>
<tr>
<td>ballScale</td>
<td></td>
<td>Size of spheres in ball-and-stick display (float)</td>
</tr>
<tr>
<td>bondList</td>
<td></td>
<td>dict with bonds</td>
</tr>
<tr>
<td>contourLevel</td>
<td></td>
<td>map contour level in e/A^3 (float)</td>
</tr>
<tr>
<td>showABC</td>
<td></td>
<td>Flag to show view point triplet (bool). True=show.</td>
</tr>
<tr>
<td>viewDir</td>
<td></td>
<td>cartesian viewing direction (np.array with three elements)</td>
</tr>
<tr>
<td>Zclip</td>
<td></td>
<td>clipping distance in A (float)</td>
</tr>
<tr>
<td>backColor</td>
<td></td>
<td>background for plot as and R,G,B triplet (default = [0, 0, 0], black). (list with three atoms)</td>
</tr>
<tr>
<td>selectedAtoms</td>
<td></td>
<td>List of selected atoms (list of int values)</td>
</tr>
<tr>
<td>showRigidBodies</td>
<td></td>
<td>Flag to highlight rigid body placement</td>
</tr>
<tr>
<td>sizeH</td>
<td></td>
<td>Size ratio for H atoms (float)</td>
</tr>
<tr>
<td>bondRadius</td>
<td></td>
<td>Size of binds in A (float)</td>
</tr>
<tr>
<td>atomPtrs</td>
<td></td>
<td>positions of x, type, site sym, ADP flag in Draw Atoms (list)</td>
</tr>
<tr>
<td>viewPoint</td>
<td></td>
<td>list of lists. First item in list is [x,y,z] in fractional coordinates for the center of the plot. Second item list of previous &amp; current atom number viewed (may be [0,0])</td>
</tr>
<tr>
<td>showHydrogen</td>
<td></td>
<td>Flag to control plotting of H atoms.</td>
</tr>
<tr>
<td>unitCellBox</td>
<td></td>
<td>Flag to control display of the unit cell.</td>
</tr>
<tr>
<td>ellipseProb</td>
<td></td>
<td>Probability limit for display of thermal ellipsoids in % (float).</td>
</tr>
<tr>
<td>vdwScale</td>
<td></td>
<td>Multiplier of van der Waals radius for display of vdW spheres.</td>
</tr>
<tr>
<td>Atoms</td>
<td></td>
<td>A list of lists with an entry for each atom that is plotted.</td>
</tr>
<tr>
<td>Zstep</td>
<td></td>
<td>Step to de/increase Z-clip (float)</td>
</tr>
<tr>
<td>Quaternion</td>
<td></td>
<td>Viewing quaternion (4 element np.array)</td>
</tr>
<tr>
<td>radiusFactor</td>
<td></td>
<td>Distance ratio for searching for bonds. ? Bonds are located that are within r(Ra+Rb) and (Ra+Rb)/r where Ra and Rb are the atomic radii.</td>
</tr>
<tr>
<td>oldxy</td>
<td></td>
<td>previous view point (list with two floats)</td>
</tr>
<tr>
<td>cameraPos</td>
<td></td>
<td>Viewing position in A for plot (float)</td>
</tr>
<tr>
<td>depthFog</td>
<td></td>
<td>True if use depthFog on plot - set currently as False (bool)</td>
</tr>
</tbody>
</table>
### Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>key</th>
<th>sub-key</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBModels</td>
<td></td>
<td>Rigid body assignments (note Rigid body definitions are stored in their own main top-level tree entry.)</td>
</tr>
<tr>
<td>Pawley ref</td>
<td></td>
<td>Pawley reflections</td>
</tr>
<tr>
<td>Histograms</td>
<td></td>
<td>A dict of dicts. The key for the outer dict is the histograms tied to this phase. The inner dict contains the combined phase/histogram parameters for items such as scale factors, size and strain parameters. The following are the keys to the inner dict. (dict)</td>
</tr>
<tr>
<td>Babinet</td>
<td></td>
<td>For protein crystallography. Dictionary with two entries, ‘BabA’, ‘BabU’</td>
</tr>
<tr>
<td>Extinction</td>
<td></td>
<td>Extinction parameter (list of float, bool)</td>
</tr>
<tr>
<td>HStrain</td>
<td></td>
<td>Hydrostatic strain. List of two lists. The first is a list of the HStrain parameters (1, 2, 3, 4, or 6 depending on unit cell), the second is a list of boolean refinement parameters (same length)</td>
</tr>
<tr>
<td>Histogram</td>
<td></td>
<td>The name of the associated histogram (str)</td>
</tr>
<tr>
<td>LeBail</td>
<td></td>
<td>Flag for LeBail extraction (bool)</td>
</tr>
<tr>
<td>Mustrain</td>
<td></td>
<td>List of microstrain parameters, in order:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0. Type, one of ‘generalized’, ‘isotropic’, ‘uniaxial’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. Isotropic/uniaxial parameters - list of 3 floats</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Refinement flags - list of 3 bools</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Microstrain axis - list of 3 ints, [h, k, l]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. Generalized mustrain parameters - list of 2-6 floats, depending on space group</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5. Generalized refinement flags - list of bools, corresponding to the parameters of (4)</td>
</tr>
<tr>
<td>Pref.Ori.</td>
<td></td>
<td>Preferred Orientation. List of eight parameters. Items marked SH are only used for Spherical Harmonics.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0. Type, ‘MD’ for March-Dollase or ‘SH’ for Spherical Harmonics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. Value, float</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Refinement flag, bool</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Preferred direction, list of ints, [h, k, l]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. SH - number of terms, int</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5. SH - dict</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6. SH - list</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7. SH - float</td>
</tr>
<tr>
<td>Scale</td>
<td></td>
<td>Phase fraction, list of [float, bool].</td>
</tr>
<tr>
<td>Show</td>
<td></td>
<td>bool</td>
</tr>
<tr>
<td>Use</td>
<td></td>
<td>bool</td>
</tr>
<tr>
<td>newLeBail</td>
<td></td>
<td>Whether to perform a new LeBail extraction</td>
</tr>
<tr>
<td>MCSA</td>
<td></td>
<td>Monte-Carlo simulated annealing parameters (dict)</td>
</tr>
</tbody>
</table>

### 1.3.4 Rigid Body Objects

Rigid body descriptions are available for two types of rigid bodies: ‘Vector’ and ‘Residue’. Vector rigid bodies are developed by a sequence of translations each with a refinable magnitude and Residue rigid bodies are described as Cartesian coordinates with defined refinable torsion angles.
<table>
<thead>
<tr>
<th>key</th>
<th>sub-key</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>RBId</td>
<td>vector rigid bodies (dict of dict)</td>
</tr>
<tr>
<td></td>
<td>AtInfo</td>
<td>Drad, Color: atom drawing radius &amp; color for each atom type (dict)</td>
</tr>
<tr>
<td></td>
<td>RName</td>
<td>Name assigned by user to rigid body (str)</td>
</tr>
<tr>
<td></td>
<td>VectMag</td>
<td>vector magnitudes in Å (list)</td>
</tr>
<tr>
<td></td>
<td>rbXYZ</td>
<td>Cartesian coordinates for Vector rigid body (list of 3 float)</td>
</tr>
<tr>
<td></td>
<td>rbRef</td>
<td>3 assigned reference atom nos. in rigid body for origin definition, use</td>
</tr>
<tr>
<td></td>
<td></td>
<td>center of atoms flag (list of 3 int &amp; 1 bool)</td>
</tr>
<tr>
<td></td>
<td>VectRef</td>
<td>refinement flags for VectMag values (list of bool)</td>
</tr>
<tr>
<td></td>
<td>rbTypes</td>
<td>Atom types for each atom in rigid body (list of str)</td>
</tr>
<tr>
<td></td>
<td>rbVect</td>
<td>Cartesian vectors for each translation used to build rigid body (list of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lists)</td>
</tr>
<tr>
<td></td>
<td>useCount</td>
<td>Number of times rigid body is used in any structure (int)</td>
</tr>
<tr>
<td>Residue</td>
<td>RBId</td>
<td>residue rigid bodies (dict of dict)</td>
</tr>
<tr>
<td></td>
<td>AtInfo</td>
<td>Drad, Color: atom drawing radius &amp; color for each atom type (dict)</td>
</tr>
<tr>
<td></td>
<td>RName</td>
<td>Name assigned by user to rigid body (str)</td>
</tr>
<tr>
<td></td>
<td>rbXYZ</td>
<td>Cartesian coordinates for Residue rigid body (list of 3 float)</td>
</tr>
<tr>
<td></td>
<td>rbTypes</td>
<td>Atom types for each atom in rigid body (list of str)</td>
</tr>
<tr>
<td></td>
<td>atNames</td>
<td>Names of each atom in rigid body (e.g. C1,N2…) (list of str)</td>
</tr>
<tr>
<td></td>
<td>rbRef</td>
<td>3 assigned reference atom nos. in rigid body for origin definition, use</td>
</tr>
<tr>
<td></td>
<td></td>
<td>center of atoms flag (list of 3 int &amp; 1 bool)</td>
</tr>
<tr>
<td></td>
<td>rbSeq</td>
<td>Orig,Piv,angle,Riding (list): definition of internal rigid body torsion;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>origin atom (int), pivot atom (int), torsion angle (float), riding atoms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(list of int)</td>
</tr>
<tr>
<td></td>
<td>SelSeq</td>
<td>[int,int] used by SeqSizer to identify objects</td>
</tr>
<tr>
<td></td>
<td>useCount</td>
<td>Number of times rigid body is used in any structure (int)</td>
</tr>
<tr>
<td>RBIds</td>
<td></td>
<td>unique Ids generated upon creation of each rigid body (dict)</td>
</tr>
<tr>
<td>Vector</td>
<td></td>
<td>Ids for each Vector rigid body (list)</td>
</tr>
<tr>
<td>Residue</td>
<td></td>
<td>Ids for each Residue rigid body (list)</td>
</tr>
</tbody>
</table>

### 1.3.5 Space Group Objects

Space groups are interpreted by `GSASIIspc.SpcGroup()` and the information is placed in a SGdata object which is a dict with these keys:
<table>
<thead>
<tr>
<th>key</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpGrp</td>
<td>space group symbol (str)</td>
</tr>
<tr>
<td>Laue</td>
<td>one of the following 14 Laue classes: -1, 2/m, mmm, 4/m, 4/mmm, 3R, 3mR, 3, 3m1, 31m, 6/m, 6/mmm, m3, m3m (str)</td>
</tr>
<tr>
<td>SGInv</td>
<td>True if centrosymmetric. False if not (bool)</td>
</tr>
<tr>
<td>SGLatt</td>
<td>Lattice centering type. Will be one of P, A, B, C, I, F, R (str)</td>
</tr>
<tr>
<td>SGUniq</td>
<td>unique axis if monoclinic. Will be a, b, or c for monoclinic space groups. Will be blank for non-monoclinic. (str)</td>
</tr>
<tr>
<td>SGCen</td>
<td>Symmetry cell centering vectors. A (n,3) np.array of centers. Will always have at least one row: np.array([[0, 0, 0]])</td>
</tr>
<tr>
<td>SGOps</td>
<td>symmetry operations as a list of form ([M1, T1], [M2, T2], \ldots) where (M_n) is a 3x3 np.array and (T_n) is a length 3 np.array. Atom coordinates are transformed where the Asymmetric unit coordinates ([X is (x,y,z)]) are transformed using (X' = M_n \ast X + T_n)</td>
</tr>
<tr>
<td>SGPolax</td>
<td>Axes for space group polarity. Will be one of ‘x’, ‘y’, ‘x y’, ‘z’, ‘x z’, ‘y z’, ‘xyz’. In the case where axes are arbitrary ’111’ is used (P 1, and ?).</td>
</tr>
</tbody>
</table>

Superspace groups \([3+1]\) are interpreted by \(\text{GSASIIspc.SSpGrp()}\) and the information is placed in a SSGdata object which is a dict with these keys:

<table>
<thead>
<tr>
<th>key</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSpGrp</td>
<td>superspace group symbol extension to space group symbol, accidental spaces removed</td>
</tr>
<tr>
<td>SSGCen</td>
<td>4D cell centering vectors ([0,0,0,0]) at least</td>
</tr>
<tr>
<td>SSGOps</td>
<td>4D symmetry operations as ([M,T]) so that (M \ast x + T = x')</td>
</tr>
</tbody>
</table>

### 1.3.6 Atom Records

If \(\text{phasedict}\) points to the phase information in the data tree, then atoms are contained in a list of atom records (list) in \(\text{phasedict['Atoms']}\). Also needed to read atom information are four pointers, \(\text{cx,ct,cs,cia = phasedict['General']}['\text{AtomPtrs}']\), which define locations in the atom record, as shown below. Items shown are always present; additional ones for macromolecular phases are marked ‘mm’
1.3.7 Drawing Atom Records

If `phasedict` points to the phase information in the data tree, then drawing atoms are contained in a list of drawing atom records (list) in `phasedict['Drawing']['Atoms']`. Also needed to read atom information are four pointers, `cx,ct,cs,ci = phasedict['Drawing']['AtomPtrs']`, which define locations in the atom record, as shown below. Items shown are always present; additional ones for macromolecular phases are marked ‘mm’.

<table>
<thead>
<tr>
<th>location</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ct-4</td>
<td>mm - residue number (str)</td>
</tr>
<tr>
<td>ct-3</td>
<td>mm - residue name (e.g. ALA) (str)</td>
</tr>
<tr>
<td>ct-2</td>
<td>mm - chain label (str)</td>
</tr>
<tr>
<td>ct-1</td>
<td>atom label (str)</td>
</tr>
<tr>
<td>ct</td>
<td>atom type (str)</td>
</tr>
<tr>
<td>cx,cx+1,cx+2</td>
<td>the x,y and z coordinates (3 floats)</td>
</tr>
<tr>
<td>cs-1</td>
<td>Sym Op symbol; sym. op number + unit cell id (e.g. ‘1,0,-1’) (str)</td>
</tr>
<tr>
<td>cs</td>
<td>atom drawing style; e.g. ‘balls &amp; sticks’ (str)</td>
</tr>
<tr>
<td>cs+1</td>
<td>atom label style (e.g. ‘name’) (str)</td>
</tr>
<tr>
<td>cs+2</td>
<td>atom color (RGB triplet) (int)</td>
</tr>
<tr>
<td>cs+3</td>
<td>ADP flag: Isotropic (‘I’) or Anisotropic (‘A’)</td>
</tr>
<tr>
<td>cs+4</td>
<td>Uiso (float)</td>
</tr>
<tr>
<td>cs+5...cs+11</td>
<td>U11, U22, U33, U12, U13, U23 (6 floats)</td>
</tr>
<tr>
<td>ci</td>
<td>unique atom identifier; matches source atom Id in Atom Records (int)</td>
</tr>
</tbody>
</table>

1.3.8 Powder Diffraction Tree Items

Every powder diffraction histogram is stored in the GSAS-II data tree with a top-level entry named beginning with the string “PWDR “. The diffraction data for that information are directly associated with that tree item and there are a series of children to that item. The routines `GSASIIdataGUI.GSASII.GetUsedHistogramsAndPhasesfromTree()` and `GSASIIstrIO.GetUsedHistogramsAndPhases()` will load this information into a dictionary where the child tree name is used as a key, and the information in the main entry is assigned a key of Data, as outlined below.

<table>
<thead>
<tr>
<th>key</th>
<th>sub-key</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comments</td>
<td></td>
<td>Text strings extracted from the original powder data header. These cannot be changed by the user; it may be empty.</td>
</tr>
<tr>
<td>Limits</td>
<td></td>
<td>A list of two two element lists, as <code>[[Ld,Hd],[L,H]]</code> where L and Ld are the current and default lowest two-theta value to be used and where H and Hd are the current and default highest two-theta value to be used.</td>
</tr>
<tr>
<td>Reflection Lists</td>
<td></td>
<td>A dict with an entry for each phase in the histogram. The contents of each dict item is a dict containing reflections, as described in the Powder Reflections description.</td>
</tr>
<tr>
<td>Instrument Parameters</td>
<td></td>
<td>A list containing two dicts where the possible keys in each dict are listed below. The value for each item is a list containing three values: the initial value, the current value and a refinement flag which can have a value of True, False or 0 where 0 indicates a value that cannot be refined. The first and second values are floats unless otherwise noted. Items in the first dict are noted as [1]</td>
</tr>
</tbody>
</table>

| Lam      |         | Specifies a wavelength in Angstroms [1] |

Continued on next page
<table>
<thead>
<tr>
<th>key</th>
<th>sub-key</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lam1</td>
<td></td>
<td>Specifies the primary wavelength in Angstrom, when an alpha1, alpha2 source is used [1]</td>
</tr>
<tr>
<td>Lam2</td>
<td></td>
<td>Specifies the secondary wavelength in Angstrom, when an alpha1, alpha2 source is used [1]</td>
</tr>
<tr>
<td>I(L2)/I(L1)</td>
<td></td>
<td>Ratio of Lam2 to Lam1 [1]</td>
</tr>
<tr>
<td>Type</td>
<td></td>
<td><strong>Histogram type (str) [1]</strong>:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ‘PXC’ for constant wavelength x-ray</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ‘PNC’ for constant wavelength neutron</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ‘PNT’ for time of flight neutron</td>
</tr>
<tr>
<td>Zero</td>
<td></td>
<td>Two-theta zero correction in degrees [1]</td>
</tr>
<tr>
<td>Azimuth</td>
<td></td>
<td>Azimuthal setting angle for data recorded with differing setting angles [1]</td>
</tr>
<tr>
<td>U, V, W</td>
<td></td>
<td>Cagliotti profile coefficients for Gaussian instrumental broadening, where the FWHM goes as $U \tan^{2} \theta + V \tan \theta + W$ [1]</td>
</tr>
<tr>
<td>X, Y</td>
<td></td>
<td>Cauchy (Lorentzian) instrumental broadening coefficients [1]</td>
</tr>
<tr>
<td>SH/L</td>
<td></td>
<td>Variant of the Finger-Cox-Jephcoat asymmetric peak broadening ratio. Note that this is the average between S/L and H/L where S is sample height, H is the slit height and L is the goniometer diameter. [1]</td>
</tr>
<tr>
<td>Polariz.</td>
<td></td>
<td>Polarization coefficient. [1]</td>
</tr>
<tr>
<td>wtFactor</td>
<td></td>
<td>A weighting factor to increase or decrease the leverage of data in the histogram (float). A value of 1.0 weights the data with their standard uncertainties and a larger value increases the weighting of the data (equivalent to decreasing the uncertainties).</td>
</tr>
<tr>
<td>Sample Parameters</td>
<td></td>
<td>Specifies a dict with parameters that describe how the data were collected, as listed below. Refinable parameters are a list containing a float and a bool, where the second value specifies if the value is refined, otherwise the value is a float unless otherwise noted.</td>
</tr>
<tr>
<td>Scale</td>
<td></td>
<td>The histogram scale factor (refinable)</td>
</tr>
<tr>
<td>Absorption</td>
<td></td>
<td>The sample absorption coefficient as $\mu r$ where r is the radius (refinable). Only valid for Debye-Scherrer geometry.</td>
</tr>
<tr>
<td>SurfaceRoughA</td>
<td></td>
<td>Surface roughness parameter A as defined by Surotti,J. Appl. Cryst, 5,325-331, 1972.(refinable - only valid for Bragg-Brentano geometry)</td>
</tr>
<tr>
<td>SurfaceRoughB</td>
<td></td>
<td>Surface roughness parameter B (refinable - only valid for Bragg-Brentano geometry)</td>
</tr>
<tr>
<td>DisplaceX, DisplaceY</td>
<td></td>
<td>Sample displacement from goniometer center where Y is along the beam direction and X is perpendicular. Units are $\mu m$ (refinable).</td>
</tr>
<tr>
<td>Phi, Chi, Omega</td>
<td></td>
<td>Goniometer sample setting angles, in degrees.</td>
</tr>
<tr>
<td>Gonio. radius</td>
<td></td>
<td>Radius of the diffractometer in mm</td>
</tr>
<tr>
<td>InstrName</td>
<td></td>
<td>A name for the instrument, used in preparing a CIF (str).</td>
</tr>
<tr>
<td>Force, Temperature, Humidity, Pressure, Voltage</td>
<td></td>
<td>Variables that describe how the measurement was performed. Not used directly in any computations.</td>
</tr>
<tr>
<td>ranId</td>
<td></td>
<td>The random-number Id for the histogram (same value as where top-level key is ranId)</td>
</tr>
<tr>
<td>Type</td>
<td></td>
<td>Type of diffraction data, may be ‘Debye-Scherrer’ or ‘Bragg-Brentano’ (str).</td>
</tr>
<tr>
<td>Diffuse</td>
<td></td>
<td>not in use?</td>
</tr>
<tr>
<td>hId</td>
<td></td>
<td>The number assigned to the histogram when the project is loaded or edited (can change)</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>key</th>
<th>sub-key</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ranId</td>
<td></td>
<td>A random number id for the histogram that does not change</td>
</tr>
<tr>
<td>Background</td>
<td></td>
<td>The background is stored as a list with where the first item in the list</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and the second item is a dict. The list contains the background</td>
</tr>
<tr>
<td></td>
<td></td>
<td>function and its coefficients; the dict contains Debye diffuse terms and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>background peaks. (TODO: this needs to be expanded.)</td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td>The data consist of a list of 6 np.arrays containing in order:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0. the x-positions (two-theta in degrees),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. the intensity values (Yobs),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. the weights for each Yobs value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. the computed intensity values (Ycalc)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. the background values</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5. Yobs-Ycalc</td>
</tr>
</tbody>
</table>

### 1.3.9 Powder Reflection Data Structure

For every phase in a histogram, the **Reflection Lists** value is a dict one element of which is `RefList`, which is a np.array containing reflections. The columns in that array are documented below.

<table>
<thead>
<tr>
<th>index</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1,2</td>
<td>h,k,l (float)</td>
</tr>
<tr>
<td>3</td>
<td>multiplicity</td>
</tr>
<tr>
<td>4</td>
<td>d-space, Angstrom</td>
</tr>
<tr>
<td>5</td>
<td>pos, two-theta</td>
</tr>
<tr>
<td>6</td>
<td>sig, Gaussian width</td>
</tr>
<tr>
<td>7</td>
<td>gam, Lorenzian width</td>
</tr>
<tr>
<td>8</td>
<td>$F_2^{obs}$</td>
</tr>
<tr>
<td>9</td>
<td>$F_2^{calc}$</td>
</tr>
<tr>
<td>10</td>
<td>reflection phase, in degrees</td>
</tr>
<tr>
<td>11</td>
<td>intensity correction for reflection, this times $F_2^{obs}$ or $F_2^{calc}$ gives Iobs or Icalc</td>
</tr>
</tbody>
</table>

### 1.3.10 Single Crystal Tree Items

Every single crystal diffraction histogram is stored in the GSAS-II data tree with a top-level entry named beginning with the string “HKLF “. The diffraction data for that information are directly associated with that tree item and there are a series of children to that item. The routines `GSASIIdataGUI.GSASII.GetUsedHistogramsAndPhasesfromTree()` and `GSASIIfstrIO.GetUsedHistogramsAndPhases()` will load this information into a dictionary where the child tree name is used as a key, and the information in the main entry is assigned a key of `Data`, as outlined below.
### 1.3.11 Single Crystal Reflection Data Structure

For every single crystal a histogram, the 'Data' item contains the structure factors as an np.array in item 'RefList'. The columns in that array are documented below.

<table>
<thead>
<tr>
<th>index</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1,2</td>
<td>h,k,l (float)</td>
</tr>
<tr>
<td>3</td>
<td>multiplicity</td>
</tr>
<tr>
<td>4</td>
<td>d-space, Angstrom</td>
</tr>
<tr>
<td>5</td>
<td>$F_{o}^2$</td>
</tr>
<tr>
<td>6</td>
<td>$\sigma(F_{o}^2)$</td>
</tr>
<tr>
<td>7</td>
<td>$F_{c}^2$</td>
</tr>
<tr>
<td>8</td>
<td>$F_{o}^2 T$</td>
</tr>
<tr>
<td>9</td>
<td>$F_{c}^2 T$</td>
</tr>
<tr>
<td>10</td>
<td>reflection phase, in degrees</td>
</tr>
<tr>
<td>11</td>
<td>intensity correction for reflection, this times $F_{o}^2$ or $F_{c}^2$ gives Iobs or Icalc</td>
</tr>
</tbody>
</table>

### 1.3.12 Image Data Structure

Every 2-dimensional image is stored in the GSAS-II data tree with a top-level entry named beginning with the string "IMG ". The image data are directly associated with that tree item and there are a series of children to that item. The routines `GSASIIdataGUI.GSASII.GetUsedHistogramsAndPhasesfromTree()` and `GSASIIstrIO.GetUsedHistogramsAndPhases()` will load this information into a dictionary where the child tree name is used as a key, and the information in the main entry is assigned a key of Data, as outlined below.
<table>
<thead>
<tr>
<th>key</th>
<th>sub-key</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comments</td>
<td></td>
<td>Text strings extracted from the original image data header or a metafile. These cannot be changed by the user; it may be empty.</td>
</tr>
<tr>
<td>Image Controls</td>
<td>azmthOff</td>
<td>The offset to be applied to an azimuthal value. Accomodates detector orientations other than with the detector X-axis horizontal.</td>
</tr>
<tr>
<td></td>
<td>background image</td>
<td>The name of a tree item (“IMG . . .”) that is to be subtracted during image integration multiplied by value. It must have the same size/shape as the integrated image. NB: value &lt; 0 for subtraction.</td>
</tr>
<tr>
<td></td>
<td>calibrant</td>
<td>The material used for determining the position/orientation of the image. The data is obtained from ImageCalibrants() and UserCalibrants.py (supplied by user).</td>
</tr>
<tr>
<td></td>
<td>calibdmin</td>
<td>The minimum d-spacing used during the last calibration run.</td>
</tr>
<tr>
<td></td>
<td>calibskip</td>
<td>The number of expected diffraction lines skipped during the last calibration run.</td>
</tr>
<tr>
<td></td>
<td>center</td>
<td>The ([X,Y]) point in detector coordinates (mm) where the direct beam strikes the detector plane as determined by calibration. This point does not have to be within the limits of the detector boundaries.</td>
</tr>
<tr>
<td></td>
<td>centerAzm</td>
<td>If True then the azimuth reported for the integrated slice of the image is at the center line otherwise it is at the leading edge.</td>
</tr>
<tr>
<td></td>
<td>color</td>
<td>The name of the colormap used to display the image. Default = ‘Paired’.</td>
</tr>
<tr>
<td></td>
<td>cutoff</td>
<td>The minimum value of I/Ib for a point selected in a diffraction ring for calibration calculations. See pixLimit for details as how point is found.</td>
</tr>
<tr>
<td></td>
<td>DetDepth</td>
<td>Coefficient for penetration correction to distance; accounts for diffraction ring offset at higher angles. Optionally determined by calibration.</td>
</tr>
<tr>
<td></td>
<td>DetDepthRef</td>
<td>If True then refine DetDepth during calibration/recalibration calculation.</td>
</tr>
<tr>
<td></td>
<td>distance</td>
<td>The distance (mm) from sample to detector plane.</td>
</tr>
<tr>
<td></td>
<td>ellipses</td>
<td>Each object in ellipses is a list ([center, phi, radii, color]) where center (list) is location (mm) of the ellipse center on the detector plane, phi is the rotation of the ellipse minor axis from the x-axis, and radii are the minor &amp; major radii of the ellipse. If radii[0] is negative then parameters describe a hyperbola. Color is the selected drawing color (one of ‘b’, ‘g’, ‘r’) for the ellipse/hyperbola.</td>
</tr>
<tr>
<td></td>
<td>edgemin</td>
<td>Not used; parameter in EdgeFinder code.</td>
</tr>
<tr>
<td></td>
<td>fullIntegrate</td>
<td>If True then integrate over full 360 deg azimuthal range.</td>
</tr>
<tr>
<td></td>
<td>GonioAngles</td>
<td>The ‘Omega’, ‘Chi’, ‘Phi’ goniometer angles used for this image. Required for texture calculations.</td>
</tr>
<tr>
<td></td>
<td>invert_x</td>
<td>If True display the image with the x-axis inverted.</td>
</tr>
<tr>
<td></td>
<td>invert_y</td>
<td>If True display the image with the y-axis inverted.</td>
</tr>
<tr>
<td></td>
<td>IOtth</td>
<td>The minimum and maximum 2-theta values to be used for integration.</td>
</tr>
<tr>
<td></td>
<td>LRazimuth</td>
<td>The minimum and maximum azimuth values to be used for integration.</td>
</tr>
<tr>
<td></td>
<td>Oblique</td>
<td>If True apply a detector absorption correction using the value to the intensities obtained during integration.</td>
</tr>
<tr>
<td></td>
<td>outAzimuths</td>
<td>The number of azimuth pie slices.</td>
</tr>
<tr>
<td></td>
<td>outChannels</td>
<td>The number of 2-theta steps.</td>
</tr>
<tr>
<td></td>
<td>pixelSize</td>
<td>The X,Y dimensions (microns) of each pixel.</td>
</tr>
</tbody>
</table>
### Table 3 – continued from previous page

<table>
<thead>
<tr>
<th>key</th>
<th>sub-key</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixLimit</td>
<td>(int)</td>
<td>A box in the image with (2^{*}\text{pixLimit}+1) edges is searched to find the maximum. This value (I) along with the minimum (Ib) in the box is reported by \texttt{GSASIIimage.ImageLocalMax()} and subject to cutoff in \texttt{GSASIIimage.makeRing()}. Locations are used to construct rings of points for calibration calculations.</td>
</tr>
<tr>
<td>PolaVal</td>
<td>(list:float,bool)</td>
<td>If type='SASD' and if True, apply polarization correction to intensities from integration using value.</td>
</tr>
<tr>
<td>rings</td>
<td>(list:lists)</td>
<td>Each entry is ([X,Y,\text{dsp}]) where X &amp; Y are lists of x,y coordinates around a diffraction ring with the same d-spacing (dsp)</td>
</tr>
<tr>
<td>ring</td>
<td>(list)</td>
<td>The x,y coordinates of the (&gt;5) points on an inner ring selected by the user.</td>
</tr>
<tr>
<td>Range</td>
<td>(list)</td>
<td>The minimum &amp; maximum values of the image</td>
</tr>
<tr>
<td>rotation</td>
<td>(float)</td>
<td>The angle between the x-axis and the vector about which the detector is tilted. Constrained to -180 to 180 deg.</td>
</tr>
<tr>
<td>SampleShape</td>
<td>(str)</td>
<td>Currently only ‘Cylinder’. Sample shape for Debye-Scherrer experiments; used for absorption calculations.</td>
</tr>
<tr>
<td>SampleAbs</td>
<td>(list: float,bool)</td>
<td>Value of absorption coefficient for Debye-Scherrer experiments, flag if True to cause correction to be applied.</td>
</tr>
<tr>
<td>setDefault</td>
<td>(bool)</td>
<td>If True the use the image controls values for all new images to be read. (might be removed)</td>
</tr>
<tr>
<td>setRings</td>
<td>(bool)</td>
<td>If True then display all the selected x,y ring positions (vida supra rings) used in the calibration.</td>
</tr>
<tr>
<td>showLines</td>
<td>(bool)</td>
<td>If True then display the integration limits to be used.</td>
</tr>
<tr>
<td>size</td>
<td>(list:int)</td>
<td>The number of pixels on the image x &amp; y axes</td>
</tr>
<tr>
<td>type</td>
<td>(str)</td>
<td>One of ‘PWDR’, ‘SASD’ or ‘REFL’ for powder, small angle or reflectometry data, respectively.</td>
</tr>
<tr>
<td>tilt</td>
<td>(float)</td>
<td>The angle the detector normal makes with the incident beam; range -90 to 90.</td>
</tr>
<tr>
<td>wavelength</td>
<td>(float)</td>
<td>The radiation wavelength (Angstroms) as entered by the user (or someday obtained from the image header).</td>
</tr>
<tr>
<td>Masks</td>
<td>Arcs</td>
<td>(list: lists) Each entry ([2-\text{theta}],[\text{azimuth}[0],[\text{azimuth}[1]],\text{thickness}]) describes an arc mask to be excluded from integration</td>
</tr>
<tr>
<td>Frames</td>
<td>(list:lists)</td>
<td>Each entry describes the x,y points (3 or more - mm) that describe a frame outside of which is excluded from recalibration and integration. Only one frame is allowed.</td>
</tr>
<tr>
<td>Points</td>
<td>(list:lists)</td>
<td>Each entry ([x,y,\text{radius}]) (mm) describes an excluded spot on the image to be excluded from integration.</td>
</tr>
<tr>
<td>Polygons</td>
<td>(list:lists)</td>
<td>Each entry is a list of 3+ ([x,y]) points (mm) that describe a polygon on the image to be excluded from integration.</td>
</tr>
<tr>
<td>Rings</td>
<td>(list: lists)</td>
<td>Each entry ([2-\text{theta},\text{thickness}]) describes a ring mask to be excluded from integration.</td>
</tr>
<tr>
<td>Thresholds</td>
<td>(list:[tuple,list])</td>
<td>([\text{Imin},\text{Imax}],[\text{Imin},\text{Imax}]) This gives lower and upper limits for points on the image to be included in integration. The tuple is the image intensity limits and the list are those set by the user.</td>
</tr>
<tr>
<td>Stress/Strain</td>
<td>Sample phi</td>
<td>(float)</td>
</tr>
<tr>
<td>Sample z</td>
<td>(float)</td>
<td>Sample translation from the calibration sample position (for Sample phi = 0) These will be restricted by space group symmetry; result of strain fit refinement.</td>
</tr>
<tr>
<td>Type</td>
<td>(str)</td>
<td>‘True’ or ‘Conventional’; The strain model used for the calculation.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 3 – continued from previous page

<table>
<thead>
<tr>
<th>key</th>
<th>sub-key</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>d-zero</td>
<td></td>
<td>(list:dict) Each item is for a diffraction ring on the image; all items are from the same phase and are used to determine the strain tensor. The dictionary items are: ‘Dset’: (float) True d-spacing for the diffraction ring; entered by the user. ‘Dcalc’: (float) Average calculated d-spacing determined from strain coeff. ‘Emat’: (list: float) The strain tensor elements e11, e12 &amp; e22 (e21=e12, rest are 0) ‘Esig’: (list: float) Esds for Emat from fitting. ‘pixLimit’: (int) Search range to find highest point on ring for each data point ‘cutoff’: (float) I/Ib cutoff for searching. ‘ImxyObs’: (list: lists) [[X],[Y]] observed points to be used for strain calculations. ‘ImtaObs’: (list: lists) [[d],[azm]] transformed via detector calibration from ImxyObs. ‘ImtaCalc’: (list: lists [[d],[azm]] calculated d-spacing &amp; azimuth from fit.</td>
</tr>
</tbody>
</table>

1.3.13 Parameter Dictionary

The parameter dictionary contains all of the variable parameters for the refinement. The dictionary keys are the name of the parameter (<phase>:<hist>:<name>:<atom>). It is prepared in two ways. When loaded from the tree (in GSASIIdataGUI.GSASII.MakeLSParmDict() and GSASIIIO.ExportBaseclass.loadParmDict()), the values are lists with two elements: [value, refine flag]

When loaded from the GPX file (in GSASIIstrMain.Refine() and GSASIIstrMain.SeqRefine()), the value in the dict is the actual parameter value (usually a float, but sometimes a letter or string flag value (such as I or A for iso/anisotropic).

1.3.14 Classes and routines

GSASIIobj.AtomIdLookup = {} 
    dict listing for each phase index as a str, the atom label and atom random Id, keyed by atom sequential index as a str; best to access this using LookupAtomLabel()

GSASIIobj.AtomRanIdLookup = {} 
    dict listing for each phase the atom sequential index keyed by atom random Id; best to access this using LookupAtomId()

GSASIIobj.CompileVarDesc() 
    Set the values in the variable lookup tables (reVarDesc and reVarStep). This is called in getDescr() and getVarStep() so this initialization is always done before use.

Note that keys may contain regular expressions, where ‘[xyz]’ matches ‘x’ ‘y’ or ‘z’ (equivalently ‘[x-z]’ describes this as range of values). ‘.*’ matches any string. For example:

'AUiso': 'Atomic isotropic displacement parameter',

will match variable 'p:AUiso:a'. If parentheses are used in the key, the contents of those parentheses can be used in the value, such as:

'AU(([123][123]))': 'Atomic anisotropic displacement parameter U11',

will match AU11, AU23, ... and U11, U23 etc will be displayed in the value when used.

GSASIIobj.CreatePDFitems(G2frame, PWDRTree, EList, Qlimits, numAtm=1, FltBkg=0, PDF-names=[]) 
    Create and initialize a new set of PDF tree entries

1.3. GSASIIobj: Data objects
Parameters

- **G2frame** *(Frame)* – main GSAS-II tree frame object
- **PWDRtree** *(str)* – name of PWDR to be used to create PDF item
- **ElList** *(dict)* – data structure with composition
- **Qlimits** *(list)* – Q limits to be used for computing the PDF
- **numAtm** *(float)* – no. atom in chemical formula
- **FltBkg** *(float)* – flat background value
- **PDFnames** *(list)* – previously used PDF names

Returns the Id of the newly created PDF entry

```python
GSASIIobj.DefaultControls = {'Author': 'no name', 'Copy2Next': False, 'F **2': False, 'FreePrm1': 'Sample humidity (%)', ... 'MinExt': 0.01, 'minF/sig': 0.0}, 'deriv type': 'analytic Hessian', 'max cyc': 3, 'min dM/M': 0.001, 'shift factor': 1.0}
```
Values to be used as defaults for the initial contents of the Controls data tree item.

**class GSASIIobj.ExpressionCalcObj(exprObj)***
An object used to evaluate an expression from a **ExpressionObj** object.

**Parameters exprObj** *(ExpressionObj)* – a **ExpressionObj** expression object with an expression string and mappings for the parameter labels in that object.

**EvalExpression()**
Evaluate an expression. Note that the expression and mapping are taken from the **ExpressionObj** expression object and the parameter values were specified in **SetupCalc()**. :returns: a single value for the expression. If parameter values are arrays (for example, from wild-carded variable names), the sum of the resulting expression is returned.

For example, if the expression is 'A*B', where A is 2.0 and B maps to '1::Afrac:*', which evaluates to:

```plaintext
[0.5, 1, 0.5]
```
then the result will be 4.0.

**SetupCalc (parmDict)**
Do all preparations to use the expression for computation. Adds the free parameter values to the parameter dict (parmDict).

**UpdateDict (parmDict)**
Update the dict for the expression with values in a dict :param dict parmDict: a dict of values, items not in use are ignored

**UpdateVars (varList, valList)**
Update the dict for the expression with a set of values :param list varList: a list of variable names :param list valList: a list of corresponding values

```python
compiledExpr = None
```
The expression as compiled byte-code

```python
eObj = None
```
The expression and mappings; a **ExpressionObj** object

```python
exprDict = None
```
dict that defines values for labels used in expression and packages referenced by functions

```python
fxnpkgdict = None
```
a dict with references to packages needed to find functions referenced in the expression.
lblLookup = None
Lookups table that specifies the expression label name that is tied to a particular GSAS-II parameters in the parmDict.

parmDict = None
A copy of the parameter dictionary, for distance and angle computation.

su = None
Standard error evaluation where supplied by the evaluator.

varLookup = None
Lookup table that specifies the GSAS-II variable(s) indexed by the expression label name. (Used for only for diagnostics not evaluation of expression.)

class GSASIIobj.ExpressionObj
Defines an object with a user-defined expression, to be used for secondary fits or restraints. Object is created null, but is changed using LoadExpression(). This contains only the minimum information that needs to be stored to save and load the expression and how it is mapped to GSAS-II variables.

CheckVars()
Check that the expression can be parsed, all functions are defined and that input loaded into the object is internally consistent. If not an Exception is raised.

Returns a dict with references to packages needed to find functions referenced in the expression.

EditExpression(exprVarLst, varSelect, varName, varValue, varReflag)
Load the expression and associated settings from the object into arrays used for editing.

Parameters

• exprVarLst (list) – parameter labels found in the expression
• varSelect (dict) – this will be 0 for Free parameters and non-zero for expression labels linked to G2 variables.
• varName (dict) – Defines a name (str) associated with each free parameter
• varValue (dict) – Defines a value (float) associated with each free parameter
• varReflag (dict) – Defines a refinement flag (bool) associated with each free parameter

Returns the expression as a str

GetDepVar()
return the dependent variable, or None

GetIndependentVars()
Returns the names of the required independent parameters used in expression

GetVaried()
Returns the names of the free parameters that will be refined

GetVariedVarVal()
Returns the names and values of the free parameters that will be refined

LoadExpression(expr, exprVarLst, varSelect, varName, varValue, varReflag)
Load the expression and associated settings into the object. Raises an exception if the expression is not parsed, if not all functions are defined or if not all needed parameter labels in the expression are defined.

This will not test if the variable referenced in these definitions are actually in the parameter dictionary. This is checked when the computation for the expression is done in SetupCalc().

Parameters

1.3. GSASIIobj: Data objects
• **expr** *(str)* – the expression

• **exprVarLst** *(list)* – parameter labels found in the expression

• **varSelect** *(dict)* – this will be 0 for Free parameters and non-zero for expression labels linked to G2 variables.

• **varName** *(dict)* – Defines a name *(str)* associated with each free parameter

• **varValue** *(dict)* – Defines a value *(float)* associated with each free parameter

• **varRefflag** *(dict)* – Defines a refinement flag *(bool)* associated with each free parameter

**ParseExpression**(expr)

Parse an expression and return a dict of called functions and the variables used in the expression. Returns None in case an error is encountered. If packages are referenced in functions, they are loaded and the functions are looked up into the modules global workspace.

Note that no changes are made to the object other than saving an error message, so that this can be used for testing prior to the save.

**Returns** a list of used variables

**SetDepVar**(var)

Set the dependent variable, if used

**UpdateVariedVars**(varyList, values)

Updates values for the free parameters (after a refinement); only updates refined vars

**assignVars** = None

A dict where keys are label names in the expression mapping to a GSAS-II variable. The value a G2 variable name. Note that the G2 variable name may contain a wild-card and correspond to multiple values.

**expression** = None

The expression as a text string

**freeVars** = None

A dict where keys are label names in the expression mapping to a free parameter. The value is a list with:

• a name assigned to the parameter

• a value for to the parameter and

• a flag to determine if the variable is refined.

**lastError** = None

Shows last encountered error in processing expression (list of 1-3 str values)

**exception** GSASIIobj.G2Exception *(msg)*

A generic GSAS-II exception class

**exception** GSASIIobj.G2RefineCancel *(msg)*

Raised when Cancel is pressed in a refinement dialog

**class** GSASIIobj.G2VarObj(*args)*

Defines a GSAS-II variable either using the phase/atom/histogram unique Id numbers or using a character string that specifies variables by phase/atom/histogram number (which can change). Note that LoadID() should be used to (re)load the current Ids before creating or later using the G2VarObj object.

This can store rigid body variables, but does not translate the residue # and body # to/from random Ids

A G2VarObj object can be created with a single parameter:

**Parameters** varname *(str/tuple)* –
a single value can be used to create a \texttt{G2VarObj} object. If a string, it must be of form 
“p:h:var” or “p:h:var:a”, where

- \( p \) is the phase number (which may be left blank or may be ‘*’ to indicate all phases);
- \( h \) is the histogram number (which may be left blank or may be ‘*’ to indicate all histograms);
- \( a \) is the atom number (which may be left blank in which case the third colon is omitted). The atom number can be specified as ‘*’ if a phase number is specified (not as ‘*’). For rigid body variables, specify \( a \) will be a string of form “residue:body#”

Alternately a single tuple of form (Phase, Histogram, VarName, AtomID) can be used, where Phase, Histogram, and AtomID are None or are ranId values (or one can be ‘*’) and VarName is a string. Note that if Phase is ‘*’ then the AtomID is an atom number. For a rigid body variables, AtomID is a string of form “residue:body#”. If four positional arguments are supplied, they are:

\textbf{Parameters}

- \texttt{phasenum (str/int)} – The number for the phase (or None or ‘*’)
- \texttt{histnum (str/int)} – The number for the histogram (or None or ‘*’)
- \texttt{varname (str)} – a single value can be used to create a \texttt{G2VarObj}
- \texttt{atomnum (str/int)} – The number for the atom (or None or ‘*’)

\texttt{varname ()}

Formats the GSAS-II variable name as a “traditional” GSAS-II variable string (p:h:<var>:a) or (p:h:<var>)

\textbf{Returns} the variable name as a str

\texttt{GSASIIobj.GenWildCard (varlist)}

Generate wildcard versions of G2 variables. These introduce ‘*’ for a phase, histogram or atom number (but only for one of these fields) but only when there is more than one matching variable in the input variable list. So if the input is this:

\begin{verbatim}
varlist = ['0::AUiso:0', '0::AUiso:1', '1::AUiso:0']
\end{verbatim}

then the output will be this:

\begin{verbatim}
wildList = ['*::AUiso:0', '0::AUiso:*']
\end{verbatim}

\textbf{Parameters} \texttt{varlist (list)} – an input list of GSAS-II variable names (such as 0::AUiso:0)

\textbf{Returns} wildList, the generated list of wild card variable names.

\texttt{GSASIIobj.GetPhaseNames (fl)}

Returns a list of phase names found under ‘Phases’ in GSASII gpx file NB: there is another one of these in GSASIIInstrIO.py that uses the gpx filename

\textbf{Parameters} \texttt{fl (file)} – opened .gpx file

\textbf{Returns} list of phase names

\texttt{GSASIIobj.HistIdLookup = {}}

dict listing histogram name and random Id, keyed by sequential histogram index as a str; best to access this using \texttt{LookupHistName()}
GSAS-II Developers Documentation, Release version 4249

GSASIIobj.HistRanIdLookup = {}
   dict listing histogram sequential index keyed by histogram random id; best to access this using
   LookupHistId()

GSASIIobj.HowDidIgetHere(wherecalledonly=False)
   Show a traceback with calls that brought us to the current location. Used for debugging.

class GSASIIobj.ImportBaseclass(formatName, longFormatName=None, extensionlist=[], strictExtension=False)
   Defines a base class for the reading of input files (diffraction data, coordinates,...). See Writing a Import
   Routine for an explanation on how to use a subclass of this class.

CIFValidator(filepointer)
   A ContentsValidator() for use to validate CIF files.

ContentsValidator(filename)
   This routine will attempt to determine if the file can be read with the current format. This will typically be
   overridden with a method that takes a quick scan of [some of] the file contents to do a “sanity” check if the
   file appears to match the selected format. the file must be opened here with the correct format (binary/text)

ExtensionValidator(filename)
   This methods checks if the file has the correct extension

   Returns
       • False if this filename will not be supported by this reader (only when strictExtension is
         True)
       • True if the extension matches the list supplied by the reader
       • None if the reader allows un-registered extensions

exception ImportException
   Defines an Exception that is used when an import routine hits an expected error, usually in .Reader.
   Good practice is that the Reader should define a value in self.errors that tells the user some information
   about what is wrong with their file.

ReInitialize()
   Reinitialize the Reader to initial settings

class GSASIIobj.ImportImage(formatName, longFormatName=None, extensionlist=[], strictExtension=False)
   Defines a base class for the reading of images

Images are read in only these places:

   • Initial reading is typically done from a menu item with a call to GSASIIdataGUI.GSASII.
     OnImportImage() which in turn calls GSASIIdataGUI.GSASII.OnImportGeneric(). That
     calls methods ExtensionValidator(), ContentsValidator() and Reader(). This returns a
     list of reader objects for each read image. Also used in GSASIIscriptable.import_generic().
   • Images are read alternatively in GSASIIIIO.ReadImages(), which puts image info directly into the
     data tree.
   • Images are reloaded with GSASIIIIO.GetImageData().

When reading an image, the Reader() routine in the ImportImage class should set:
   • Comments: a list of strings (str),
   • Npix: the number of pixels in the image (int),
   • Image: the actual image as a numpy array (np.array)
Data: a dict defining image parameters (dict). Within this dict the following data items are needed:

- ‘pixelSize’: size of each pixel in microns (such as [200., 200.]).
- ‘wavelength’: wavelength in Angstroms.
- ‘distance’: distance of detector from sample in cm.
- ‘center’: uncalibrated center of beam on detector (such as [204.8, 204.8]).
- ‘size’: size of image (such as [2048, 2048]).
- ‘ImageTag’: image number or other keyword used to retrieve image from a multi-image data file (defaults to 1 if not specified).
- ‘sumfile’: holds sum image file name if a sum was produced from a multi image file

Optional data items:

- repeat: set to True if there are additional images to read in the file, False otherwise
- repeatcount: set to the number of the image.

Note that the above is initialized with InitParameters(). (Also see Writing a Import Routine for an explanation on how to use import classes in general.)

InitParameters()
initialize the instrument parameters structure

LoadImage (ParentFrame, imagefile, imagetag=None)
Optionally, call this after reading in an image to load it into the tree. This saves time by preventing a reread of the same information.

ReInitialize()
Reinitialize the Reader to initial settings – not used at present

class GSASIIobj.ImportPDFData (formatName, longFormatName=None, extensionlist=[], strictExtension=False)
Defines a base class for the reading of files with PDF G(R) data. See Writing a Import Routine for an explanation on how to use this class.

ReInitialize()
Reinitialize the Reader to initial settings

class GSASIIobj.ImportPhase (formatName, longFormatName=None, extensionlist=[], strictExtension=False)
Defines a base class for the reading of files with PDF G(R) data. See Writing a Import Routine for an explanation on how to use this class.

ReInitialize()
Reinitialize the Reader to initial settings

class GSASIIobj.ImportPowderData (formatName, longFormatName=None, extensionlist=[], strictExtension=False)
Defines a base class for the reading of files with powder data.

Objects constructed that subclass this (in import/G2phase_*.py etc.) will be used in GSASIIdataGUI.
GSASII.OnImportPhase() and in GSASIIscriptable.import_generic(). See Writing a Import Routine for an explanation on how to use this class.

ReInitialize()
Reinitialize the Reader to initial settings

1.3. GSASIIobj: Data objects
class GSASIIobj.ImportReflectometryData (formatName, longFormatName=None, extensionList=[], strictExtension=False)

Defines a base class for the reading of files with reflectometry data. See Writing a Import Routine for an explanation on how to use this class.

ReInitialize ()
Reinitialize the Reader to initial settings

class GSASIIobj.ImportSmallAngleData (formatName, longFormatName=None, extensionList=[], strictExtension=False)

Defines a base class for the reading of files with small angle data. See Writing a Import Routine for an explanation on how to use this class.

ReInitialize ()
Reinitialize the Reader to initial settings

class GSASIIobj.ImportStructFactor (formatName, longFormatName=None, extensionList=[], strictExtension=False)

Defines a base class for the reading of files with tables of structure factors. Structure factors are read with a call to GSASIIdataGUI.GSASII.OnImportSfact () which in turn calls GSASIIdataGUI.GSASII.OnImportGeneric (), which calls methods ExtensionValidator (), ContentsValidator () and Reader ().

See Writing a Import Routine for an explanation on how to use import classes in general. The specifics for reading a structure factor histogram require that the Reader () routine in the import class need to do only a few things: It should load RefDict item 'RefList' with the reflection list, and set Parameters with the instrument parameters (initialized with InitParameters () and set with UpdateParameters ()).

Banks = None
self.RefDict is a dict containing the reflection information, as read from the file. Item ‘RefList’ contains the reflection information. See the Single Crystal Reflection Data Structure for the contents of each row. Dict element ‘FF’ contains the form factor values for each element type; if this entry is left as initialized (an empty list) it will be initialized as needed later.

InitParameters ()
initialize the instrument parameters structure

Parameters = None
self.Parameters is a list with two dicts for data parameter settings

ReInitialize ()
Reinitialize the Reader to initial settings

UpdateParameters (Type=None, Wave=None)
Revise the instrument parameters

GSASIIobj.IndexAllIds (Histories, Phases)
Scan through the used phases & histograms and create an index to the random numbers of phases, histograms and atoms. While doing this, confirm that assigned random numbers are unique – just in case lightning strikes twice in the same place.

Note: this code assumes that the atom random Id (ranId) is the last element each atom record.

This is called in three places (only): GSASIIStrIO.GetUsedHistogramsAndPhases () (which loads the histograms and phases from a GPX file), GetUsedHistogramsAndPhasesfromTree () (which loads the histograms and phases from the data tree.) and GSASIIconstrGUI.UpdateConstraints () (which displays & edits the constraints in a GUI)

TODO: do we need a lookup for rigid body variables?

GSASIIobj.LookupAtomId (pld, ranId)
Get the atom number from a phase and atom random Id
Parameters

- **pId** (*int/str*) – the sequential number of the phase
- **ranId** (*int*) – the random Id assigned to an atom

**Returns** the index number of the atom (str)

**GSASIIobj.LookupAtomLabel**(pId, index)

Get the atom label from a phase and atom index number

Parameters

- **pId** (*int/str*) – the sequential number of the phase
- **index** (*int*) – the index of the atom in the list of atoms

**Returns** the label for the atom (str) and the random Id of the atom (int)

**GSASIIobj.LookupHistId**(ranId)

Get the histogram number and name from a histogram random Id

Parameters **ranId** (*int*) – the random Id assigned to a histogram

**Returns** the sequential Id (hId) number for the histogram (str)

**GSASIIobj.LookupHistName**(hId)

Get the histogram number and name from a histogram Id

Parameters **hId** (*int/str*) – the sequential assigned to a histogram

**Returns** (hist, ranId) where hist is the name of the histogram (str) and ranId is the random # id for the histogram (int)

**GSASIIobj.LookupPhaseId**(ranId)

Get the phase number and name from a phase random Id

Parameters **ranId** (*int*) – the random Id assigned to a phase

**Returns** the sequential Id (pId) number for the phase (str)

**GSASIIobj.LookupPhaseName**(pId)

Get the phase number and name from a phase Id

Parameters **pId** (*int/str*) – the sequential assigned to a phase

**Returns** (phase, ranId) where phase is the name of the phase (str) and ranId is the random # id for the phase (int)

**GSASIIobj.LookupWildCard**(varname, varlist)

returns a list of variable names from list varname that match wildcard name in varname

Parameters

- **varname** (*str*) – a G2 variable name containing a wildcard (such as *::var)
- **varlist** (*list*) – the list of all variable names used in the current project

**Returns** a list of matching GSAS-II variables (may be empty)

**GSASIIobj.MakeUniqueLabel**(lbl, labellist)

Make sure that every a label is unique against a list by adding digits at the end until it is not found in list.

Parameters

- **lbl** (*str*) – the input label
- **labellist** (*list*) – the labels that have already been encountered

1.3. **GSASIIobj: Data objects**
Returns  lbl if not found in labellist or lbl with _1-9 (or _10-99, etc.) appended at the end

GSASIIobj.PhaseIdLookup = {}
dict listing phase name and random Id keyed by sequential phase index as a str; best to access this using LookupPhaseName()

GSASIIobj.PhaseRanIdLookup = {}
dict listing phase sequential index keyed by phase random Id; best to access this using LookupPhaseId()

GSASIIobj.ReadCIF (URLorFile)
Open a CIF, which may be specified as a file name or as a URL using PyCifRW (from James Hester). The open routine gets confused with DOS names that begin with a letter and colon “C:dir” so this routine will try to open the passed name as a file and if that fails, try it as a URL

Parameters  URLorFile (str) – string containing a URL or a file name. Code will try first to open it as a file and then as a URL.

Returns  a PyCifRW CIF object.

GSASIIobj.SetDefaultSample ()
Fills in default items for the Sample dictionary for Debye-Scherrer & SASD

GSASIIobj.SetNewPhase (Name='New Phase', SGData=None, cell=None, Super=None)
Create a new phase dict with default values for various parameters

Parameters
• Name (str) – Name for new Phase
• SGData (dict) – space group data from GSASIIspc:SpcGroup(); defaults to data for P 1
• cell (list) – unit cell parameter list; defaults to [1.0,1.0,1.0,90.,90,90.,1.]

GSASIIobj.ShortHistNames = {}
a dict containing a possibly shortened and when non-unique numbered version of the histogram name. Keyed by the histogram sequential index.

GSASIIobj.ShortPhaseNames = {}
a dict containing a possibly shortened and when non-unique numbered version of the phase name. Keyed by the phase sequential index.

class GSASIIobj.ShowTiming
An object to use for timing repeated sections of code.

Create the object with::

tim0 = ShowTiming()

Tag sections of code to be timed with::
tim0.start('start') tim0.start('in section 1') tim0.start('in section 2') etc. (Note that each section should have a unique label.)

After the last section, end timing with::
tim0.end()

Show timing results with::
tim0.show()

GSASIIobj.SortVariables (varlist)
Sorts variable names in a sensible manner

GSASIIobj.StripUnicode (string, subs=’.’)
Strip non-ASCII characters from strings

Parameters
• string (str) – string to strip Unicode characters from
• **subs** *(str)* – character(s) to place into string in place of each Unicode character. Defaults to ‘‘.

**Returns** a new string with only ASCII characters

GSASIIobj.*VarDescr*(varname)

Return two strings with a more complete description for a GSAS-II variable

**Parameters** name *(str)* – A full G2 variable name with 2 or 3 or 4 colons (<p>:<h>:name[:<a>]
or <p>::RBname:<r>:<t>))

**Returns** (loc, meaning) where loc describes what item the variable is mapped (phase, histogram, etc.) and meaning describes what the variable does.

GSASIIobj.*fmtVarDescr*(varname)

Return a string with a more complete description for a GSAS-II variable

**Parameters** varname *(str)* – A full G2 variable name with 2 or 3 or 4 colons (<p>:<h>:name[:<a>]
or <p>::RBname:<r>:<t>))

**Returns** a string with the description

GSASIIobj.*getDescr*(name)

Return a short description for a GSAS-II variable

**Parameters** name *(str)* – The descriptive part of the variable name without colons (:)

**Returns** a short description or None if not found

GSASIIobj.*getVarDescr*(varname)

Return a short description for a GSAS-II variable

**Parameters** name *(str)* – A full G2 variable name with 2 or 3 or 4 colons (<p>:<h>:name[:<a1>[:<a2>])

**Returns** a six element list as [p, ’h’, ’name’, ’a1’, ’a2’, ’description’], where p, h, a1, a2 are str values or None, for the phase number, the histogram number and the atom number; name will always be a str; and description is str or None. If the variable name is incorrectly formed (for example, wrong number of colons), None is returned instead of a list.

GSASIIobj.*getVarStep*(name, parmDict=None)

Return a step size for computing the derivative of a GSAS-II variable

**Parameters**

- **name** *(str)* – A complete variable name (with colons, :)
- **parmDict** *(dict)* – A dict with parameter values or None (default)

**Returns** a float that should be an appropriate step size, either from the value supplied in CompileVarDesc() or based on the value for name in parmDict, if supplied. If not found or the value is zero, a default value of 1e-5 is used. If parmDict is None (default) and no value is provided in CompileVarDesc(), then None is returned.

GSASIIobj.*reVarDesc* = {}

This dictionary lists descriptions for GSAS-II variables where keys are compiled regular expressions that will match the name portion of a parameter name. Initialized in CompileVarDesc().

GSASIIobj.*reVarStep* = {}

This dictionary lists the preferred step size for numerical derivative computation w/r to a GSAS-II variable. Keys are compiled regular expressions and values are the step size for that parameter. Initialized in CompileVarDesc().

1.3. **GSASIIobj: Data objects**
GSAS-II Developers Documentation, Release version 4249

GSASIIobj.removeNonRefined(parmList)
Remove items from variable list that are not refined and should not appear as options for constraints

Parameters parmList (list) – a list of strings of form “p:h:VAR:a” where VAR is the variable name

Returns a list after removing variables where VAR matches a entry in local variable NonRefinedList

1.4 GSAS-II Utility Modules

1.4.1 GSASIIpath: locations & updates

Routines for dealing with file locations, etc.

Determines the location of the compiled (.pyd or .so) libraries.

Interfaces with subversion (svn): Determine the subversion release number by determining the highest version number where SetVersionNumber() is called (best done in every GSASII file). Other routines will update GSASII from the subversion server if svn can be found.

Accesses configuration options, as defined in config.py

GSASIIpath.DoNothing()
A routine that does nothing. This is called in place of IPyBreak and pdbBreak except when the debug option is set True in config.py

GSASIIpath.DownloadG2Binaries(g2home, verbose=True)
Download GSAS-II binaries from appropriate section of the GSAS-II svn repository based on the platform, numpy and Python version

GSASIIpath.GetConfigValue(key, default=None)
Return the configuration file value for key or a default value if not present

Parameters

• key (str) – a value to be found in the configuration (config.py) file
• default – a value to be supplied is none is in the config file or the config file is not found.
  Defaults to None

Returns the value found or the default.

GSASIIpath.GetVersionNumber()
Return the maximum version number seen in SetVersionNumber()

GSASIIpath.IPyBreak()
A routine that does nothing. This is called in place of IPyBreak and pdbBreak except when the debug option is set True in config.py

GSASIIpath.IPyBreak_base(userMsg=None)
A routine that invokes an IPython session at the calling location This routine is only used when debug=True is set in config.py

GSASIIpath.InvokeDebugOpts()
Called in GSASII.py to set up debug options

GSASIIpath.LoadConfigFile(filename)
Read a GSAS-II configuration file. Comments (starting with “%”) are removed, as are empty lines

Parameters filename (str) – base file name (such as ‘file.dat’). Files with this name are located from the path and the contents of each are concatenated.
Returns a list containing each non-empty (after removal of comments) line found in every matching config file.

```python
GSASIIpath.MacStartGSASII(g2script, project="")
```
Start a new instance of GSAS-II by opening a new terminal window and starting a new GSAS-II process. Used on Mac OS X only.

Parameters

- **g2script** (`str`) – file name for the GSASII.py script
- **project** (`str`) – GSAS-II project (.gpx) file to be opened, default is blank which opens a new project

```python
GSASIIpath.MakeByte2str(arg)
```
Convert output from subprocess pipes (bytes) to str (unicode) in Python 3. In Python 2: Leaves output alone (already str). Leaves stuff of other types alone (including unicode in Py2) Works recursively for string-like stuff in nested loops and tuples.

```python
out = MakeByte2str(out)
```
or:

```python
out, err = MakeByte2str(s.communicate())
```

```python
GSASIIpath.SetBinaryPath(printInfo=False, loadBinary=True)
```
Add location of GSAS-II shared libraries (binaries: .so or .pyd files) to path

This routine must be executed after GSASIIpath is imported and before any other GSAS-II imports are done.

```python
GSASIIpath.SetConfigValue(parmdict)
```
Set configuration variables from a dictionary where elements are lists First item in list is the default value and second is the value to use.

```python
GSASIIpath.SetVersionNumber(RevString)
```
Set the subversion version number

**Parameters**

- **RevString** (`str`) – something like “$Revision: 4213 $” that is set by subversion when the file is retrieved from subversion.

```python
Place GSASIIpath.SetVersionNumber("$Revision: 4213 $") in every python file.
```

```python
GSASIIpath.TestSPG(fpth)
```
Test if pyspg.[so,.pyd] can be run from a location in the path

```python
GSASIIpath.addPrevGPX(fil, configDict)
```
Add a GPX file to the list of previous files. Move previous names to start of list. Keep most recent five files

```python
GSASIIpath.exceptHook(*args)
```
A routine to be called when an exception occurs. It prints the traceback with fancy formatting and then calls an IPython shell with the environment of the exception location.

This routine is only used when debug=True is set in config.py

```python
GSASIIpath.g2home = 'https://subversion.xray.aps.anl.gov/pyGSAS'
```
Define the location of the GSAS-II subversion repository

```python
GSASIIpath.getsvnProxy()
```
Loads a proxy for subversion from the file created by bootstrap.py
GSASIIpath.pdbBreak()
A routine that does nothing. This is called in place of IPyBreak and pdbBreak except when the debug option is set True in config.py

GSASIIpath.proxycmds = []
Used to hold proxy information for subversion, set if needed in whichsvn

GSASIIpath.setsvnProxy(host, port)
Sets the svn commands needed to use a proxy

GSASIIpath.svnCleanup(fpath='/home/docs/checkouts/readthedocs.org/user_builds/gsas-ii/checkouts/latest', verbose=True)
This runs svn cleanup on a selected local directory.

Parameters fpath (str) – path to repository dictionary, defaults to directory where the current file is located

Returns a list of files that were changed locally. If no files are changed, the list has length 0

Parameters fpath – path to repository dictionary, defaults to directory where the current file is located

Returns None if there is a subversion error (likely because the path is not a repository or svn is not found)

GSASIIpath.svnGetLog(fpath='/home/docs/checkouts/readthedocs.org/user_builds/gsas-ii/checkouts/latest', version=None)
Get the revision log information for a specific version of the specified package

Parameters
• fpath (str) – path to repository dictionary, defaults to directory where the current file is located.
• version (int) – the version number to be looked up or None (default) for the latest version.

Returns a dictionary with keys (one hopes) ‘author’, ‘date’, ‘msg’, and ‘revision’

GSASIIpath.svnGetRev(fpath='/home/docs/checkouts/readthedocs.org/user_builds/gsas-ii/checkouts/latest', local=True)
Obtain the version number for the either the last update of the local version or contacts the subversion server to get the latest update version (# of Head).

Parameters
• fpath (str) – path to repository dictionary, defaults to directory where the current file is located
• local (bool) – determines the type of version number, where True (default): returns the latest installed update version False: returns the version number of Head on the server

Returns the version number as an int or None if there is a subversion error (likely because the path is not a repository or svn is not found). The error message is placed in global variable svnLastError

GSASIIpath.svnInstallDir(URL, loadpath)
Load a subversion tree into a specified directory

Parameters
• URL (str) – the repository URL
• **loadpath** *(str)* – path to locate files

GSASIIpath.svnList *(URL, verbose=True)*

Get a list of subdirectories from and svn repository

GSASIIpath.svnLocCache = None

Cached location of svn to avoid multiple searches for it

GSASIIpath.svnSwitch2branch *(branch=None, loc=None, svnHome=None)*

Switch to a subversion branch if specified. Switches to trunk otherwise.

GSASIIpath.svnSwitchDir *(rpath, filename, baseURL, loadpath=None, verbose=True)*

This performs a switch command to move files between subversion trees. Note that if the files were previously downloaded, the switch command will update the files to the newest version.

**Parameters**

- **rpath** *(str)* – path to locate files, relative to the GSAS-II installation path (defaults to path2GSAS2)
- **URL** *(str)* – the repository URL
- **loadpath** *(str)* – the prefix for the path, if specified. Defaults to path2GSAS2
- **verbose** *(bool)* – if True (default) diagnostics are printed

GSASIIpath.svnUpdateDir *(fpath='/home/docs/checkouts/readthedocs.org/user_builds/gsas-ii/checkouts/latest', version=None, verbose=True)*

This performs an update of the files in a local directory from a server.

**Parameters**

- **fpath** *(str)* – path to repository dictionary, defaults to directory where the current file is located
- **version** – the number of the version to be loaded. Used only cast as a string, but should be an integer or something that corresponds to a string representation of an integer value when cast. A value of None (default) causes the latest version on the server to be used.

GSASIIpath.svnUpdateProcess *(version=None, projectfile=None, branch=None)*

perform an update of GSAS-II in a separate python process

GSASIIpath.svnUpgrade *(fpath='/home/docs/checkouts/readthedocs.org/user_builds/gsas-ii/checkouts/latest')*

This reformats subversion files, which may be needed if an upgrade of subversion is done.

**Parameters**

- **fpath** *(str)* – path to repository dictionary, defaults to directory where the current file is located

GSASIIpath.svnVersion *(svn=None)*

Get the version number of the current subversion executable

**Returns** a string with a version number such as “1.6.6” or None if subversion is not found.

GSASIIpath.svnVersionNumber *(svn=None)*

Get the version number of the current subversion executable

**Returns** a fractional version number such as 1.6 or None if subversion is not found.

GSASIIpath.whichsvn()

Returns a path to the subversion exe file, if any is found. Searches the current path after adding likely places where GSAS-II might install svn.

**Returns** None if svn is not found or an absolute path to the subversion executable file.
1.4.2 GSASIIlog: Logging of “Actions”

Module to provide logging services, e.g. track and replay “actions” such as menu item, tree item, button press, value change and so on.

GSASIIlog.ButtonBindingLookup = {}
Lookup table for button objects

class GSASIIlog.ButtonLogEntry(locationcode, label)
Object to track button press

GSASIIlog.G2logList = [None]
Contains a list of logged actions; first item is ignored

GSASIIlog.InvokeMenuCommand(id, G2frame, event)
Called when a menu item is used to log the action as well as call the routine “bind”ed to that menu item

class GSASIIlog.LogEntry
Base class to define logging objects. These store information on events in a manner that can be pickled and saved – direct references to wx objects is not allowed.

Each object must define:

• __init__: stores the information needed to log & later recreate the action
• __str__: shows a nice ASCII string for each action
• Replay: recreates the action when the log is played

optional:

• Repaint: redisplays the current window

GSASIIlog.LogInfo = {'LastPaintAction': None, 'Logging': False, 'Tree': None}
Contains values that are needed in the module for past actions & object location

GSASIIlog.LogOff()
Turn Off logging of actions

GSASIIlog.LogOn()
Turn On logging of actions

GSASIIlog.LogVarChange(result, key)
Called when a variable is changed to log that action

GSASIIlog.MakeButtonLog(locationcode, label)
Create a ButtonLogEntry action log

GSASIIlog.MakeTabLog(title, tabname)
Create a TabLogEntry action log

GSASIIlog.MakeTreeLog(textlist)
Create a TreeLogEntry action log

GSASIIlog.MenuBindingLookup = {}
Lookup table for Menu buttons

class GSASIIlog.MenuLogEntry(menuLabelList)
object that tracks when a menu command is executed

    Replay()
    Perform a Menu item action when read from the log

GSASIIlog.OnReplayPress(event)
execute one or more commands when the replay button is pressed
GSASIIlog.ReplayLog(event)
    replay the logged actions

GSASIIlog.SaveMenuCommand(id, G2frame, handler)
    Creates a table of menu items and their pseudo-bindings

GSASIIlog.ShowLogStatus()
    Return the logging status

class GSASIIlog.TabLogEntry(title, tabname)
    Object to track when tabs are pressed in the DataFrame window

    Repaint()
        Used to redraw a window created in response to a Tab press

    Replay()
        Perform a Tab press action when read from the log

class GSASIIlog.TreeLogEntry(itemlist)
    Object to track when tree items are pressed in the main window

    Repaint()
        Used to redraw a window created in response to a click on a data tree item

    Replay()
        Perform a Tree press action when read from the log

class GSASIIlog.VarLogEntry(treeRefs, indexRefs, value)
    object that tracks changes to a variable

    Replay()
        Perform a Variable Change action, when read from the log

class GSASIIlog.dictLogged(obj, treeRefs, indexRefs=[])  
    A version of a dict object that tracks the source of the object back to the location on the G2 tree. If a list (tuple)  
    or dict are pulled from inside this object the source information is appended to the provenance tracking lists.  
    tuples are converted to lists.

class GSASIIlog.listLogged(obj, treeRefs, indexRefs=[])  
    A version of a list object that tracks the source of the object back to the location on the G2 tree. If a list (tuple)  
    or dict are pulled from inside this object the source information is appended to the provenance tracking lists.  
    tuples are converted to lists.

1.4.3 config_example.py: Configuration options

This file contains optional configuration options for GSAS-II. The variables in this file can be copied to file config.py,  
which is imported if present. Access these variables using GSASIIpath.GetConfigValue(), which returns  
None if the variable is not set. Note that a config.py file need not be present, but if in use it will typically be found with  
the GSAS-II source directory (GSASIIpath.Path2GSAS) or a directory for local GSAS-II modifications (~/.G2local/  
or /Documents and Settings/<User>/G2local/).

When defining new config variables for GSAS-II, define them here with a default value: use None or a string for  
strings, or use integers or real values. Include a doc string after each variable is defined to explain what it does. Use  
names ending in _location or _directory for items that will contain directory names.

For example:
test_int = 0

test_float = 0.0

test_string = None (or)
test_string = 'value'

```python
config_example.Arc_mask_azimuth = 10.0

Specifies the default azimuthal range for creation of arc masks. Default is 10.0 degrees 2-theta.

config_example.Autoint_PollTime = 30.0

Specifies the frequency, in seconds that AutoInt checks for new files. Default is 30 seconds

config_example.Clip_on = True

if True then line plots will be clipped at plot border; if False line plots extend into white space around plot frame

config_example.Column_Metadata_directory = None

When specified and when images are read, GSAS-II will read metadata from a 1-ID style .par and a .EXT_.lbls
(EXT = image extension) or .lbls file. See `GSASIIfiles.readColMetadata()` for information on how this is done.

config_example.Contour_color = 'Paired'

Specifies the color map to be used for contour plots (images, pole figures, etc.) will be applied for new images
and if Saved for a new start of GSAS-II

config_example.DefaultAutoScale = 'userComment2'

Gives the possible selection of incident monitor names as found in an image metadata file. DefaultAutoScale
must be one of the AutoScale_ParmNames Used in AutoIntegration

config_example.Enable_logging = False

Set to True to enable use of command logging (under development.)

config_example.Help_mode = 'browser'

Set to “internal” to use a Python-based web viewer to display help documentation and tutorials. If set to the
default (“browser”) the default web browser is used.

config_example.Import_directory = None

Specifies a default location for importing (reading) input files. Will be updated if Save_paths is True. Note that
os.path.expanduser is run on this before it is used, so the user’s home directory can be specified with a ‘~’.

config_example.Instprm_default = False

when True, GSAS-II instprm file are shown as default; when False, old GSAS stype prm, etc files are default

config_example.Main_Pos = '(100,100)'

Main window location - will be updated & saved when user moves it. If position is outside screen then it will
be repositioned to default

config_example.Main_Size = '(700,450)'

Main window size (width, height) - initially uses wx.DefaultSize but will updated and saved as the user changes
the window

config_example.Movie_fps = 10

Specifies movie frames-per-second; larger number will make smoother modulation movies but larger files.

config_example.Movie_time = 5

Specifies time in sec for one modulation loop; larger number will give more frames for same fps’

config_example.Multiprocessing_cores = 0

Specifies the number of cores to use when performing multicore computing. A number less than zero causes the
recommended number of cores [using multiprocessing.cpu_count()/2] to be used. Setting this number to 0 or 1
avoids use of the multiprocessing module: all computations are performed in-line.
config_example.PDF_Rmax = 100.0
Maximum radius for G(r) calculations: range is from 10-200A; default is 100A

config_example.Plot_Colors = 'k r g b m c'
The colors for line plots: use one of 'k'-black, 'r'-red, 'b'-blue, 'g'-green, 'm'-magenta, 'c'-cyan for the line colors in order of obs., calc., back., diff., color5 & color6 separated by spaces; 6 items required.

config_example.Plot_Pos = '(200,200)'
Plot window location - will be updated & saved when user moves it these widows. If position is outside screen then it will be repositioned to default

config_example.Plot_Size = '(700,600)'
Plot window size (width, height) - initially uses wx.DefaultSize but will updated and saved as the user changes the window

config_example.Ring_mask_thickness = 0.1
Specifies the default thickness for creation of ring and arc masks. Default is 0.1 degrees 2-theta.

config_example.Save_paths = False
When set to True, the last-used path for saving of .gpx and for importing of input files is saved in the configuration file. Note that since this causes the config.py file to be updated whenever files are saved/imported, any temporary config settings can be saved to disk at that point.

config_example.Show_timing = False
If True, shows various timing results.

config_example.Spot_mask_diameter = 1.0
Specifies the default diameter for creation of spot masks. Default is 1.0 mm

config_example.Starting_directory = None
Specifies a default location for starting GSAS-II and where .gpx files should be read from. Will be updated if Save_paths is True. Note that os.path.expanduser is run on this before it is used, so the user’s home directory can be specified with a ‘~’.

config_example.Tick_length = 8.0
Specifies the length of phase tick marks in pixels. Default is 8.

config_example.Tick_width = 1.0
Specifies the width of phase tick marks in pixels. Fractional values do seem to produce an effect. Default is 1.

config_example.Transpose = False
Set to True to cause images to be Transposed when read (for code development)

config_example.Tutorial_location = None
Change this to place tutorials by in a different spot. If None, this defaults to <user>/My Documents/G2tutorials (on windows) or <user>/G2tutorials. If you want to use a different location, this can be set here. To install into the location where GSAS-II is installed, use this:

```
Tutorial_location = GSASIIpath.path2GSAS2
```

As another example, to use ~/.G2tutorials do this:

```
Tutorial_location = '~/.G2tutorials'
```

Note that os.path.expanduser is run on Tutorial_location before it is used. Also note that GSASIIpath is imported inside config.py; other imports should be avoided.

config_example.debug = False
Set to True to turn on debugging mode. This enables use of IPython on exceptions and on calls to GSASIIpath.IPyBreak(). Calls to GSASIIpath.pdbBreak() will invoke pdb at that location.

If debug is False, calls to GSASIIpath.IPyBreak() and GSASIIpath.pdbBreak() are ignored.

1.4. GSAS-II Utility Modules

35
config_example.fullIntegrate = True
If True then full image integration is default; False otherwise

cfg_example.logging_debug = False
Set to True to enable debug for logging (under development.)

cfg_example.previous_GPX_files = []
A list of previously used .gpx files

cfg_example.wxInspector = False
If set to True, the wxInspector widget is displayed when GSAS-II is started.

1.4.4 GSASIIElem: functions for element types

GSASIIElem.CheckElement(El)
Check if element El is in the periodic table

Parameters
  • El (str) – One or two letter element symbol, capitalization ignored

Returns True if the element is found

GSASIIElem.ComptonFac(El, SQ)
compute Compton scattering factor

Parameters
  • El – element dictionary
  • SQ – (sin-theta/lambda)**2

Returns compton scattering factor

GSASIIElem.FPcalc(Orbs, KEv)
Compute real & imaginary resonant X-ray scattering factors

Parameters
  • Orbs – list of orbital dictionaries as defined inGetXsectionCoeff
  • KEv – x-ray energy in keV

Returns C: (f', f", \mu): real, imaginary parts of resonant scattering & atomic absorption coeff.

GSASIIElem.FixValence(El)
Returns the element symbol, even when a valence is present

GSASIIElem.GetAtomInfo(El, ifMag=False)
reads element information from atmdata.py

GSASIIElem.GetBLtable(General)
returns a dictionary of neutron scattering length data for atom types & isotopes found in General

Parameters
  • General (dict) – dictionary of phase info.; includes AtomTypes & Isotopes

Returns BTable, dictionary of scattering length data; key is atom type

GSASIIElem.GetFFC5(ElSym)
Get 5 term form factor and Compton scattering data

Parameters
  • ElSym – str(1-2 character element symbol with proper case);

Returns El dictionary with 5 term form factor & compton coefficients

GSASIIElem.GetFFtable(atomTypes)
returns a dictionary of form factor data for atom types found in atomTypes
Parameters **atomTypes** (*list*) – list of atom types

Returns **FFtable**, dictionary of form factor data; key is atom type

GSASIIElem\textunderscore GetFormFactorCoeff \((El)\)

Read X-ray form factor coefficients from *atomdata.py* file

Parameters **El** (*str*) – element 1-2 character symbol, case irrelevant

Returns **FormFactors**: list of form factor dictionaries

Each X-ray form factor dictionary is:
- **Symbol**: 4 character element symbol with valence (e.g. ‘NI+2’)
- **Z**: atomic number
- **fa**: 4 A coefficients
- **fb**: 4 B coefficients
- **fc**: C coefficient

GSASIIElem\textunderscore GetMFtable \((atomTypes, Landeg)\)

returns a dictionary of magnetic form factor data for atom types found in *atomTypes*

Parameters
- **atomTypes** (*list*) – list of atom types
- **Landeg** (*list*) – Lande g factors for *atomTypes*

Returns **FFtable**, dictionary of form factor data; key is atom type

GSASIIElem\textunderscore GetMagFormFacCoeff \((El)\)

Read magnetic form factor data from *atmdata.py*

Parameters **El** – 2 character element symbol

Returns **MagFormFactors**: list of all magnetic form factors dictionaries for element *El.*

Each dictionary contains:
- ‘Symbol’: Symbol
- ‘Z’: Z
- ‘mfa’: 4 MA coefficients
- ‘nfa’: 4 NA coefficients
- ‘mfb’: 4 MB coefficients
- ‘nfb’: 4 NB coefficients
- ‘mfc’: MC coefficient
- ‘nfc’: NC coefficient

GSASIIElem\textunderscore GetXsectionCoeff \((El)\)

Read atom orbital scattering cross sections for fprime calculations via Cromer-Lieberman algorithm

Parameters **El** – 2 character element symbol

Returns **Orbs**: list of orbitals each a dictionary with detailed orbital information used by FPcalc

Each dictionary is:
- ‘OrbName’: Orbital name read from file
• ‘IfBe’ 0/2 depending on orbital
• ‘BindEn’: binding energy
• ‘BB’: BindEn/0.02721
• ‘XSectIP’: 5 cross section inflection points
• ‘ElEterm’: energy correction term
• ‘SEdge’: absorption edge for orbital
• ‘Nval’: 10/11 depending on IfBe
• ‘LEner’: 10/11 values of log(energy)
• ‘LXSect’: 10/11 values of log(cross section)

\textbf{GSASIElem. MagScatFac \((El, SQ)\)}

compute value of form factor

\textbf{Parameters}

• \textit{El} – element dictionary defined in GetFormFactorCoeff
• \textit{SQ} – (sin-theta/lambda)**2
• \textit{gfac} – Lande g factor (normally = 2.0)

\textbf{Returns} real part of form factor

\textbf{GSASIElem. ScatFac \((El, SQ)\)}

compute value of form factor

\textbf{Parameters}

• \textit{El} – element dictionary defined in GetFormFactorCoeff
• \textit{SQ} – (sin-theta/lambda)**2

\textbf{Returns} real part of form factor

\textbf{GSASIElem. getBLvalues \((BLtables, ifList=False)\)}

Needs a doc string

\textbf{GSASIElem. getFFvalues \((FFtables, SQ, ifList=False)\)}

Needs a doc string

\textbf{GSASIElem. getMFvalues \((MFtables, SQ, ifList=False)\)}

Needs a doc string

\section*{1.4.5 \textit{GSASIIlattice: Unit cells}}

Perform lattice-related computations

Note that \(G\) is the reciprocal lattice tensor, and \(g\) is its inverse, \(G = g^{-1}\), where

\[
g = \begin{pmatrix}
a^2 & a b \cos \gamma & a c \cos \beta \\
ab c \cos \gamma & b^2 & b c \cos \alpha \\
ac \cos \beta & bc \cos \alpha & c^2
\end{pmatrix}
\]

The “A tensor” terms are defined as \(A = (G_{11} \quad G_{22} \quad G_{33} \quad 2G_{12} \quad 2G_{13} \quad 2G_{23})\) and \(A\) can be used in this fashion:
\[
d^* = \sqrt{A_{11} h^2 + A_{22} k^2 + A_{33} l^2 + A_{44} h^2 + A_{55} k^2 + A_{66} l^2},
\]
where \(d\) is the \(d\)-spacing, and \(d^*\) is the reciprocal lattice spacing, \(Q = 2\pi d^* = 2\pi / d\).
GSASIIlattice.A2Gmat(A, inverse=True)
Fill real & reciprocal metric tensor (G) from A.

Parameters
• A – reciprocal metric tensor elements as [G11,G22,G33,2*G12,2*G13,2*G23]
• inverse (bool) – if True return both G and g; else just G

Returns reciprocal (G) & real (g) metric tensors (list of two numpy 3x3 arrays)

GSASIIlattice.A2cell(A)
Compute unit cell constants from A


Returns a,b,c,alpha, beta, gamma (degrees) - lattice parameters

GSASIIlattice.A2invcell(A)
Compute reciprocal unit cell constants from A returns tuple with a*,b*,c*,alpha*, beta*, gamma* (degrees)

GSASIIlattice.CellAbsorption(ElList, Volume)
Compute unit cell absorption

Parameters
• ElList (dict) – dictionary of element contents including mu and number of atoms be cell
• Volume (float) – unit cell volume

Returns mu-total/Volume

GSASIIlattice.CellBlock(nCells)
Generate block of unit cells n*n*n on a side; [0,0,0] centered, n = 2*nCells+1 currently only works for nCells = 0 or 1 (not >1)

GSASIIlattice.CentCheck(Cent, H)
needs doc string

GSASIIlattice.CosAngle(U, V, G)
calculate cos of angle between U & V in generalized coordinates defined by metric tensor G

Parameters
• U – 3-vectors assume numpy arrays, can be multiple reflections as (N,3) array
• V – 3-vectors assume numpy arrays, only as (3) vector
• G – metric tensor for U & V defined space assume numpy array

Returns cos(phi)

GSASIIlattice.CosSinAngle(U, V, G)
calculate sin & cos of angle between U & V in generalized coordinates defined by metric tensor G

Parameters
• U – 3-vectors assume numpy arrays
• V – 3-vectors assume numpy arrays
• G – metric tensor for U & V defined space assume numpy array

Returns cos(phi) & sin(phi)

GSASIIlattice.CrsAng(H, cell, SGData)
needs doc string
GSASIllattice.Dsp2pos (Inst, dsp)
   convert d-spacing to powder pattern position (2-theta or TOF, musc)

GSASIllattice.FindNonstandard (controls, Phase)
   Find nonstandard setting of magnetic cell that aligns with parent nuclear cell

   Parameters
   • controls – list unit cell indexing controls
   • Phase – dict new magnetic phase data (NB: not G2 phase construction); modified here

   Returns  None

GSASIllattice.Flnh (Start, SHCoef, phi, beta, SGData)
   needs doc string

GSASIllattice.GenHBravais (dmin, Bravais, A)
   Generate the positionally unique powder diffraction reflections

   Parameters
   • dmin – minimum d-spacing in A
   • Bravais – lattice type (see GetBraviasNum). Bravais is one of:
     – 0 F cubic
     – 1 I cubic
     – 2 P cubic
     – 3 R hexagonal (trigonal not rhombohedral)
     – 4 P hexagonal
     – 5 I tetragonal
     – 6 P tetragonal
     – 7 F orthorhombic
     – 8 I orthorhombic
     – 9 A orthorhombic
     – 10 B orthorhombic
     – 11 C orthorhombic
     – 12 P orthorhombic
     – 13 I monoclinic
     – 14 C monoclinic
     – 15 P monoclinic
     – 16 P triclinic
   • A – reciprocal metric tensor elements as [G11,G22,G33,2*G12,2*G13,2*G23]

   Returns  HKL unique d list of [h,k,l,d,-1] sorted with largest d first

GSASIllattice.GenHLauae (dmin, SGData, A)
   Generate the crystallographically unique powder diffraction reflections for a lattice and Bravais type

   Parameters
   • dmin – minimum d-spacing
• **SGData** – space group dictionary with at least
  - ‘SGUniq’: code for unique monoclinic axis one of ‘a’, ‘b’, ‘c’ (only if ‘SGLaue’ is ‘2/m’) otherwise an empty string

• **A** – reciprocal metric tensor elements as \([G_{11}, G_{22}, G_{33}, 2*G_{12}, 2*G_{13}, 2*G_{23}]\)

**Returns**

HKL = list of \([h, k, l, d]\) sorted with largest \(d\) first and is unique part of reciprocal space ignoring anomalous dispersion

`GSASIIlattice.GenPfHKLs(nMax, SGData, A)`

Generate the unique pole figure reflections for a lattice and Bravais type. Min \(d\)-spacing=1.0Å & no more than \(nMax\) returned

**Parameters**

- **nMax** – maximum number of hkls returned
- **SGData** – space group dictionary with at least
  - ‘SGUniq’: code for unique monoclinic axis one of ‘a’, ‘b’, ‘c’ (only if ‘SGLaue’ is ‘2/m’) otherwise an empty string
- **A** – reciprocal metric tensor elements as \([G_{11}, G_{22}, G_{33}, 2*G_{12}, 2*G_{13}, 2*G_{23}]\)

**Returns**

HKL = list of ‘h k l’ strings sorted with largest \(d\) first; no duplicate zones

`GSASIIlattice.GenSHCoeff(SGLaue, SamSym, L, IfLMN=True)`

`GSASIIlattice.GenSSHLaue(dmin, SGData, SSGData, Vec, maxH, A)`

`GSASIIlattice.GetBraviasNum(center, system)`

Determine the Bravais lattice number, as used in GenHBravais

**Parameters**


**Returns** a number between 0 and 13 or throws a ValueError exception if the combination of center, system is not found (i.e. non-standard)

`GSASIIlattice.GetKcl(L, N, SGLaue, phi, beta)`

`GSASIIlattice.GetKclKsl(L, N, SGLaue, psi, phi, beta)`

This is used for spherical harmonics description of preferred orientation; cylindrical symmetry only \((M=0)\) and no sample angle derivatives returned

`GSASIIlattice.GetKsl(L, M, SamSym, psi, gam)`

1.4. **GSAS-II Utility Modules**
GSASIIlattice. **Glnh** *(Start, SHCoeff, psi, gam, SamSym)*
needs doc string

GSASIIlattice. **Gmat2A** *(G)*

Extract A from reciprocal metric tensor (G)

**Parameters**
G – reciprocal maetic tensor (3x3 numpy array)

**Returns**
A = [G11,G22,G33,2*G12,2*G13,2*G23]

GSASIIlattice. **Gmat2AB** *(G)*

Computes orthogonalization matrix from reciprocal metric tensor G

**Returns**

tuple of two 3x3 numpy arrays (A,B)

- A for crystal to Cartesian transformations (A*x = np.inner(A,x) = X)
- B (= inverse of A) for Cartesian to crystal transformation (B*X = np.inner(B,X) = x)

GSASIIlattice. **Gmat2cell** *(g)*

Compute real/reciprocal lattice parameters from real/reciprocal metric tensor (g/G) The math works the same either way.

**Parameters**
(or G)

(g) – real (or reciprocal) metric tensor 3x3 array

**Returns**
a,b,c,alpha, beta, gamma (degrees) (or a*,b*,c*,alpha*,beta*,gamma* degrees)

GSASIIlattice. **HKL2SpAng** *(H, cell, SGData)*

Computes spherical coords for hkls; view along 001

**Parameters**

- **H** *(array)* – arrays of hkl
- **cell** *(tuple)* – a,b,c, alpha, beta, gamma (degrees)
- **SGData** *(dict)* – space group dictionary

**Returns**
arrays of r,phi,psi (radius,inclination,azimuth) about 001

GSASIIlattice. **Hx2Rh** *(Hx)*

needs doc string

GSASIIlattice. **LaueUnique** *(Laue, HKLF)*

Impose Laue symmetry on hkl

**Parameters**

- **Laue** *(str)* – Laue symbol, as below
  - centrosymmetric Laue groups:
    - "-1", "2/m", "112/m", "2/m11", "mmm", "-42m", "-4m2", "4/mmm", "-3",
    - "-31m", "-3m1", "6/m", "6/mmm", "m3", "m3m"
  - noncentrosymmetric Laue groups:
    - "1", "2", "211", "112", "m", "m11", "1m", "222", "mm2", "m2m", "2mm",
    - "4", "-4", "422", "4mm", "3", "312", "321", "31m", "3m1", "6", "-6",
    - "622", "6mm", "-62m", "-6m2", "23", "432", "-43m"
- **HKLF** – np.array([[h,k,l,...]]) reflection set to be converted

**Returns**
HKLF new reflection array with imposed Laue symmetry
GSASIIlattice.**LaueUnique2** *(SGData, refList)*

Impose Laue symmetry on hkl

**Parameters**

- **SGData** – space group data from ‘P ‘+Laue
- **HKLF** – np.array([[h,k,l,...]]) reflection set to be converted

**Returns** HKLF new reflection array with imposed Laue symmetry

GSASIIlattice.**MaxIndex** *(dmin, A)*

needs doc string

GSASIIlattice.**OdfChk** *(SGLaue, L, M)*

needs doc string

GSASIIlattice.**PlaneIntercepts** *(Amat, H, phase, stack)*

find unit cell intercepts for a stack of hkl planes

GSASIIlattice.**Pos2dsp** *(Inst, pos)*

convert powder pattern position (2-theta or TOF, musec) to d-spacing

GSASIIlattice.**Rh2Hx** *(Rh)*

needs doc string

GSASIIlattice.**SamAng** *(Tth, Gangls, Sangl, IFCoup)*

Compute sample orientation angles vs laboratory coord. system

**Parameters**

- **Tth** – Signed theta
- **Gangls** – Sample goniometer angles phi,chi,omega,azimuth
- **Sangl** – Sample angle zeros om-0, chi-0, phi-0
- **IFCoup** – True if omega & 2-theta coupled in CW scan

**Returns** psi,gam: Sample odf angles dPSdA,dGMdA: Angle zero derivatives

GSASIIlattice.**SwapIndex** *(Axis, H)*

needs doc string

GSASIIlattice.**SwapItems** *(Alist, pos1, pos2)*

exchange 2 items in a list

GSASIIlattice.**TOF2dsp** *(Inst, Pos)*

convert powder pattern TOF, musec to d-spacing by successive approximation Pos can be numpy array

GSASIIlattice.**TransformCell** *(cell, Trans)*

Transform lattice parameters by matrix

**Parameters**

- **cell** – list a,b,c,alpha,beta,gamma,(volume)
- **Trans** – array transformation matrix

**Returns** array transformed a,b,c,alpha,beta,gamma,volume

GSASIIlattice.**TransformPhase** *(oldPhase, newPhase, Trans, Uvec, Vvec, fMag)*

Transform atoms from oldPhase to newPhase M’ is inv(M) does X’ = M(X-U)+V transformation for coordinates and U’ = MUM/det(M) for anisotropic thermal parameters

**Parameters**

---

1.4. **GSAS-II Utility Modules**
**oldPhase** – dict G2 phase info for old phase

**newPhase** – dict G2 phase info for new phase; with new cell & space group atoms are from oldPhase & will be transformed

**Trans** – lattice transformation matrix M

**Uvec** – array parent coordinates transformation vector U

**Vvec** – array child coordinate transformation vector V

**ifMag** – bool True if convert to magnetic phase; if True all nonmagnetic atoms will be removed

**Returns** newPhase dict modified G2 phase info

**Returns** atCodes list atom transformation codes

```python
GSASIIlattice.U6toUij(U6)
```

Fill matrix (Uij) from U6 = [U11,U22,U33,U12,U13,U23] NB: there is a non numpy version in GSASIIspace: U2Uij

**Parameters**

- **U6** (`list`) – 6 terms of u11,u22,...

**Returns** Uij - numpy [3][3] array of uij

```python
GSASIIlattice.Uij2Ueqv(Uij, GS, Amat)
```

returns 1/3 trace of diagonalized U matrix

```python
GSASIIlattice.Uij2betaij(Uij, G)
```

Convert Uij to beta-ij tensors – stub for eventual completion

**Parameters**

- **Uij** – numpy array [Uij]
- **G** – reciprocal metric tensor

**Returns** beta-ij - numpy array [beta-ij]

```python
GSASIIlattice.UijtoU6(U)
```

Fill vector [U11,U22,U33,U12,U13,U23] from Uij NB: there is a non numpy version in GSASIIspace: Uij2U

```python
GSASIIlattice.betaij2Uij(betaij, G)
```

Convert beta-ij to Uij tensors


```python
GSASIIlattice.calc_V(A)
```

Compute the real lattice volume (V) from A

```python
GSASIIlattice.calc_rDsq(H, A)
```

needs doc string

```python
GSASIIlattice.calc_rDsq2(H, G)
```

needs doc string

```python
GSASIIlattice.calc_rDsqSS(H, A, vec)
```

needs doc string

```python
GSASIIlattice.calc_rDsqT(H, A, Z, tof, difC)
```

needs doc string

```python
GSASIIlattice.calc_rDsqTSS(H, A, vec, Z, tof, difC)
```

needs doc string
GSASIIlattice.calc_rDsqZ(H, A, Z, tth, lam)
needs doc string

GSASIIlattice.calc_rDsqZSS(H, A, vec, Z, tth, lam)
needs doc string

GSASIIlattice.calc_rV(A)
Compute the reciprocal lattice volume (V*) from A

GSASIIlattice.calc_rVsq(A)
Compute the square of the reciprocal lattice volume (1/V**2) from A'

GSASIIlattice.cell2A(cell)

Parameters cell – [a,b,c,alpha,beta,gamma] (degrees)

Returns G reciprocal metric tensor as 3x3 numpy array

GSASIIlattice.cell2AB(cell)
Computes orthogonalization matrix from unit cell constants

Parameters cell (tuple) – a,b,c, alpha, beta, gamma (degrees)

Returns tuple of two 3x3 numpy arrays (A,B) A for crystal to Cartesian transformations A*x = np.inner(A,x) = X B (= inverse of A) for Cartesian to crystal transformation B*X = np.inner(B,X) = x

GSASIIlattice.cell2GS(cell)
returns Uij to betaij conversion matrix

GSASIIlattice.cell2Gmat(cell)
Compute real and reciprocal lattice metric tensor from unit cell constants

Parameters cell – tuple with a,b,c,alpha, beta, gamma (degrees)

Returns reciprocal (G) & real (g) metric tensors (list of two numpy 3x3 arrays)

GSASIIlattice.cellDijFill(pfx, phfx, SGData, parmDict)
Returns the filled-out reciprocal cell (A) terms from the parameter dictionaries corrected for Dij.

Parameters

• pfx (str) – parameter prefix (“n::”, where n is a phase number)
• SGdata (dict) – a symmetry object
• parmDict (dict) – a dictionary of parameters

Returns A,sigA where each is a list of six terms with the A terms

GSASIIlattice.combinations(items, n)
take n distinct items, order matters

GSASIIlattice.criticalEllipse(prob)
Calculate critical values for probability ellipsoids from probability

GSASIIlattice.fillgmat(cell)
Compute lattice metric tensor from unit cell constants

Parameters cell – tuple with a,b,c,alpha, beta, gamma (degrees)

Returns 3x3 numpy array

GSASIIlattice.getHKLmax(dmin, SGData, A)
finds maximum allowed hkl for given A within dmin

1.4. GSAS-II Utility Modules

GSAS-II Developers Documentation, Release version 4249
GSASIIlattice.getPeakPos\(\text{\textit{dataType}}, \text{\textit{parmdict}}, \text{\textit{dsp}}\)
convert d-spacing to powder pattern position (2-theta or TOF, musec)

GSASIIlattice.invcell2Gmat\(\text{\textit{invcell}}\)
Compute real and reciprocal lattice metric tensor from reciprocal unit cell constants

Parameters

\textit{invcell} – \([a^*, b^*, c^*, \alpha^*, \beta^*, \gamma^*]\) (degrees)

Returns reciprocal (G) & real (g) metric tensors (list of two 3x3 arrays)

GSASIIlattice.invpolfcal\(\text{\textit{ODFln}}, \text{\textit{SGData}}, \text{\textit{phi}}, \text{\textit{beta}}\)
needs doc string

GSASIIlattice.permutations\(\text{\textit{items}}\)
take all items, order matters

GSASIIlattice.polfcal\(\text{\textit{ODFln}}, \text{\textit{SamSym}}, \text{\textit{psi}}, \text{\textit{gam}}\)
Perform a pole figure computation. Note that the the number of gam values must either be 1 or must match psi. Updated for numpy 1.8.0

GSASIIlattice.prodMGMT\(\text{\textit{G}}, \text{\textit{Mat}}\)
Transform metric tensor by matrix

Parameters

\begin{itemize}
  \item \textit{G} – array metric tensor
  \item \textit{Mat} – array transformation matrix
\end{itemize}

Returns array new metric tensor

GSASIIlattice.rotdMat\(\text{\textit{angle}}, \text{\textit{axis}=0}\)
Prepare rotation matrix for angle in degrees about axis(=0,1,2)

Parameters

\begin{itemize}
  \item \textit{angle} – angle in degrees
  \item \textit{axis} – axis \((0,1,2 = x,y,z)\) about which for the rotation
\end{itemize}

Returns rotation matrix - 3x3 numpy array

GSASIIlattice.rotdMat4\(\text{\textit{angle}}, \text{\textit{axis}=0}\)
Prepare rotation matrix for angle in degrees about axis(=0,1,2) with scaling for OpenGL

Parameters

\begin{itemize}
  \item \textit{angle} – angle in degrees
  \item \textit{axis} – axis \((0,1,2 = x,y,z)\) about which for the rotation
\end{itemize}

Returns rotation matrix - 4x4 numpy array (last row/column for openGL scaling)

GSASIIlattice.sec2HMS\(\text{\textit{sec}}\)
Convert time in sec to H:M:S string

Parameters

\textit{sec} – time in seconds

Returns H:M:S string (to nearest 100th second)

GSASIIlattice.selections\(\text{\textit{items}}, \text{\textit{n}}\)
take n (not necessarily distinct) items, order matters

GSASIIlattice.selftestlist = []
Defines a list of self-tests
**GSASIIlattice.sortHKLD**(HKLD, ifreverse, ifdup, ifSS=False)
sort reflection list on d-spacing; can sort in either order

**Parameters**
- **HKLD** – a list of [h,k,l,d,...];
- **ifreverse** – True for largest d first
- **ifdup** – True if duplicate d-spacings allowed

**Returns** sorted reflection list

**GSASIIlattice.test1()**
test cell2A and A2Gmat

**GSASIIlattice.test2()**
test Gmat2A, A2cell, A2Gmat, Gmat2cell

**GSASIIlattice.test3()**
test invcell2Gmat

**GSASIIlattice.test4()**
test calc_rVsq, calc_rV, calc_V

**GSASIIlattice.test5()**
test A2invcell

**GSASIIlattice.test6()**
test cell2AB

**GSASIIlattice.test7()**
test GetBraviasNum(...) and GenHBravais(...)

**GSASIIlattice.test8()**
test GenHLaue

**GSASIIlattice.test9()**
test GenHLaue

**GSASIIlattice.textureIndex**(SHCoef)
needs doc string

**GSASIIlattice.transposeHKLF**(transMat, Super, refList)
Apply transformation matrix to hkl(m) param: transmat: 3x3 or 4x4 array param: Super: 0 or 1 for extra index param: refList list of h,k,l,... return: newRefs transformed list of h',k',l',... return: badRefs list of noninteger h',k',l'...

**GSASIIlattice.uniqueCombinations**(items, n)
take n distinct items, order is irrelevant

### 1.4.6 GSASIIspc: Space group module

Space group interpretation routines. Note that space group information is stored in a *Space Group (SGData)* object.

**GSASIIspc.AllOps**(SGData)
Returns a list of all operators for a space group, including those for centering and a center of symmetry

**Parameters** SGData – from SpcGroup()

**Returns**
(SGTextList,offsetList,symOpList,G2oprList) where

---

**1.4. GSAS-II Utility Modules**
• SGTextList: a list of strings with formatted and normalized symmetry operators.
• offsetList: a tuple of (dx,dy,dz) offsets that relate the GSAS-II symmetry operation to the operator in SGTextList and symOpList. These dx (etc.) values are added to the GSAS-II generated positions to provide the positions that are generated by the normalized symmetry operators.
• symOpList: a list of tuples with the normalized symmetry operations as (M,T) values (see SGOps in the Space Group object)
• G2oprList: The GSAS-II operations for each symmetry operation as a tuple with (center, mult, opnum, opcode), where center is (0,0,0), (0.5,0,0), (0.5,0.5,0.5),...; where mult is 1 or -1 for the center of symmetry where opnum is the number for the symmetry operation, in SGOps (starting with 0) and opcode is mult*(100*icen+j+1).

GSASIIspc.ApplyStringOps(A, SGData, X, Uij=[]) Needs a doc string
GSASIIspc.ApplyStringOpsMom(A, SGData, Mom) Needs a doc string
GSASIIspc.CheckSpin(isym, SGData) Check for exceptions in spin rules
GSASIIspc.ElemPosition(SGData) Under development. Object here is to return a list of symmetry element types and locations suitable for say drawing them. So far I have the element type... getting all possible locations without lookup may be impossible!

GSASIIspc.GenAtom(XYZ, SGData, All=False, Uij=[], Move=True) Generates the equivalent positions for a specified coordinate and space group

Parameters
• XYZ – an array, tuple or list containing 3 elements: x, y & z
• SGData – from SpcGroup()
• All – True return all equivalent positions including duplicates; False return only unique positions
• Uij – [U11,U22,U33,U12,U13,U23] or [] if no Uij
• Move – True move generated atom positions to be inside cell False do not move atoms

Returns
[[XYZEquiv],[Idup],[UijEquiv],spnflp]
• [XYZEquiv] is list of equivalent positions (XYZ is first entry)
• Idup = [-][C]SS where SS is the symmetry operator number (1-24), C (if not 0,0,0)
• is centering operator number (1-4) and - is for inversion Cell = unit cell translations needed to put new positions inside cell [UijEquiv] - equivalent Uij; absent if no Uij given
• +1/-1 for spin inversion of operator - empty if not magnetic

GSASIIspc.GenHKL(HKL, SGData) Generates all equivalent reflections including Friedel pairs ;param HKL: [h,k,l] must be integral values ;param SGData: space group data obtained from SpcGroup ;returns: array Uniq: equivalent reflections

GSASIIspc.GenHKLf(HKL, SGData) Uses old GSAS Fortran routine genhkl.for

Parameters
• **HKL** – \([h,k,l]\) must be integral values for `genhkl.for` to work

• **SGData** – space group data obtained from `SpcGroup`

**Returns**

- `iabsnt, mulp, Uniq, phi`
  - `iabsnt` = True if reflection is forbidden by symmetry
  - `mulp` = reflection multiplicity including Friedel pairs
  - `Uniq` = numpy array of equivalent `hkl` in descending order of `h,k,l`
  - `phi` = phase offset for each equivalent `h,k,l`

```python
GSASIIspc.GetCSpqinel(SpnFlp, dupDir)
returns Mxyz terms, multipliers, GUI flags
```

```python
GSASIIspc.GetCSuinel(siteSym)
returns Uij terms, multipliers, GUI flags & Uiso2Uij multipliers
```

```python
GSASIIspc.GetCSxinel(siteSym)
returns Xyz terms, multipliers, GUI flags
```

```python
GSASIIspc.GetGenSym(SGData)
Get the space group generator symbols
:param SGData: from `SpcGroup()`

LaueSym = ('-1', '2/m', 'mmm', '4/m', '4/mmm', '3R', '3mR', '3', '3m1', '31m', '6/m', '6/mmm', 'm3', 'm3m')
LattSym = ('P', 'A', 'B', 'C', 'I', 'F', 'R')
```

```python
GSASIIspc.GetKNsym(key)
Needs a doc string
```

```python
GSASIIspc.GetLittleGrpOps(SGData, vec)
Find rotation part of operators that leave vec unchanged

**Parameters**

• **SGData** – space group data structure as defined in `SpcGroup` above.

• **vec** – a numpy array of fractional vector coordinates

**Returns** Little - list of operators [M,T] that form the little group

```python
GSASIIspc.GetNXUPQsym(siteSym)
The codes XUPQ are for lookup of symmetry constraints for position(X), thermal parm(U) & magnetic moments (P & Q)
```

```python
GSASIIspc.GetOprName(key)
Needs a doc string
```

```python
GSASIIspc.GetOprPtrName(key)
Needs a doc string
```

```python
GSASIIspc.GetOprPtrNumber(key)
Needs a doc string
```

```python
GSASIIspc.GetSGSpin(SGData, MSgSym)
get spin generators from magnetic space group symbol
```

```python
GSASIIspc.HStrainNames(SGData)
Needs a doc string
```

```python
GSASIIspc.Latt2text(Cen)
From lattice centering vectors returns ‘;’ delimited cell centering vectors
```

1.4. **GSAS-II Utility Modules**
GSASIIspc.MT2text(Opr, reverse=False)
From space group matrix/translation operator returns text version

GSASIIspc.MagSSText2MTS(Opr, G2=False)
From magnetic super space group cif text returns matrix/translation + spin flip

GSASIIspc.MagSytSym(SytSym, dupDir, SGData)
site sym operations: 1,-1,2,3,-3,4,-4,6,-6,m need to be marked if spin inversion

GSASIIspc.MagText2MTS(mcifOpr, CIF=True)
From magnetic space group cif text returns matrix/translation + spin flip

GSASIIspc.MoveToUnitCell(xyz)
Translates a set of coordinates so that all values are >=0 and < 1

Parameters
- xyz – a list or numpy array of fractional coordinates

Returns
- XYZ - numpy array of new coordinates now 0 or greater and less than 1

GSASIIspc.Muiso2Shkl(muiso, SGData, cell)
this is to convert isotropic mustrain to generalized Shkls

GSASIIspc.MustrainCoeff(HKL, SGData)
Needs a doc string

GSASIIspc.MustrainNames(SGData)
Needs a doc string

GSASIIspc.Opposite(XYZ, toler=0.0002)
Gives opposite corner, edge or face of unit cell for position within tolerance. Result may be just outside the cell within tolerance

Parameters
- XYZ – 0 >= np.array[x,y,z] > 1 as by MoveToUnitCell
- toler – unit cell fraction tolerance making opposite

Returns
- XYZ: dict of opposite positions; key=unit cell & always contains XYZ

GSASIIspc.SGErrors(IErr)
Interprets the error message code from SpcGroup. Used in SpaceGroup.

Parameters
- IErr – see SGErro in SpcGroup()

Returns
- ErrString - a string with the error message or “Unknown error”

GSASIIspc.SGPrint(SGData, AddInv=False)
Print the output of SpcGroup in a nicely formatted way. Used in SpaceGroup

Parameters
- SGData – from SpcGroup()

Returns
- SGText - list of strings with the space group details SGTable - list of strings for each of the operations

GSASIIspc.SGProd(OpA, OpB)
Form space group operator product. OpA & OpB are [M,V] pairs; both must be of same dimension (3 or 4). Returns [M,V] pair

GSASIIspc.SGPtGroup(SGData)
Determine point group of the space group - done after space group symbol has been evaluated by SpcGroup. Only short symbols are allowed
Parameters **SGData** – from :func: SpcGroup

Returns **SSGPtGrp & SSGKl** (only defaults for Mono & Ortho)

**GSASIIspc.SGpolar** *(SGData)*
Determine identity of polar axes if any

**GSASIIspc.SSChoice** *(SGData)*
Gets the unique set of possible super space groups for a given space group

**GSASIIspc.SSGModCheck** *(Vec, modSymb, newMod=True)*
Checks modulation vector compatibility with supersymmetry space group symbol. if newMod: Superspace group symbol takes precidence & the vector will be modified accordingly

**GSASIIspc.SSGPrint** *(SGData, SSGData, AddInv=False)*
Print the output of SSpcGroup in a nicely formatted way. Used in SSpaceGroup

Parameters

- **SGData** – space group data structure as defined in SpcGroup above.
- **SSGData** – from **SSpcGroup()**

Returns **SSGText** - list of strings with the superspace group details **SGTable** - list of strings for each of the operations

**GSASIIspc.SSLatt2text** *(SSGCen)*
Lattice centering vectors to text

**GSASIIspc.SSMT2text** *(Opr)*
From superspace group matrix/translation operator returns text version

**GSASIIspc.SSpaceGroup** *(SGSymbol, SSymbol)*
Print the output of SSpcGroup in a nicely formatted way.

Parameters

- **SGSymbol** – space group symbol with spaces between axial fields.
- **SSSymbol** – superspace group symbol extension (string).

Returns nothing

**GSASIIspc.SSpGrp** *(SGData, SSymbol)*
Determines supersymmetry information from superspace group name; currently only for (3+1) superlattices

Parameters

- **SGData** – space group data structure as defined in SpcGroup above (see **SGData**).
- **SSSymbol** – superspace group symbol extension (string) defining modulation direction & generator info.

Returns

(SSGError,SSGData)

- **SGError** = 0 for no errors; >0 for errors (see SGErrors below for details)
- **SSGData** - is a dict (see **Superspace Group object**) with entries:
  - ‘SSGrp’: full superspace group symbol, accidental spaces removed; for display only
  - ‘SSCen’: 4D cell centering vectors [0,0,0,0] at least
  - ‘SSGOps’: 4D symmetry operations as [M,T] so that M*x+T = x’
GSASIIspc.\texttt{SpaceGroup}(\textit{SGSymbol})

Print the output of \texttt{SpcGroup} in a nicely formatted way.

\textbf{Parameters} \textit{SGSymbol} – space group symbol (string) with spaces between axial fields

\textbf{Returns} nothing

GSASIIspc.\texttt{SpcGroup}(\textit{SGSymbol})

Determines cell and symmetry information from a short H-M space group name

\textbf{Parameters} \textit{SGSymbol} – space group symbol (string) with spaces between axial fields

\textbf{Returns}

(\textit{SGError}, \textit{SGData})

- \textit{SGError} = 0 for no errors; >0 for errors (see \textit{SGErrors} below for details)

- \textit{SGData} - is a dict (see \textit{Space Group object}) with entries:
  - ‘SpGrp’: space group symbol, slightly cleaned up
  - ‘SGFixed’: True if space group data can not be changed, e.g. from magnetic cif; otherwise False
  - ‘SGGray’: True if ‘1’ in symbol - gray group for mag. incommensurate phases
  - ‘SGInv’: boolean; True if centrosymmetric, False if not
  - ‘SGUniq’: one of ‘a’, ‘b’, ‘c’ if monoclinic, ‘’ otherwise
  - ‘SGCen’: cell centering vectors [0,0,0] at least
  - ‘SGPtGrp’: one of 32 point group symbols (with some permutations), which is filled by \texttt{SGPtGroup}, is external (KE) part of supersymmetry point group
  - ‘SSGKl’: default internal (Kl) part of supersymmetry point group; modified in supersymmetry stuff depending on chosen modulation vector for Mono & Ortho
  - ‘BNSlattsym’: BNS lattice symbol & centering op - used for magnetic structures

GSASIIspc.\texttt{StandardizeSpcName}(\textit{spcgroup})

Accept a spacegroup name where spaces may have not been used in the names according to the GSAS convention (spaces between symmetry for each axis) and return the space group name as used in GSAS

GSASIIspc.\texttt{StringOpsProd}(\textit{A, B, SGData})

Find A*B where A & B are in strings ‘-‘ + ‘100*c+n’ + ‘+ijk’ where ‘-‘ indicates inversion, c(>0) is the cell centering operator, n is operator number from SgOps and ijk are unit cell translations (each may be <0). Should return resultant string - C. \textit{SGData} - dictionary using entries:

- ‘SGCen’: cell centering vectors [0,0,0] at least

- ‘SGOps’: symmetry operations as [M,T] so that M*x+T = x’
GSASIIspc.**SytSym** *(XYZ, SGData)*
Generates the number of equivalent positions and a site symmetry code for a specified coordinate and space group

**Parameters**
- **XYZ** – an array, tuple or list containing 3 elements: x, y & z
- **SGData** – from SpcGroup

**Returns** a four element tuple:
- The 1st element is a code for the site symmetry (see GetKNsym)
- The 2nd element is the site multiplicity
- Ndup number of overlapping operators
- dupDir Dict - dictionary of overlapping operators

GSASIIspc.**Text2MT** *(mcifOpr, CIF=True)*
From space group cif text returns matrix/translation

GSASIIspc.**TextOps** *(text, table, reverse=False)*
Makes formatted operator list :param text,table: arrays of text made by SGPrint :param reverse: True for x+1/2 form; False for 1/2+x form :returns: OpText: full list of symmetry operators; one operation per line generally printed to console for use via cut/paste in other programs, but could be used for direct input

GSASIIspc.**Trans2Text** *(Trans)*
from transformation matrix to text

GSASIIspc.**UpdateSytSym** *(Phase)*
Update site symmetry/site multiplicity after space group/BNS lattice change

GSASIIspc.**altSettingOrtho**
A dictionary of alternate settings for orthorhombic unit cells

GSASIIspc.**checkHKLextc** *(HKL, SGData)*
Checks if reflection extinct - does not check centering

**Parameters**
- **HKL** – [h,k,l]
- **SGData** – space group data obtained from SpcGroup

**Returns** True if extinct; False if allowed

GSASIIspc.**checkMagextc** *(HKL, SGData)*

**Parameters**
- **HKL** – [h,k,l]
- **SGData** – space group data obtained from SpcGroup; must have magnetic symmetry Spn- Flp data

**Returns** True if magnetically extinct; False if allowed (to match GenHKLf)

GSASIIspc.**fixMono** *(SpGrp)*
fills b-unique monoclinics in e.g. P 1 2/1c 1 -> P 21/c

GSASIIspc.**selftestlist** = [<function test0>, <function test1>, <function test2>, <function test3>]
Defines a list of self-tests
A dictionary of orthorhombic space groups that were renamed in the 2002 Volume A, along with the pre-2002 name. The e designates a double glide-plane

A dictionary of all spacegroups that have 2nd settings; the value is the 1st -> 2nd setting transformation vector as X(2nd) = X(1st)-V, nonstandard ones are included.

Space groups indexed by number

A dictionary of space groups as ordered and named in the pre-2002 International Tables Volume A, except that spaces are used following the GSAS convention to separate the different crystallographic directions. Note that the symmetry codes here will recognize many non-standard space group symbols with different settings. They are ordered by Laue group

GSASIIspc.splitSSsym(SSymbol)
Splits supersymmetry symbol into two lists of strings

GSASIIspc.test0()
self-test #0: exercise MoveToUnitCell

GSASIIspc.test1()
self-test #1: SpcGroup against previous results

GSASIIspc.test2()
self-test #2: SpcGroup against cctbx (sgtbx) computations

GSASIIspc.test3()
self-test #3: exercise SytSym (includes GetOprPtrName, GenAtom, GetKNsym) for selected space groups against info in IT Volume A

1.4.7 GSASIIdata: Data for computations

At present this module defines one dict, ramachandranDist, which contains arrays for All and specific amino acids

1.4.8 GSASIIfiles: data (non-GUI) I/O routines

Module with miscellaneous routines for input and output from files.

This module should not contain any references to wxPython so that it can be imported for scriptable use or potentially on clients where wx is not installed.

Future refactoring: This module and GSASIIIO.py needs some work to move non-wx routines here. It may will likely make sense to rename the module(s) at that point.

GSASIIfiles.G2Print(*args, **kwargs)
Print with filtering based level of output (see G2SetPrintLevel()). Use G2Print() as replacement for print().

Parameters mode (str) – if specified, this should contain the mode for printing (‘error’, ‘warn’ or anything else). If not specified, the first argument of the print command (args[0]) should contain the string ‘error’ for error messages and ‘warn’ for warning messages (capitalization and additional letters ignored.)
GSASIIfiles.G2SetPrintLevel(level)
Set the level of output from calls to G2Print(), which should be used in place of print() within GSASII. Settings for the mode are ‘all’, ‘warn’, ‘error’ or ‘none’

Parameters level (str) – a string used to set the print level, which may be ‘all’, ‘warn’, ‘error’ or ‘none’. Note that capitalization and extra letters in level are ignored, so ‘Warn’, ‘warnings’, etc. will all set the mode to ‘warn’

GSASIIfiles.G2printLevel = 'all'
This defines the level of output from calls to G2Print(), which should be used in place of print() within this module. Settings for this are ‘all’, ‘warn’, ‘error’ or ‘none’. Also see: G2Print() and G2SetPrintLevel().

GSASIIfiles.GetColumnMetadata(reader)
Add metadata to an image from a column-type metadata file using readColMetadata()

Parameters reader – a reader object from reading an image

GSASIIfiles.LoadControls(Slines, data)
Read values from a .imctrl (Image Controls) file

GSASIIfiles.LoadExportRoutines(parent, traceback=False)
Routine to locate GSASII exporters

GSASIIfiles.LoadImportRoutines(prefix, errprefix=None, traceback=False)
Routine to locate GSASII importers matching a prefix string

GSASIIfiles.PDFWrite(PDFentry, fileroot, PDFsaves, PDFControls, Inst={}, Limits=[])
Write PDF-related data (G(r), S(Q),...) into files, as selected.

Parameters

• PDFentry (str) – name of the PDF entry in the tree. This is used for comments in the file specifying where it came from; it can be arbitrary

• fileroot (str) – name of file(s) to be written. The extension will be ignored.

• PDFsaves (list) – flags that determine what type of file will be written: PDFsaves[0], if True writes a I(Q) file with a .iq extension PDFsaves[1], if True writes a S(Q) file with a .sq extension PDFsaves[2], if True writes a F(Q) file with a .fq extension PDFsaves[3], if True writes a G(r) file with a .gr extension PDFsaves[4], if True writes G(r) in a pdfGUI input file with a .gr extension. Note that if PDFsaves[3] and PDFsaves[4] are both True, the pdfGUI overwrites the G(r) file.

• PDFControls (dict) – The PDF parameters and computed results

• Inst (dict) – Instrument parameters from the PDWR entry used to compute the PDF. Needed only when PDFsaves[4] is True.

• Limits (list) – Computation limits from the PDWR entry used to compute the PDF. Needed only when PDFsaves[4] is True.

GSASIIfiles.ReadPowderInstprm(instLines, bank, databanks, rd)
Read lines from a GSAS-II (new) instrument parameter file similar to G2pwdGUI.OnLoad If instprm file has multiple banks each with header #Bank n: ..., this finds matching bank no. to load - problem with nonmatches?

Note that this routine performs a similar role to GSASIIdataGUI.GSASII.ReadPowderInstprm(), but that will call a GUI routine for selection when needed. This routine will raise exceptions on errors and will select the first bank when a choice might be appropriate. TODO: refactor to combine the two routines.

Parameters

• instLines (list) – strings from GSAS-II parameter file; can be concatenated with ‘;’
• **bank** *(int)* – bank number to check when instprm file has ‘#BANK n:...’ strings when bank = n then use parameters; otherwise skip that set. Ignored if BANK n: not present. NB: this kind of instprm file made by a Save all profile command in Instrument Parameters

**Return dict**  Inst instrument parameter dict if OK, or str: Error message if failed

(transliterated from GSASIIdataGUI.py:1235 (rev 3008), function of the same name)

GSASIIfiles.**RereadImageData** *(ImageReaderlist, imagefile, ImageTag=None, FormatName=“”)*

Read a single image with an image importer. This is called to reread an image after it has already been imported, so it is not necessary to reload metadata.

Based on **GetImageData.GetImageData()** which this can replace where imageOnly=True

**Parameters**

- **ImageReaderlist** *(list)* – list of Reader objects for images
- **imagefile** *(str)* – name of image file
- **ImageTag** *(int/str)* – specifies a particular image to be read from a file. First image is read if None (default).
- **formatName** *(str)* – the image reader formatName

**Returns**  an image as a numpy array

GSASIIfiles.**SetPowderInstParms** *(Iparm, rd)*

extracts values from instrument parameters in rd.instdict or in array Iparm. Create and return the contents of the instrument parameter tree entry.

GSASIIfiles.**WriteControls** *(filename, data)*

Write current values to a .imctrl (Image Controls) file

GSASIIfiles.**evalColMetadataDicts** *(items, labels, lbldict, keyCols, keyExp, ShowError=False)*

Evaluate the metadata for a line in the .par file

GSASIIfiles.**find** *(name, path)*

find 1st occurrence of file in path

GSASIIfiles.**readColMetadata** *(imagefile)*

Reads image metadata from a column-oriented metadata table (1-ID style .par file). Called by **GetColumnMetadata()**

The .par file has any number of columns separated by spaces. The directory for the file must be specified in Config variable *config_example.Column_Metadata_directory*. As an index to the .par file a second “label file” must be specified with the same file root name as the .par file but the extension must be .XXX_lbls (where .XXX is the extension of the image) or if that is not present extension .lbls.

**Parameters** **imagefile** *(str)* – the full name of the image file (with extension, directory optional)

**Returns**  a dict with parameter values. Named parameters will have the type based on the specified Python function, named columns will be character strings

The contents of the label file will look like this:

```
# define keywords
filename: lambda x,y: "{}_{:0>6}".format(x,y)|33,34
distance: float | 75
wavelength: lambda keV: 12.398425/float(keV)|9
pixelSize: lambda x: [74.8, 74.8]|0
ISOlikeDate: lambda dow,m,d,t,y:"{dow}-{m}-{d}T{t}".format(y,m,d,t,dow)|0,1,2,3,4
```
This file contains three types of lines in any order.

- Named parameters are evaluated with user-supplied Python code (see subsequent information). Specific named parameters are used to determine values that are used for image interpretation (see table, below). Any others are copied to the Comments subsection of the Image tree item.

- Column labels are defined with a column number (integer) followed by a colon (:) and a label to be assigned to that column. All labeled columns are copied to the Image’s Comments subsection.

- Comments are any line that does not contain a colon.

Note that columns are numbered starting at zero.

Any named parameter may be defined provided it is not a valid integer, but the named parameters in the table have special meanings, as described. The parameter name is followed by a colon. After the colon, specify Python code that defines or specifies a function that will be called to generate a value for that parameter.

Note that several keywords, if defined in the Comments, will be found and placed in the appropriate section of the powder histogram(s)’s Sample Parameters after an integration: Temperature, Pressure, Time, FreePrm1, FreePrm2, FreePrm3, Omega, Chi, and Phi.

After the Python code, supply a vertical bar (|) and then a list of one more more columns that will be supplied as arguments to that function.

Note that the labels for the three FreePrm items can be changed by including that label as a third item with an additional vertical bar. Labels will be ignored for any other named parameters.

The examples above are discussed here:

**filename:** `lambda x,y: "{}_{:0>6}".format(x,y) | 33,34` Here the function to be used is defined with a lambda statement:

```python
lambda x,y: "{}_{:0>6}".format(x,y)
```

This function will use the format function to create a file name from the contents of columns 33 and 34. The first parameter (x, col. 33) is inserted directly into the file name, followed by a underscore (_), followed by the second parameter (y, col. 34), which will be left-padded with zeros to six characters (format directive :0>6).

When there will be more than one image generated per line in the .par file, an alternate way to generate list of file names takes into account the number of images generated:

```python
lambda x,y,z: ["{}_{:0>6}".format(x,int(y)+i) for i in range(int(z))]
```

Here a third parameter is used to specify the number of images generated, where the image number is incremented for each image.

**distance:** `float | 75` Here the contents of column 75 will be converted to a floating point number by calling float on it. Note that the spaces here are ignored.
Here we define an algebraic expression to convert an energy in keV to a wavelength and pass the contents of column 9 as that input energy.

In this case the pixel size is a constant (a list of two numbers). The first column is passed as an argument as at least one argument is required, but that value is not used in the expression.

This example defines a parameter that takes items in the first five columns and formats them in a different way. This parameter is not one of the pre-defined parameter names below. Some external code could be used to change the month string (argument \(m\)) to a integer from 1 to 12.

In this example, the contents of column 34 will be converted to an integer and placed as the second free-named parameter in the Sample Parameters after an integration. The label for this parameter will be changed to “Free Parm2 Label”.

### Pre-defined parameter names

<table>
<thead>
<tr>
<th>key-word</th>
<th>required</th>
<th>type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file-name</td>
<td>yes</td>
<td>str or list</td>
<td>generates the file name prefix for the matching image file (MyImage001 for file /tmp/MyImage001.tif) or a list of file names.</td>
</tr>
<tr>
<td>polarization</td>
<td>no</td>
<td>float</td>
<td>generates the polarization expected based on the monochromator angle, defaults to 0.99.</td>
</tr>
<tr>
<td>center</td>
<td>no</td>
<td>list of 2 floats</td>
<td>generates the approximate beam center on the detector in mm, such as [204.8, 204.8].</td>
</tr>
<tr>
<td>distance</td>
<td>yes</td>
<td>float</td>
<td>generates the distance from the sample to the detector in mm.</td>
</tr>
<tr>
<td>pixel-Size</td>
<td>no</td>
<td>list of 2 floats</td>
<td>generates the size of the pixels in microns such as [200.0, 200.0].</td>
</tr>
<tr>
<td>wavelength</td>
<td>yes</td>
<td>float</td>
<td>generates the wavelength in Angstroms</td>
</tr>
</tbody>
</table>

GSASIIfiles.readColMetadataLabels (lblFil)
Read the .*lbs file and setup for metadata assignments

GSASIIfiles.readMasks (filename, masks, ignoreThreshold)
Read a GSAS-II masks file

GSASIIfiles.sfloat (S)
Convert a string to float. An empty field or a unconvertable value is treated as zero

### 1.4.9 GSASIImpsubs - routines used in multiprocessing

The routines here are called either directly when GSAS-II is used without multiprocessing or in separate cores when multiprocessing is used.

These routines are designed to be used in one of two ways:

- when multiprocessing is enabled (see global variable useMP) the computational routines are called in separate Python interpreter that is created and then deleted after use.
- when useMP is False, these routines are called directly from the main “thread”.

Note that GSASIImpsubs.InitMP() should be called before any of the other routines in this module are used.

GSASIImpsubs.ComputePwdrProfCW (profList)
Compute the peaks profile for a set of CW peaks and add into the yc array
GSASIImpsubs.ComputePwdrProfTOF (profList)
Compute the peaks profile for a set of TOF peaks and add into the yc array

GSASIImpsubs.InitFobsSqGlobals (xl, ratio1, shl1, xB1, xF1, im1, lamRatio1, kRatio1, xMask1, Ka21)
Initialize for the computation of Fobs Squared for powder histograms. Puts lots of junk into the global names-
pace in this module.

GSASIImpsubs.InitMP (allowMP=True)
Called to initialize use of Multiprocessing

GSASIImpsubs.InitPwdrProfGlobals (im1, shl1, xl)
Initialize for the computation of Fobs Squared for powder histograms. Puts lots of junk into the global names-
pace in this module.

GSASIImpsubs.ResetMP ()
Call after changing Config var ‘Multiprocessing_cores’ to force a resetting of the useMP from the parameter.

1.4.10 gltext: draw OpenGL text

Routines that render text on OpenGL without use of GLUT.
Code written by Christian Brugger & Stefan Hacker and distributed under GNU General Public License.

class gltext.Text (text='Text', font=None, font_size=8, foreground=<sphinxeextautodocimporter_MockObject object>, centered=False)
A simple class for using System Fonts to display text in an OpenGL scene. The Text adds a global Cache of
already created text elements to TextElement’s base functionality so you can save some memory and increase
speed

centered
Display the text centered
draw_text (position=<sphinxeextautodocimporter_MockObject object>, scale=1.0, rotation=0)
position (wx.Point) - x/y Position to draw in scene scale (float) - Scale rotation (int) - Rotation in degree
Draws the text to the scene

font
Font of the object

font_size
Font size

foreground
Color/Overlay bitmap of the text

getTextElement ()
Returns the text element bound to the Text class

getTexture ()
Returns the texture of the bound TextElement

getTexture_size ()
Returns a texture size tuple

setCentered (value, reinit=True)
value (bool) - New centered value reinit (bool) - Create a new texture
Sets a new value for ‘centered’

1.4. GSAS-II Utility Modules
setFont (value, reinit=True)
  value (bool) - New Font reinit (bool) - Create a new texture
  Sets a new font

setFont_size (value, reinit=True)
  value (bool) - New font size reinit (bool) - Create a new texture
  Sets a new font size

setForeground (value, reinit=True)
  value (bool) - New centered value reinit (bool) - Create a new texture
  Sets a new value for 'centered'

setText (value, reinit=True)
  value (bool) - New Text reinit (bool) - Create a new texture
  Sets a new text

text
  Text of the object

text_element
  TextElement bound to this class

texture
  Texture of bound TextElement

texture_size
  Size of the used texture

class gltext.TextElement (text=", font=None, foreground=<sphinx.ext.autodoc.importer._MockObject object>, centered=False)
  A simple class for using system Fonts to display text in an OpenGL scene

bind ()
  Increase refcount

centered
  Is text centered

createTexture ()
  Creates a texture from the settings saved in TextElement, to be able to use normal system fonts convienently
  a wx.MemoryDC is used to draw on a wx.Bitmap. As wxwidgets device contexts don’t support alpha at all
  it is necessary to apply a little hack to preserve antialiasing without sticking to a fixed background color:

  We draw the bmp in b/w mode so we can use its data as a alpha channel for a solid color bitmap which
  after GL_ALPHA_TEST and GL_BLEND will show a nicely antialiased text on any surface.

  To access the raw pixel data the bmp gets converted to a wx.Image. Now we just have to merge our
  foreground color with the alpha data we just created and push it all into a OpenGL texture and we are
  DONE inhalesdelpy

  DRAWBACK of the whole conversion thing is a really long time for creating the texture. If you see any
  optimizations that could save time PLEASE CREATE A PATCH!!!

deleteTexture ()
  Deletes the OpenGL texture object

draw_text (position=<sphinx.ext.autodoc.importer._MockObject object>, scale=1.0, rotation=0)
  position (wx.Point) - x/y Position to draw in scene scale (float) - Scale rotation (int) - Rotation in degree
  Draws the text to the scene
1.4.11 **ElementTable: Periodic Table Data**

Element table data for building periodic table with valences & JMOL colors. Need these in case we go back to this periodic table coloring scheme.

Defines list ElTable which contains all defined oxidation states for each element, the location in the table, an element name, a color, a size and a second color.

1.4.12 **FormFactors: Scattering Data**


1.4.13 **ImageCalibrants: Calibration Standards**

GSASII powder calibrants as a dictionary ImageCalibrants.Calibrants with substances commonly used for powder calibrations for image data.

Each entry in ImageCalibrants consists of:

```
'key':([Bravais num,],[space group,],[a,b,c,alpha,beta,gamma,],no. lines skipped,
    (dmin,pixLimit,cutOff),(absent list))
(The space group may be an empty string)
the absent list is optional; it gives indices of lines that have no intensity despite being allowed - see the Si example below;

```

as an example:
or where “Bravais num” and “(a,b,..)” are repeated in the case of mixtures:

To expand this list with locally needed additions, do not modify this file, because you may lose these changes during a software update. Instead duplicate the format of this file in a file named UserCalibrants.py and there define the material(s) you want:

```python
Calibrants=
    # LaB6 skip 2 lines':([2,],[''],[(4.1569162,4.1569162,4.1569162,90,90,90),],2,(1.0,10,10),()),
```

New key values will be added to the list of options. If a key is duplicated, the information in UserCalibrants.py will override the information in this file.

**Note, the Bravais numbers are:**

- 0 F cubic
- 1 I cubic
- 2 P cubic
- 3 R hexagonal (trigonal not rhombohedral)
- 4 P hexagonal
- 5 I tetragonal
- 6 P tetragonal
- 7 F orthorhombic
- 8 I orthorhombic
- 9 C orthorhombic
- 10 P orthorhombic
- 11 C monoclinic
- 12 P monoclinic
- 13 P triclinic

### 1.4.14 atmdatat: Table of atomic data

The entries here are:

- **XrayFF**: a dict of form factor coefficients
- **AtmSize**: atom Sizes, bond radii, angle radii, H-bond radii
- **AtmBlens**: atom masses & neutron scattering length (b,b'), sig(incoh) @ 1A
- **MagFF**: neutron magnetic form factor coeff: M for j<0> & N for j<2>

Sources:


Neutron anomalous coeff (LS) from fitting Lynn & Seeger, At. Data & Nuc. Data Tables, 44, 191-207(1990)

O2- x-ray scattering factor from Tokonami (1965) Acta Cryst 19, 486

At wts from 14th ed Nuclides & Isotopes, 1989 GE Co.

1.4.15 defaultIparms: Table of instrument parameters

Defines some default instrument parameters. Format for each is a list of strings finished with a ‘ ‘. Begin with ‘#GSAS-II . . . ’ as the reader routine checks this. Each line can be comprised of a block of ‘;’ delimited name:value pairs. All
instrument parameters must be included; even those = 0. Use a GSAS-II instprm file as a source for the entries.

For a new entry:

Append a useful name to defaultIparms_lbl.

Append the list of lines to defaultIparms.

defaultIparm_lbl: defines a list of labels.

defaultIparms: defines a list of multiple strings with values for each set of defaults.

defaultIparms.
defaultIparms = [
    'gsas-ii instrument parameter file for lab CuKa data
    Type:PXC;Bank:1
    \ldots
    alpha:1.0
    beta-0:0.;beta-1:0.005;beta-q:0.0
    sig-0:0.;sig-1:-50.;sig-2:0.;sig-q:0.0
    X:0.;Y:0.;Z:0.
']

the phrase ‘lab data’ in the following makes the default goniometer type='Bragg-Brentano’ otherwise it is set to
‘Debye-Scherrer’

1.5 GSAS-II GUI Support Routines

1.5.1 GSASIICtrlGUI: Custom GUI controls

A library of GUI controls for reuse throughout GSAS-II, as indexed below

<table>
<thead>
<tr>
<th>Class or function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnumSelector</td>
<td>A combo box with a built-in call back routine that automatically sets a dict or list entry.</td>
</tr>
<tr>
<td>DisAgl1Dialog</td>
<td>Distance/Angle Controls input dialog.</td>
</tr>
<tr>
<td>FlagSetDialog</td>
<td>Dialog that provides a table of items along with a checkbox for each.</td>
</tr>
<tr>
<td>G2ChoiceButton</td>
<td>A customized wx.Choice that automatically initializes to the initial value and saves the choice.</td>
</tr>
<tr>
<td>G2CheckBox</td>
<td>A customized wx.CheckBox that automatically initializes to the initial value and saves the choice.</td>
</tr>
<tr>
<td>G2Column1DDialog</td>
<td>A dialog for matching column data to desired items; some columns may be ignored.</td>
</tr>
<tr>
<td>G2HistogramDialog</td>
<td>A dialog for global edits to histogram data globally</td>
</tr>
<tr>
<td>G2MultiChoiceDialog</td>
<td>Dialog similar to wx.MultiChoiceDialog, but provides a filter to select choices and buttons to move to other choices.</td>
</tr>
<tr>
<td>G2SingleChoiceDialog</td>
<td>Dialog similar to wx.SingleChoiceDialog, but provides a filter to help search through choices.</td>
</tr>
<tr>
<td>HelpButton</td>
<td>Creates a button labeled with a “?” that when pressed displays help text in a modal message window.</td>
</tr>
<tr>
<td>MultiColumnSelection</td>
<td>A dialog that builds a multicolumn table, word wrapping is used for the 2nd, 3rd,... columns.</td>
</tr>
<tr>
<td>MultiDataDialog</td>
<td>Dialog to obtain multiple data values from user, with optional range validation; items can be float, str or bool.</td>
</tr>
<tr>
<td>MultiIntegerDialog</td>
<td>Dialog to obtain multiple integer values from user, with a description for each value and optional default values.</td>
</tr>
<tr>
<td>MultiStringDialog</td>
<td>Dialog to obtain multiple string values from user, with a description for each value and optional default values.</td>
</tr>
<tr>
<td>OrderBox</td>
<td>Creates a wx.Panel with scrollbars where items can be ordered into columns.</td>
</tr>
<tr>
<td>ScrolledMultiEditor</td>
<td>wx.Dialog for editing many dict- or list-contained items. with validation. Results are placed into a dict or list.</td>
</tr>
<tr>
<td>SGMagSpinBox</td>
<td>Special version of MessageBox that displays magnetic spin text.</td>
</tr>
<tr>
<td>SGMessageBox</td>
<td>Special version of MessageBox that displays space group &amp; super space group text in two block dialog.</td>
</tr>
<tr>
<td>SingleFloatDialog</td>
<td>Dialog to obtain a single float value from user, with optional range validation.</td>
</tr>
<tr>
<td>SingleIntDialog</td>
<td>Dialog to obtain a single integer value from user, with optional range validation.</td>
</tr>
<tr>
<td>SingleStringDialog</td>
<td>Dialog to obtain a single string value from user, with optional an optional default value.</td>
</tr>
<tr>
<td>ValidatedTextCtrl</td>
<td>A text control with a built-in call back routine to set dict or list elements. Optionally validated values.</td>
</tr>
<tr>
<td>CallScrolledMultiEditor()</td>
<td>Routine for editing many dict- or list-contained items. using the ScrolledMultiEditor dialog.</td>
</tr>
<tr>
<td>Define_wxId()</td>
<td>Create a unique wx.Id symbol in _initMenus in GSASIIdataGUI. Such symbols are needed to create unique menu items.</td>
</tr>
<tr>
<td>G2MessageBox()</td>
<td>Displays text typically used for errors or warnings.</td>
</tr>
<tr>
<td>GetItemOrder()</td>
<td>Creates a dialog for ordering items into columns.</td>
</tr>
<tr>
<td>GetImportFile()</td>
<td>Gets one ore more file from the appropriate import directory, which can be overridden. Arguments follow those of wx.FileDialog().</td>
</tr>
<tr>
<td>HorizontalLine()</td>
<td>Places a line in a Frame or Dialog to separate sections.</td>
</tr>
<tr>
<td>ItemSelector()</td>
<td>Select a single item or multiple items from list of choices. Creates and then destroys a wx.Dialog dialog.</td>
</tr>
<tr>
<td>SelectEdit1Var()</td>
<td>Select a variable from a list, then edit it and select histograms to copy it to.</td>
</tr>
</tbody>
</table>
Other miscellaneous routines that may be of use:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>askSaveFile()</td>
<td>Get a file name from user</td>
</tr>
<tr>
<td>askSaveDirectory()</td>
<td>Get a directory name from user</td>
</tr>
</tbody>
</table>

Other miscellaneous routines that may be of use:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StripIndents()</td>
<td>Regularizes the indentation from a string with multiple newline characters by removing spaces at the beginning of each line.</td>
</tr>
<tr>
<td>StripUnicode()</td>
<td>Removes unicode characters from strings</td>
</tr>
<tr>
<td>GetImportPath()</td>
<td>Determines the default location to use for importing files. Tries sequentially G2frame.TutorialImportDir, config var Import_directory and G2frame.LastImportDir.</td>
</tr>
<tr>
<td>GetExportPath()</td>
<td>Determines the default location to use for writing files. Tries sequentially G2frame.LastExportDir and G2frame.LastGPXdir.</td>
</tr>
</tbody>
</table>

Documentation for all the routines in module GSASIIctrlGUI.

```python
class GSASIIctrlGUI.ASCIIValidator(result=\None, key=\None)
    A validator to be used with a TextCtrl to prevent entering characters other than ASCII characters.

    The value is checked for validity after every keystroke If an invalid number is entered, the box is highlighted. If the number is valid, it is saved in result[key].

    Parameters
    • result (dict/list) – List or dict where value should be placed when valid
    • key (any) – key to use for result (int for list)

    Clone()
    Create a copy of the validator, a strange, but required component

    OnChar(event)
    Called each type a key is pressed ignores keys that are not allowed for int and float types

    TestValid(tc)
    Check if the value is valid by casting the input string into ASCII.
    Save it in the dict/list where the initial value was stored

    Parameters tc (wx.TextCtrl) – A reference to the TextCtrl that the validator is associated with.

    TransferFromWindow()
    Needed by validator, strange, but required component

    TransferToWindow()
    Needed by validator, strange, but required component
```

GSASIIctrlGUI.CallScrolledMultiEditor(parent, dictlst, elemlst, prelbl=[], postlbl=[], title='Edit items', header='', size=(300, 250), CopyButton=False, ASCIIonly=False, **kw)

Shell routine to call a ScrolledMultiEditor dialog. See ScrolledMultiEditor for parameter definitions.

Returns True if the OK button is pressed; False if the window is closed with the system menu or the Cancel button.
GSAS-II Developers Documentation, Release version 4249

GSASIIctrlGUI.Define_wxId (*args)
routine to create unique global wx Id symbols in this module.

class GSASIIctrlGUI.DisAglDialog (parent, data, default, Reset=True, Angle=True)
Distance/Angle Controls input dialog. After ShowModal() returns, the results are found in dict self.data, which is accessed using GetData().

Parameters
• parent (wx.Frame) – reference to parent frame (or None)
• data (dict) – a dict containing the current search ranges or an empty dict, which causes default values to be used. Will be used to set element DisAgICls in Phase Tree Item
• default (dict) – A dict containing the default search ranges for each element.
• Reset (bool) – if True (default), show Reset button
• Angle (bool) – if True (default), show angle radii

Draw (data)
Creates the contents of the dialog. Normally called by __init__().

GetData ()
Returns the values from the dialog

OnOk (event)
Called when the OK button is pressed

OnReset (event)
Called when the Reset button is pressed

class GSASIIctrlGUI.EnumSelector (parent, dct, item, choices, values=None, **kw)
A customized wxpython.ComboBox that selects items from a list of choices, but sets a dict (list) entry to the corresponding entry from the input list of values.

Parameters
• parent (wx.Panel) – the parent to the ComboBox (usually a frame or panel)
• dct (dict) – a dict (or list) to contain the value set for the ComboBox.
• item – the dict key (or list index) where dct[item] will be set to the value selected in the ComboBox. Also, dct[item] contains the starting value shown in the widget. If the value does not match an entry in values, the first value in choices is used as the default, but dct[item] is not changed.
• choices (list) – a list of choices to be displayed to the user such as

    ["default","option 1","option 2",]

Note that these options will correspond to the entries in values (if specified) item by item.

• values (list) – a list of values that correspond to the options in choices, such as

    [0,1,2]

The default for values is to use the same list as specified for choices.

• (other) – additional keyword arguments accepted by ComboBox can be specified.

class GSASIIctrlGUI.FlagSetDialog (parent, title, colnames, rownames, flags)
Creates popup with table of variables to be checked for e.g. refinement flags
class GSASIIctrlGUI.G2CheckBox (parent, label, loc, key, OnChange=None)
A customized version of a CheckBox that automatically initializes the control to a supplied list or dict entry and updates that entry as the widget is used.

Parameters

- **parent (wx.Panel)** – name of panel or frame that will be the parent to the widget. Can be None.
- **label (str)** – text to put on check button
- **loc (dict/list)** – the dict or list with the initial value to be placed in the CheckBox.
- **key (int/str)** – the dict key or the list index for the value to be edited by the CheckBox. The loc[key] element must exist. The CheckBox will be initialized from this value. If the value is anything other that True (or 1), it will be taken as False.
- **OnChange (function)** – specifies a function or method that will be called when the CheckBox is changed (Default is None). The called function is supplied with one argument, the calling event.

class GSASIIctrlGUI.G2ChoiceButton (parent, choiceList, indLoc=None, indKey=None, strLoc=None, strKey=None, onChoice=None, **kwargs)
A customized version of a wx.Choice that automatically initializes the control to match a supplied value and saves the choice directly into an array or list. Optionally a function can be called each time a choice is selected. The widget can be used with an array item that is set to to the choice by number (indLoc[indKey]) or by string value (strLoc[strKey]) or both. The initial value is taken from indLoc[indKey] if not None or strLoc[strKey] if not None.

Parameters

- **parent (wx.Panel)** – name of panel or frame that will be the parent to the widget. Can be None.
- **choiceList (list)** – a list or tuple of choices to offer the user.
- **indLoc (dict/list)** – a dict or list with the initial value to be placed in the Choice button. If this is None, this is ignored.
- **indKey (int/str)** – the dict key or the list index for the value to be edited by the Choice button. If indLoc is not None then this must be specified and the indLoc[indKey] will be set. If the value for indLoc[indKey] is not None, it should be an integer in range(len(choiceList)). The Choice button will be initialized to the choice corresponding to the value in this element if not None.
- **strLoc (dict/list)** – a dict or list with the string value corresponding to indLoc[indKey]. Default (None) means that this is not used.
- **strKey (int/str)** – the dict key or the list index for the string value The strLoc[strKey] element must exist or strLoc must be None (default).
- **onChoice (function)** – name of a function to call when the choice is made.

class GSASIIctrlGUI.G2ColumnIDDiallog (parent, title, header, Comments, ChoiceList, ColumnData, monoFont=False, **kw)
A dialog for matching column data to desired items; some columns may be ignored.

Parameters

- **ParentFrame (wx.Frame)** – reference to parent frame
- **title (str)** – heading above list of choices
- **header (str)** – Title to place on window frame
• **ChoiceList** *(list)* – a list of possible choices for the columns

• **ColumnData** *(list)* – lists of column data to be matched with ChoiceList

• **monoFont** *(bool)* – If False (default), use a variable-spaced font; if True use a equally-spaced font.

• **kw** – optional keyword parameters for the wx.Dialog may be included such as size [which defaults to 
(320, 310)] and style (which defaults to `wx.DEFAULT_DIALOG_STYLE | wx.RESIZE_BORDER | wx.CENTRE | wx.OK | wx.CANCEL); note that wx.OK and wx.CANCEL controls the presence of the eponymous buttons in the dialog.

**Returns** the name of the created dialog

**GetSelection()**

Returns the selected sample parm for each column

```python
class GSASIIctrlGUI.G2HistoDataDialog(parent, title, header, ParmList, ParmFmt, HistoList, ParmData, monoFont=False, **kw)
```

A dialog for editing histogram data globally.

**Parameters**

• **ParentFrame** *(wx.Frame)* – reference to parent frame

• **title** *(str)* – heading above list of choices

• **header** *(str)* – Title to place on window frame

• **ParmList** *(list)* – a list of names for the columns

• **ParmFmt** *(list)* – a list of formatting strings for the columns

• **list** – HistoList: a list of histogram names

• **ParmData** *(list)* – a list of lists of data matched to ParmList; one for each item in HistoList

• **monoFont** *(bool)* – If False (default), use a variable-spaced font; if True use a equally-spaced font.

• **kw** – optional keyword parameters for the wx.Dialog may be included such as size [which defaults to 
(320, 310)] and style (which defaults to `wx.DEFAULT_DIALOG_STYLE | wx.RESIZE_BORDER | wx.CENTRE | wx.OK | wx.CANCEL); note that wx.OK and wx.CANCEL controls the presence of the eponymous buttons in the dialog.

**Returns** the modified ParmData

**GetData()**

Returns the modified ParmData

```python
class GSASIIctrlGUI.G2HtmlWindow(parent, *args, **kwargs)
```

Displays help information in a primitive HTML browser type window

```python
class GSASIIctrlGUI.G2LoggedButton(parent, id=<sphinx.ext.autodoc.importer._MockObject object>, label=", locationcode=", handler=None, *args, **kwargs)
```

A version of wx.Button that creates logging events. Bindings are saved in the object, and are looked up rather than directly set with a bind. An index to these buttons is saved as log.ButtonBindingLookup :param wx.Panel parent: parent widget :param int id: Id for button :param str label: label for button :param str locationcode: a label used internally to uniquely indentify the button :param function handler: a routine to call when the button is pressed

```python
onPress(event)
```

create log event and call handler
GSASIIctrlGUI.G2MessageBox (parent, msg, title='Error')
Simple code to display a error or warning message

class GSASIIctrlGUI.G2MultiChoiceDialog (parent, title, header, ChoiceList, toggle=True, monoFont=False, filterBox=True, extraOpts={}, **kw)
A dialog similar to MultiChoiceDialog except that buttons are added to set all choices and to toggle all choices.

Parameters

- **ParentFrame** (wx.Frame) – reference to parent frame
- **title** (str) – heading above list of choices
- **header** (str) – Title to place on window frame
- **ChoiceList** (list) – a list of choices where one more will be selected
- **toggle** (bool) – If True (default) the toggle and select all buttons are displayed
- **monoFont** (bool) – If False (default), use a variable-spaced font; if True use a equally-spaced font.
- **filterBox** (bool) – If True (default) an input widget is placed on the window and only entries matching the entered text are shown.
- **extraOpts** (dict) – a dict containing a entries of form label_i and value_i with extra options to present to the user, where value_i is the default value. Options are listed ordered by the value_i values.
- **kw** – optional keyword parameters for the wx.Dialog may be included such as size [which defaults to (320,310)] and style (which defaults to wx.DEFAULT_DIALOG_STYLE|wx.RESIZE_BORDER|wx.CENTRE|wx.OK|wx.CANCEL); note that wx.OK and wx.CANCEL style items control the presence of the eponymous buttons in the dialog.

Returns

the name of the created dialog

Filter (event)
Read text from filter control and select entries that match. Called by Timer after a delay with no input or if Enter is pressed.

GetSelections ()
Returns a list of the indices for the selected choices

OnCheck (event)
for CheckListBox events; if Set Range is in use, this sets/clears all entries in range between start and end according to the value in start. Repeated clicks on the start change the checkbox state, but do not trigger the range copy. The caption next to the button is updated on the first button press.

SetRange (event)
Respond to a press of the Set Range button. Set the range flag and the caption next to the button

SetSelections (selList)
Sets the selection indices in selList as selected. Resets any previous selections for compatibility with wx.MultiChoiceDialog. Note that the state for only the filtered items is shown.

Parameters selList (list) – indices of items to be selected. These indices are referenced to the order in self.ChoiceList

onChar (event)
Respond to keyboard events in the Filter box
class GSASIIctrlGUI.G2SingleChoiceDialog(parent, title, header, ChoiceList, monoFont=False, filterBox=True, **kw)
A dialog similar to wx.SingleChoiceDialog except that a filter can be added.

Parameters

- **ParentFrame** *(wx.Frame)* – reference to parent frame
- **title** *(str)* – heading above list of choices
- **header** *(str)* – Title to place on window frame
- **ChoiceList** *(list)* – a list of choices where one will be selected
- **monoFont** *(bool)* – If False (default), use a variable-spaced font; if True use a equally-spaced font.
- **filterBox** *(bool)* – If True (default) an input widget is placed on the window and only entries matching the entered text are shown.
- **kw** – optional keyword parameters for the wx.Dialog may be included such as size [which defaults to (320,310)] and style (which defaults to wx.DEFAULT_DIALOG_STYLE | wx.RESIZE_BORDER | wx.CENTRE | wx.OK | wx.CANCEL); note that wx.OK and wx.CANCEL controls the presence of the eponymous buttons in the dialog.

Returns
the name of the created dialog

GetSelection()
Returns the index of the selected choice

class GSASIIctrlGUI.G2TreeCtrl (parent=None, *args, **kwargs)
Create a wrapper around the standard TreeCtrl so we can “wrap” various events.

This logs when a tree item is selected (in onSelectionChanged())

This also wraps lists and dicts pulled out of the tree to track where they were retrieved from.

ConvertRelativeHistNum (histtype, histnum)
Converts a histogram type and relative histogram number to a histogram name in the current project

ConvertRelativePhaseNum (phasenum)
Converts relative phase number to a phase name in the current project

GetImageLoc (TreeId)
Get Image data from the Tree. Handles cases where the image name is specified, as well as where the image file name is a tuple containing the image file and an image number

GetRelativeHistNum (histname)
Returns list with a histogram type and a relative number for that histogram, or the original string if not a histogram

GetRelativePhaseNum (phasename)
Returns a phase number if the string matches a phase name or else returns the original string

RestoreExposedItems()
Traverse the top level tree items and restore exposed (expanded) tree items back to their previous state (done after a reload of the tree after a refinement)

SaveExposedItems()
Traverse the top level tree items and save names of exposed (expanded) tree items. Done before a refinement.

UpdateImageLoc (TreeId, imagefile)
Saves a new imagefile name in the Tree. Handles cases where the image name is specified, as well as where the image file name is a tuple containing the image file and an image number
class GSASIIctrlGUI.GSGrid(parent, name="")
   Basic wx.Grid implementation

InstallGridToolTip(rowcolhintcallback, collblCallback=None, rowLblCallback=None)
   code to display a tooltip for each item on a grid from http://wiki.wxpython.org/wxGrid%20ToolTips
   (buggy!), expanded to column and row labels using hints from https://groups.google.com/forum/#!topic/wxPython-users/bm8OARRVDCs

Parameters

   • rowcolhintcallback (function) – a routine that returns a text string depending on the selected row and column,
     to be used in explaining grid entries.

   • collblCallback (function) – a routine that returns a text string depending on the selected column,
     to be used in explaining grid columns (if None, the default), column labels do not get a tooltip.

   • rowLblCallback (function) – a routine that returns a text string depending on the selected row,
     to be used in explaining grid rows (if None, the default), row labels do not get a tooltip.

class GSASIIctrlGUI.GSNoteBook(parent, name="", size=None, style=None)
   Notebook used in various locations; implemented with wx.aui extension

GSASIIctrlGUI.GetConfigValsDocs()
   Reads the module referenced in fname (often <module>.__file__) and return a dict with names of global vari-
   ables as keys. For each global variable, the value contains four items:

   Returns

   a dict where keys are names defined in module config_example.py where the value is a list of four items, as follows:

   • item 0: the default value
   • item 1: the current value
   • item 2: the initial value (starts same as item 1)
   • item 3: the “docstring” that follows variable definition

GSASIIctrlGUI.GetExportPath(G2frame)
   Determines the default location to use for writing files. Tries sequentially G2frame.LastExportDir and
   G2frame.LastGPXdir.

   Returns a string containing the path to be used when writing files or ‘.’ if none of the above are
   specified.

GSASIIctrlGUI.GetImportFile(G2frame, message, defaultDir="", defaultFile="",
   style=<sphinx.ext.autodoc.importer._MockObject object>, parent=None, *args, **kwargs)

   Uses a customized dialog that gets files from the appropriate import directory. Arguments are used the same as
   in wx.FileDialog(). Selection of multiple files is allowed if argument style includes wx.FD_MULTIPLE.

   The default initial directory (unless overridden with argument defaultDir) is found in
   G2frame.TutorialImportDir, config setting Import_directory or G2frame.LastImportDir, see
   GetImportPath().

   The path of the first file entered is used to set G2frame.LastImportDir and optionally config setting Im-
   port_directory.

   Returns a list of files or an empty list
GSASIIctrlGUI.GetImportPath(G2frame)
Determines the default location to use for importing files. Tries sequentially G2frame.TutorialImportDir, config var Import_directory and G2frame.LastImportDir.

Returns a string containing the path to be used when reading files or None if none of the above are specified.

GSASIIctrlGUI.GetItemOrder(parent, keylist, vallookup, posdict)
Creates a dialog where items can be ordered into columns

Parameters

• keylist(list) – is a list of keys for column assignments
• vallookup(dict) – is a dict keyed by names in keylist where each item is a dict. Each inner dict contains variable names as keys and their associated values
• posdict(dict) – is a dict keyed by names in keylist where each item is a dict. Each inner dict contains column numbers as keys and their associated variable name as a value. This is used for both input and output.

class GSASIIctrlGUI.GridFractionEditor(grid)
A grid cell editor class that allows entry of values as fractions as well as sine and cosine values [as s() and c()]

ApplyEdit(row, col, grid)
Called only in wx >= 2.9 Save the value of the control into the grid if EndEdit() returns as True

class GSASIIctrlGUI.HelpButton(parent, msg='', helpIndex='')
Create a help button that displays help information. The text can be displayed in a modal message window or it can be a reference to a location in the gsasII.html help web page, in which case that page is opened in a web browser.

TODO: it might be nice if it were non-modal: e.g. it stays around until the parent is deleted or the user closes it, but this did not work for me.

Parameters

• parent – the panel/frame where the button will be placed
• msg(str) – the help text to be displayed
• helpIndex(str) – location of the help information in the gsasII.html help file in the form of an anchor string. The URL will be constructed from: location + gsasII.html + “#” + helpIndex

GSASIIctrlGUI.HorizontalLine(sizer, parent)
Draws a horizontal line as wide as the window. This shows up on the Mac as a very thin line, no matter what I do

GSASIIctrlGUI.ItemSelector(ChoiceList, ParentFrame=None, title='Select an item', size=None, header='Item Selector', useCancel=True, multiple=False)
Provide a wx dialog to select a single item or multiple items from list of choices

Parameters

• ChoiceList(list) – a list of choices where one will be selected
• ParentFrame(wx.Frame) – Name of parent frame (default None)
• title(str) – heading above list of choices (default ‘Select an item’)
• size(wx.Size) – Size for dialog to be created (default None – size as needed)
• header(str) – Title to place on window frame (default ‘Item Selector’)
• useCancel(bool) – If True (default) both the OK and Cancel buttons are offered
• **multiple** *(bool)* – If True then multiple items can be selected (default False)

**Returns** the selection index or None or a selection list if multiple is true

Called by GSASIIdataGUI.OnReOrgSelSeq() Which is not fully implemented.

```python
class GSASIIctrlGUI.MultiColumnSelection(
    parent, title, colLabels, choices, colWidths,
    checkLbl='', height=400, centerCols=False,
    *args, **kw)
```

Defines a Dialog widget that can be used to select an item from a multicolumn list. The first column should be short, but remaining columns are word-wrapped if the length of the information extends beyond the column.

When created, the dialog will be shown and `<dlg>.Selection` will be set to the index of the selected row, or -1. Be sure to use `<dlg>.Destroy()` to remove the window after reading the selection. If the dialog cannot be shown because a very old version of wxPython is in use, `<dlg>.Selection` will be None.

If checkLbl is provided with a value, then a set of check buttons starts the table and `<dlg>.Selections` has the checked rows.

**Parameters**

- **parent** *(wx.Frame)* – the parent frame (or None)
- **title** *(str)* – A title for the dialog window
- **colLabels** *(list)* – labels for each column
- **choices** *(list)* – a nested list with a value for each row in the table. Within each value should be a list of values for each column. There must be at least one value, but it is OK to have more or fewer values than there are column labels (colLabels). Extra are ignored and unspecified columns are left blank.
- **colWidths** *(list)* – a list of int values specifying the column width for each column in the table (pixels). There must be a value for every column label (colLabels).
- **checkLbl** *(str)* – A label for a row of checkboxes added at the beginning of the table
- **height** *(int)* – an optional height (pixels) for the table (defaults to 400)
- **centerCols** *(bool)* – if True, items in each column are centered. Default is False

Example use:

```python
lbls = ('col 1', 'col 2', 'col 3')
choices=[['test1', 'explanation of test 1'],
         ['b', 'a really really long line that will be word-wrapped'],
         ['test3', 'more explanation text', 'optional 3rd column text'])
colWidths=[200, 400, 100]
dlg = MultiColumnSelection(frm, 'select tutorial', lbls, choices, colWidths)
value = choices[dlg.Selection][0]
dlg.Destroy()
```

```python
class GSASIIctrlGUI.MultiDataDialog(
    parent, title, prompts, values, limits=[0.0, 1.0], formats=['%.5g'])
```

Dialog to obtain multiple values from user

```python
class GSASIIctrlGUI.MultiIntegerDialog(
    parent, title, prompts, values)
```

Input a series of integers based on prompts

```python
class GSASIIctrlGUI.MultiStringDialog(
    parent, title, prompts, values=[], size=-1)
```

Dialog to obtain a multi string values from user

**Parameters**

- **parent** *(wx.Frame)* – name of parent frame
• **title** *(str)* – title string for dialog

• **prompts** *(str)* – strings to tell use what they are inputting

• **values** *(str)* – default input values, if any

• **size** *(int)* – length of the input box in pixels

**GetValues** ()

Use this method to get the value entered by the user :returns: string entered by user

**Show** ()

Use this method after creating the dialog to post it :returns: True if the user pressed OK; False if the User pressed Cancel

class **GSASIIctrlGUI.MyHelp** *(frame, includeTree=False, morehelpitems=[])*

A class that creates the contents of a help menu. The menu will start with two entries:

• ‘Help on <helpType>’: where helpType is a reference to an HTML page to be opened

• About: opens an About dialog using OnHelpAbout. N.B. on the Mac this gets moved to the App menu to be consistent with Apple style.

NOTE: for this to work properly with respect to system menus, the title for the menu must be &Help, or it will not be processed properly:

```python
menu.Append(menu=MyHelp(self,...),title="&Help")
```

**OnCheckUpdates** *(event)*

Check if the GSAS-II repository has an update for the current source files and perform that update if requested.

**OnHelpAbout** *(event)*

Display an ‘About GSAS-II’ box

**OnHelpById** *(event)*

Called when Help on... is pressed in a menu. Brings up a web page for documentation. Uses the helpKey value from the dataWindow window unless a special help key value has been defined for this menu id in self.HelpById

Note that self should now (2frame) be child of the main window (G2frame)

**OnSelectVersion** *(event)*

Allow the user to select a specific version of GSAS-II

class **GSASIIctrlGUI.MyHtmlPanel** *(frame, newId)*

Defines a panel to display HTML help information, as an alternative to displaying help information in a web browser.

class **GSASIIctrlGUI.NumberValidator** *(typ, positiveonly=False, min=None, max=None, exclLim=[False, False], result=None, key=None, OKcontrol=None, CIFinput=False)*

A validator to be used with a TextCtrl to prevent entering characters other than digits, signs, and for float input, a period and exponents.

**The value is checked for validity after every keystroke** If an invalid number is entered, the box is highlighted. If the number is valid, it is saved in result[key]

**Parameters**

• **typ** *(type)* – the base data type. Must be int or float.
• **positiveonly** *(bool)* – If True, negative integers are not allowed (default False). This prevents the + or - keys from being pressed. Used with typ=int; ignored for typ=float.

• **min**(number) – Minimum allowed value. If None (default) the lower limit is unbounded

• **max**(number) – Maximum allowed value. If None (default) the upper limit is unbounded

• **exclLim**(list) – if True exclude min/max value ([exclMin,exclMax]); (Default=[False,False])

• **result**(dict/list) – List or dict where value should be placed when valid

• **key**(any) – key to use for result (int for list)

• **OKcontrol**(function) – function or class method to control an OK button for a window. Ignored if None (default)

• **CIFinput**(bool) – allows use of a single ‘?’ or ‘.’ character as valid input.

**CheckInput**(previousInvalid)
called to test every change to the TextCtrl for validity and to change the appearance of the TextCtrl
Anytime the input is invalid, call self.OKcontrol (if defined) because it is fast. If valid, check for any other invalid entries only when changing from invalid to valid, since that is slower.

**Parameters** previousInvalid*(bool)* – True if the TextCtrl contents were invalid prior to the current change.

**Clone**( )
Create a copy of the validator, a strange, but required component

**OnChar**(event)
Called each type a key is pressed ignores keys that are not allowed for int and float types

**ShowValidity**(tc)
Set the control colors to show invalid input

**Parameters** tc*(wx.TextCtrl)* – A reference to the TextCtrl that the validator is associated with.

**TestValid**(tc)
Check if the value is valid by casting the input string into the current type.

Set the invalid variable in the TextCtrl object accordingly.

If the value is valid, save it in the dict/list where the initial value was stored, if appropriate.

**Parameters** tc*(wx.TextCtrl)* – A reference to the TextCtrl that the validator is associated with.

**TransferFromWindow**( )
Needed by validator, strange, but required component

**TransferToWindow**( )
Needed by validator, strange, but required component

**class** GSASIIctrlGUI.OpenTutorial**(parent)**
Open a tutorial web page, optionally copying the web page, screen images and data file(s) to the local disk.

**ChooseTutorial**(choices)
choose a tutorial from a list (will eventually only be used with very old wxPython

**ChooseTutorial2**(choices)
Select tutorials from a two-column table, when possible
**DownloadAll**(event)
Download or update all tutorials

**SelectAndDownload**(event)
Make a list of all tutorials on web and allow user to choose one to download and then view

**SelectDownloadLoc**(event)
Select a download location, Cancel resets to the default

**SetTutorialPath**()
Get the tutorial location if set; if not pick a default directory in a logical place

**UpdateDownloaded**(event)
Find the downloaded tutorials and run an svn update on them

**onSelectDownloaded**(event)
Select a previously downloaded tutorial

**onWebBrowse**(event)
Make a list of all tutorials on web and allow user to view one.

**class** **GSASIIctrlGUI.OrderBox**(parent, keylist, vallookup, posdict, *arg, **kw)
Creates a panel with scrollbars where items can be ordered into columns

**Parameters**

- **keylist**(list) – is a list of keys for column assignments
- **vallookup**(dict) – is a dict keyed by names in keylist where each item is a dict. Each inner dict contains variable names as keys and their associated values
- **posdict**(dict) – is a dict keyed by names in keylist where each item is a dict. Each inner dict contains column numbers as keys and their associated variable name as a value. This is used for both input and output.

**OnChoice**(event)
Called when a column is assigned to a variable

**class** **GSASIIctrlGUI.PickTwoDialog**(parent, title, prompt, names, choices)
This does not seem to be in use

**class** **GSASIIctrlGUI.SGMagSpinBox**(parent, title, text, table, Cents, names, spins, ifGray)
Special version of MessageBox that displays magnetic spin text

**Show**()
Use this method after creating the dialog to post it

**class** **GSASIIctrlGUI.SGMessageBox**(parent, title, text, table, spins=[])
Special version of MessageBox that displays space group & super space group text in two blocks

**Show**()
Use this method after creating the dialog to post it

**GSASIIctrlGUI.SaveConfigVars**(vars, parent=None)
Write the current config variable values to config.py

**Params dict vars** a dictionary of variable settings and meanings as created in **GetConfigValsDocs()**.

**Parameters**

- **parent** – wx.Frame object or None (default) for parent of error message if no file can be written.

**Returns**
True if unable to write the file, None otherwise
Define a window for editing a potentially large number of dict- or list-contained values with validation for each item. Edited values are automatically placed in their source location. If invalid entries are provided, the TextCtrl is turned yellow and the OK button is disabled.

The type for each TextCtrl validation is determined by the initial value of the entry (int, float or string). Float values can be entered in the TextCtrl as numbers or also as algebraic expressions using operators + - / * () and **, in addition pi, sind(), cosd(), tand(), and sqrt() can be used, as well as abbreviations s(), sin(), c(), cos(), t(), tan() and sq().

Parameters

- **parent** *(wx.Frame)* – name of parent window, or may be None
- **dictlst** *(tuple)* – a list of dicts or lists containing values to edit
- **elemlst** *(tuple)* – a list of keys for each item in a dictlst. Must have the same length as dictlst.
- **parent** – name of parent window, or may be None
- **prelbl** *(tuple)* – a list of labels placed before the TextCtrl for each item (optional)
- **postlbl** *(tuple)* – a list of labels placed after the TextCtrl for each item (optional)
- **title** *(str)* – a title to place in the frame of the dialog
- **header** *(str)* – text to place at the top of the window. May contain new line characters.
- **size** *(wx.Size)* – a size parameter that dictates the size for the scrolled region of the dialog. The default is (300,250).
- **CopyButton** *(bool)* – if True adds a small button that copies the value for the current row to all fields below (default is False)
- **ASCIIonly** *(bool)* – if set as True will remove unicode characters from strings
- **minvals** *(list)* – optional list of minimum values for validation of float or int values. Ignored if value is None.
- **maxvals** *(list)* – optional list of maximum values for validation of float or int values. Ignored if value is None.
- **sizevals** *(list)* – optional list of wx.Size values for each input widget. Ignored if value is None.
- **checkdictlst** *(tuple)* – an optional list of dicts or lists containing bool values (similar to dictlst).
- **checkelemlst** *(tuple)* – an optional list of dicts or lists containing bool key values (similar to elemlst). Must be used with checkdictlst.
- **checklabel** *(string)* – a string to use for each checkbutton

Returns the wx.Dialog created here. Use method .ShowModal() to display it.

Example for use of ScrolledMultiEditor:
dlg = <pkg>.ScrolledMultiEditor(frame, dictlst, elemlst, prelbl, postlbl, 
        header=header)
if dlg.ShowModal() == wx.ID_OK:
    for d,k in zip(dictlst, elemlst):
        print d[k]

Example definitions for dictlst and elemlst:

dictlst = (dict1, list1, dict1, list1)
elemlst = ('a', 1, 2, 3)

This causes items dict1['a'], list1[1], dict1[2] and list1[3] to be edited.

Note that these items must have int, float or str values assigned to them. The dialog will force these types to be 
retained. String values that are blank are marked as invalid.

ControlOKButton(setvalue)

Enable or Disable the OK button for the dialog. Note that this is passed into the ValidatedTxtCtrl for use
by validators.

Parameters

setvalue(bool) – if True, all entries in the dialog are checked for validity. if
False then the OK button is disabled.

class GSASIIctrlGUI.SelectConfigSetting(parent=None)

Dialog to select configuration variables and set associated values.

OnApplyChanges(event=None)

Set config variables to match the current settings

OnBoolSelect(event)

Respond to a change in a True/False variable

OnChange(event=None)

Check if anything been changed. Turn the save button on/off.

OnSave(event)

Write the config variables to config.py and then set them as the current settings

OnSelection()

show a selected variable

onSelDir(event)

Select a directory from a menu

GSASIIctrlGUI.SelectEdit1Var(G2frame, array, labelLst, elemKeysLst, dspLst, refFlgElem)

Select a variable from a list, then edit it and select histograms to copy it to.

Parameters

• G2frame(wx.Frame) – main GSAS-II frame
• array(dict) – the array (dict or list) where values to be edited are kept
• labelLst(list) – labels for each data item
• elemKeysLst(list) – a list of lists of keys needed to be applied (see below) to obtain
  the value of each parameter
• dspLst(list) – list list of digits to be displayed (10,4) is 10 digits with 4 decimal places.
  Can be None.
• refFlgElem(list) – a list of lists of keys needed to be applied (see below) to obtain the
  refine flag for each parameter or None if the parameter does not have refine flag.
Example::

    array = data labelLst = ['v1','v2']
    elemKeysLst = [['v1'], ['v2',0]]
    refFlgElem = [None, ['v2',1]]

- The value for v1 will be in data['v1'] and this cannot be refined while,
- The value for v2 will be in data['v2'][0] and its refinement flag is data['v2'][1]

**GSASIIctrlGUI.ShowHelp**(helpType, frame)

Called to bring up a web page for documentation.

**class GSASIIctrlGUI.ShowLSParms**(parent, title, parmDict, varyList, fullVaryList, size=(375, 430))

Create frame to show least-squares parameters

**DrawPanel**()

Draws the contents of the entire dialog. Called initially & when radio buttons are pressed

**repaintScrollTbl**()

Shows the selected variables

This routine is pretty slow when lots of variables are selected. Why?

**GSASIIctrlGUI.ShowWebPage**(URL, frame)

Called to show a tutorial web page.

**class GSASIIctrlGUI.SingleFloatDialog**(parent, title, prompt, value, limits=[0.0, 1.0], format='%.5g')

Dialog to obtain a single float value from user

**Parameters**

- **parent**(wx.Frame) – name of parent frame
- **title**(str) – title string for dialog
- **prompt**(str) – string to tell user what they are inputing
- **value**(str) – default input value, if any
- **limits**(list) – upper and lower value used to set bounds for entry, use [None,None] for no bounds checking, [None,val] for only upper bounds, etc. Default is [0,1]. Values outside of limits will be ignored.
- **format**(str) – string to format numbers. Defaults to ‘%.5g’. Use ‘%d’ to have integer input (but dlg.GetValue will still return a float).

Typical usage:

```python
limits = (0,1)
dlg = G2G.SingleFloatDialog(G2frame,'New value','Enter new value for...',limits)
if dlg.ShowModal() == wx.ID_OK:
    parm = dlg.GetValue()
dlg.Destroy()
```

**class GSASIIctrlGUI.SingleIntDialog**(parent, title, prompt, value, limits=[None, None])

Dialog to obtain a single int value from user

**Parameters**

- **parent**(wx.Frame) – name of parent frame
- **title**(str) – title string for dialog
- **prompt**(str) – string to tell user what they are inputing
- **value** *(str)* – default input value, if any
- **limits** *(list)* – upper and lower value used to set bounds for entries. Default is [None,None] – for no bounds checking; use [None,val] for only upper bounds, etc. Default is [0,1]. Values outside of limits will be ignored.

Typical usage:

```python
limits = (0, None)  # allows zero or positive values only
dlg = G2G.SingleIntDialog(G2frame, 'New value', 'Enter new value for...', default,
                           limits)
if dlg.ShowModal() == wx.ID_OK:
    parm = dlg.GetValue()

dlg.Destroy()
```

```python
class GSASIIctrlGUI.SingleStringDialog(parent, title, prompt, value='', size=(200, -1), help='')
```

Dialog to obtain a single string value from user

**Parameters**

- **parent** *(wx.Frame)* – name of parent frame
- **title** *(str)* – title string for dialog
- **prompt** *(str)* – string to tell user what they are inputting
- **value** *(str)* – default input value, if any

**GetValue** ()

Use this method to get the value entered by the user: returns: string entered by user

**Show** ()

Use this method after creating the dialog to post it: returns: True if the user pressed OK; False if the User pressed Cancel

```python
class GSASIIctrlGUI.StripIndents(msg, singleLine=False)
```

Strip indentation from multiline strings

```python
class GSASIIctrlGUI.StripUnicode(string, subs='.')
```

Strip non-ASCII characters from strings

**Parameters**

- **string** *(str)* – string to strip Unicode characters from
- **subs** *(str)* – character(s) to place into string in place of each Unicode character. Defaults to `.`

**Returns** a new string with only ASCII characters

```python
class GSASIIctrlGUI.Table(data=[], rowLabels=None, colLabels=None, types=None)
```

Basic data table for use with GSgrid

```python
class GSASIIctrlGUI.ValidatedTxtCtrl(parent, loc, key, nDig=None, notBlank=True, min=None, max=None, OKcontrol=None, OnLeave=None, typeHint=None, CIFinput=False, exclLim=[False, False], OnLeaveArgs={}, ASCIIonly=False, **kw)
```

Create a TextCtrl widget that uses a validator to prevent the entry of inappropriate characters and changes color to highlight when invalid input is supplied. As valid values are typed, they are placed into the dict or list where the initial value came from. The type of the initial value must be int, float or str or None (see key and typeHint); this type (or the one in typeHint) is preserved.
Float values can be entered in the TextCtrl as numbers or also as algebraic expressions using operators + - / * () and **, in addition pi, sind(), cosd(), tand(), and sqrt() can be used, as well as abbreviations s, sin, c, cos, t, tan and sq.

Parameters

- **parent** *(wx.Panel)* – name of panel or frame that will be the parent to the TextCtrl. Can be None.
- **loc** *(dict/list)* – the dict or list with the initial value to be placed in the TextCtrl.
- **key** *(int/str)* – the dict key or the list index for the value to be edited by the TextCtrl. The loc[key] element must exist, but may have value None. If None, the type for the element is taken from typeHint and the value for the control is set initially blank (and thus invalid.) This is a way to specify a field without a default value: a user must set a valid value.

If the value is not None, it must have a base type of int, float, str or unicode; the TextCtrl will be initialized from this value.

- **nDig** *(list)* – number of digits, places and optionally the format ([nDig,nPlc,fmt]) after decimal to use for display of float. The format is either ‘f’ (default) or ‘g’. Alternately, None can be specified which causes numbers to be displayed with approximately 5 significant figures for floats. If this is specified, then typeHint = float becomes the default. (Default=None).
- **notBlank** *(bool)* – if True (default) blank values are invalid for str inputs.
- **min** *(number)* – minimum allowed valid value. If None (default) the lower limit is unbounded. NB: test in NumberValidator is val >= min not val > min
- **max** *(number)* – maximum allowed valid value. If None (default) the upper limit is unbounded NB: test in NumberValidator is val <= max not val < max
- **exclLim** *(list)* – if True exclude min/max value ([exclMin,exclMax]); (Default=[False,False])
- **OKcontrol** *(function)* – specifies a function or method that will be called when the input is validated. The called function is supplied with one argument which is False if the TextCtrl contains an invalid value and True if the value is valid. Note that this function should check all values in the dialog when True, since other entries might be invalid. The default for this is None, which indicates no function should be called.
- **OnLeave** *(function)* – specifies a function or method that will be called when the focus for the control is lost. The called function is supplied with (at present) three keyword arguments:
  - invalid: *(bool)* True if the value for the TextCtrl is invalid
  - value: *(int/float/str)* the value contained in the TextCtrl
  - tc: *(wx.TextCtrl)* the TextCtrl object

The number of keyword arguments may be increased in the future should needs arise, so it is best to code these functions with a **kwargs argument so they will continue to run without errors

The default for OnLeave is None, which indicates no function should be called.

- **typeHint** *(type)* – the value of typeHint should be int, float or str (or None). The value for this will override the initial type taken from value for the dict/list element loc[key] if not None and thus specifies the type for input to the TextCtrl. Defaults as None, which is ignored, unless nDig is specified in which case the default is float.
• **CIFinput (bool)** – for str input, indicates that only printable ASCII characters may be entered into the TextCtrl. Forces output to be ASCII rather than Unicode. For float and int input, allows use of a single ‘?’ or ‘.’ character as valid input.

• **OnLeaveArgs (dict)** – a dict with keyword args that are passed to the OnLeave function. Defaults to {}.

• **ASCIIonly (bool)** – if set as True will remove unicode characters from strings

• (other) – other optional keyword parameters for the wx.TextCtrl widget such as size or style may be specified.

**OnKeyDown (event)**
Special callback for wx 2.9+ on Mac where backspace is not processed by validator

**ShowStringValidity (previousInvalid=True)**
Check if input is valid. Anytime the input is invalid, call self.OKcontrol (if defined) because it is fast. If valid, check for any other invalid entries only when changing from invalid to valid, since that is slower.

**Parameters previousInvalid (bool)** – True if the TextCtrl contents were invalid prior to the current change.

**GSASIIctrlGUI.askSaveDirectory (G2frame)**
Ask the user to supply a directory name. Path name is used as the starting point for the next export path search.

**Returns** a directory name (str) or None if Cancel is pressed

**GSASIIctrlGUI.askSaveFile (G2frame, defnam, extension, longFormatName, parent=None)**
Ask the user to supply a file name

**Parameters**

- **G2frame (wx.Frame)** – The main GSAS-II window
- **defnam (str)** – a default file name
- **extension (str)** – the default file extension beginning with a ‘.’
- **longFormatName (str)** – a description of the type of file
- **parent (wx.Frame)** – the parent window for the dialog. Defaults to G2frame.

**Returns** a file name (str) or None if Cancel is pressed

**class GSASIIctrlGUI.downdate (parent=None)**
Dialog to allow a user to select a version of GSAS-II to install

**getVersion ()**
Get the version number in the dialog

**GSASIIctrlGUI.tutorialIndex = ([Getting started], [StartingGSASII], [Starting GSAS.htm], ...)**
A catalog of GSAS-II tutorials with headings. This is the master list of GSAS-II tutorials and must be updated when tutorials are added. Each item has either one or three items. Titles are single item in a list or tuple. Tutorials have four items: (a) the name of the directory, (b) the name of the web page, (c) a title for the tutorial and (d) a short text description (optional). Tutorials that depend on a previous tutorial being completed should have the title for the tutorial indented by five spaces.

Note that tutorialCatalog is generated from this tuple. Also see makeTutorial which is used to read this and create a web page.

### 1.5.2 GSASIIIO: Misc I/O routines

Module with miscellaneous routines for input and output. Many are GUI routines to interact with user.
Includes support for image reading.
Also includes base classes for data import routines.

This module needs some work to separate wx from non-wx routines

```python
GSASIIIO.BlockSelector(ChoiceList, ParentFrame=None, title='Select a block', size=None, header='Block Selector', useCancel=True)
```

Provide a wx dialog to select a block if the file contains more than one set of data and one must be selected

```python
class GSASIIIO.ExportBaseclass(G2frame, formatName, extension, longFormatName=None)
```

Defines a base class for the exporting of GSAS-II results.

This class is subclassed in the various exports/G2export_*.py files. Those files are imported in GSASIIdataGUI.GSASII._init_Exports() which defines the appropriate menu items for each one and the .Exporter method is called directly from the menu item.

Routines may also define a .Writer method, which is used to write a single file without invoking any GUI objects.

```python
CloseFile(fp=None)
```

Close a file opened in OpenFile

**Parameters**

- **fp** *(file)* – the file object to be closed. If None (default) file object self.fp is closed.

```python
ExportSelect(AskFile='ask')
```

Selects histograms or phases when needed. Sets a default file name when requested into self.filename; always sets a default directory in self.dirname.

**Parameters**

- **AskFile** *(bool)* – Determines how this routine processes getting a location to store the current export(s).
  - if AskFile is ‘ask’ (default option), get the name of the file to be written; self.filename and self.dirname are always set. In the case where multiple files must be generated, the export routine should do this based on self.filename as a template.
  - if AskFile is ‘dir’, get the name of the directory to be used; self.filename is not used, but self.dirname is always set. The export routine will always generate the file name.
  - if AskFile is ‘single’, get only the name of the directory to be used when multiple items will be written (as multiple files) are used or a complete file name is requested when a single file name is selected. self.dirname is always set and self.filename used only when a single file is selected.
  - if AskFile is ‘default’, creates a name of the file to be used from the name of the project (.gpx) file. If the project has not been saved, then the name of file is requested. self.filename and self.dirname are always set. In the case where multiple file names must be generated, the export routine should do this based on self.filename.
  - if AskFile is ‘default-dir’, sets self.dirname from the project (.gpx) file. If the project has not been saved, then a directory is requested. self.filename is not used.

**Returns**

True in case of an error

```python
GetAtoms(phasenam)
```

Gets the atoms associated with a phase. Can be used with standard or macromolecular phases

**Parameters**

- **phasenam** *(str)* – the name for the selected phase

**Returns**

a list of items for each atom where each item is a list containing: label, typ, mult, xyz, and td, where
• label and typ are the atom label and the scattering factor type (str)
• mult is the site multiplicity (int)
• xyz is contains a list with four pairs of numbers: x, y, z and fractional occupancy and their standard uncertainty (or a negative value)
• td is contains a list with either one or six pairs of numbers: if one number it is $U_{iso}$ and with six numbers it is $U_{11}, U_{22}, U_{33}, U_{12}, U_{13} & U_{23}$ paired with their standard uncertainty (or a negative value)

**GetCell (phasenam)**

Gets the unit cell parameters and their s.u.’s for a selected phase

Parameters  
phasenam (**str**) – the name for the selected phase

Returns  
cellList, cellSig where each is a 7 element list corresponding to a, b, c, alpha, beta, gamma, volume where cellList has the cell values and cellSig has their uncertainties.

**GetSeqCell (phasenam, data_name)**

Gets the unit cell parameters and their s.u.’s for a selected phase and histogram in a sequential fit

Parameters

• phasenam (**str**) – the name for the selected phase
• data_name (**dict**) – the sequential refinement parameters for the selected histogram

Returns  
cellList, cellSig where each is a 7 element list corresponding to a, b, c, alpha, beta, gamma, volume where cellList has the cell values and cellSig has their uncertainties.

**InitExport (event)**

Determines the type of menu that called the Exporter and misc initialization.

**MakePWDRfilename(hist)**

Make a filename root (no extension) from a PWDR histogram name

Parameters  
hist (**str**) – the histogram name in data tree (starts with “PWDR “)

**OpenFile (fil=None, mode='w')**

Open the output file

Parameters  
fil (**str**) – The name of the file to open. If None (default) the name defaults to self.dirname + self.filename. If an extension is supplied, it is not overridden, but if not, the default extension is used.

Returns  
the file object opened by the routine which is also saved as self.fp

**SetSeqRef (data, hist)**

Set the exporter to retrieve results from a sequential refinement rather than the main tree

**Write (line)**

Write a line of output, attaching a line-end character

Parameters  
line (**str**) – the text to be written.

**askSaveDirectory ()**

Ask the user to supply a directory name. Path name is used as the starting point for the next export path search.

Returns  
a directory name (**str**) or None if Cancel is pressed

TODO: Can this be replaced with G2G.askSaveDirectory?

**askSaveFile ()**

Ask the user to supply a file name
Returns a file name (str) or None if Cancel is pressed

dumpTree (mode='type')
Print out information on the data tree dicts loaded in loadTree. Used for testing only.

loadParmDict ()
Load the GSAS-II refinable parameters from the tree into a dict (self.parmDict). Update refined values to
those from the last cycle and set the uncertainties for the refined parameters in another dict (self.sigDict).
Expands the parm & sig dicts to include values derived from constraints.
This could be made faster for sequential fits by reducing the histogram list to only the active histogram
being exported.

loadTree ()
Load the contents of the data tree into a set of dicts (self.OverallParms, self.Phases and self.Histogram as
well as self.powderDict & self.xtalDict)
• The childrenless data tree items are overall parameters/controls for the entire project and are placed
in self.OverallParms
• Phase items are placed in self.Phases
• Data items are placed in self.Histogram. The key for these data items begin with a keyword, such as
PWDR, IMG, HKLF,... that identifies the data type.

GSASIIIO.ExportPowder (G2frame, TreeName, fileroot, extension)
Writes a single powder histogram using the Export routines. This is used in GSASIIimgGUI. AutoIntFrame() only.
Parameters
• G2frame (wx.Frame) – the GSAS-II main data tree window
• TreeName (str) – the name of the histogram (PWDR ...) in the data tree
• fileroot (str) – name for file to be written, extension ignored
• extension (str) – extension for file to be written (start with '.'). Must match a powder
export routine that has a Writer object.

GSASIIIO.ExportPowderList (G2frame)
Returns a list of extensions supported by GSASIIIO:ExportPowder() This is used in GSASIIimgGUI. AutoIntFrame() only.
Parameters G2frame (wx.Frame) – the GSAS-II main data tree window

GSASIIIO.ExportSequential (G2frame, data, obj, exporttype)
Used to export from every phase/dataset in a sequential refinement using a .Writer method for either projects or
phases. Prompts to select histograms and for phase exports, which phase(s).
Parameters
• G2frame (wx.Frame) – the GSAS-II main data tree window
• data (dict) – the sequential refinement data object
• exporttype (str) – indicates the type of export (‘project’ or ‘phase’)

GSASIIIO.ExtractFileFromZip (filename, selection=None, confirmread=True, confirmoverwrite=True, parent=None, multipleselect=False)
If the filename is a zip file, extract a file from that archive.
Parameters
• **Selection** (*list*) – used to predefine the name of the file to be extracted. Filename case and zip directory name are ignored in selection; the first matching file is used.

• **confirmread** (*bool*) – if True asks the user to confirm before expanding the only file in a zip

• **confirmoverwrite** (*bool*) – if True asks the user to confirm before overwriting if the extracted file already exists

• **multipleselect** (*bool*) – if True allows more than one zip file to be extracted, a list of file(s) is returned. If only one file is present, do not ask which one, otherwise offer a list of choices (unless selection is used).

**Returns** the name of the file that has been created or a list of files (see multipleselect)

If the file is not a zipfile, return the name of the input file. If the zipfile is empty or no file has been selected, return None

GSASIIIO.FileDlgFixExt(*dlg, file*)

this is needed to fix a problem in linux wx.FileDialog

GSASIIIO.GetCheckImageFile(*G2frame, treeId*)

Try to locate an image file if the project and image have been moved together. If the image file cannot be found, request the location from the user.

**Parameters**

• **G2frame** (*wx.Frame*) – main GSAS-II Frame and data object

• **treeId** (*wx.Id*) – Id for the main tree item for the image

**Returns** Npix,imagefile,imagetag with (Npix) number of pixels, imagefile, if it exists, or the name of a file that does exist or False if the user presses Cancel and (imagetag) an optional image number

GSASIIIO.GetImageData(*G2frame, imagefile, imageOnly=False, ImageTag=None, FormatName="")

Read a single image with an image importer. This is called to reread an image after it has already been imported with GSASIIdataGUI.GSASII.OnImportGeneric() (or ReadImages() in Auto Integration) so it is not necessary to reload metadata.

**Parameters**

• **G2frame** (*wx.Frame*) – main GSAS-II Frame and data object.

• **imagefile** (*str*) – name of image file

• **imageOnly** (*bool*) – If True return only the image, otherwise (default) return more (see below)

• **ImageTag** (*int/str*) – specifies a particular image to be read from a file. First image is read if None (default).

• **formatName** (*str*) – the image reader formatName

**Returns** an image as a numpy array or a list of four items: Comments, Data, Npix and the Image, as selected by imageOnly

GSASIIIO.GetPowderPeaks(*fileName*)

Read powder peaks from a file

GSASIIIO.IndexPeakListSave(*G2frame, peaks*)

Save powder peaks from the indexing list
**GSASIIIO.LoadImage2Tree** *(imagefile, G2frame, Comments, Data, Npix, Image)*

Load an image into the tree. Saves the location of the image, as well as the ImageTag (where there is more than one image in the file), if defined.

**GSASIIIO.MultipleBlockSelector** *(ChoiceList, ParentFrame=None, title='Select a block', size=None, header='Block Selector')*

Provide a wx dialog to select a block of data if the file contains more than one set of data and one must be selected.

**Returns** a list of the selected blocks

**class GSASIIIO.MultipleChoicesDialog** *(choicelist, headinglist, head='Select options', title='Please select from options below', parent=None)*

A dialog that offers a series of choices, each with a title and a wx.Choice widget. Intended to be used Modally. Typical input:

- choicelist=[('a','b',c'), ('test1','test2'), ('no choice',)]
- headinglist=['select a, b or c', 'select 1 of 2', 'No option here']

Selections are placed in self.chosen when OK is pressed.

Also see GSASIIctrlGUI

**GSASIIIO.MultipleChoicesSelector** *(choicelist, headinglist, ParentFrame=None, **kwargs)*

A modal dialog that offers a series of choices, each with a title and a wx.Choice widget. Typical input:

- choicelist=[('a','b','c'), ('test1','test2'), ('no choice',)]
- headinglist=['select a, b or c', 'select 1 of 2', 'No option here']

Optional keyword parameters are: head (window title) and title returns a list of selected indicies for each choice (or None)

**GSASIIIO.PeakListSave** *(G2frame, file, peaks)*

Save powder peaks to a data file

**GSASIIIO.PhaseSelector** *(ChoiceList, ParentFrame=None, title='Select a phase', size=None, header='Phase Selector')*

Provide a wx dialog to select a phase if the file contains more than one phase

**GSASIIIO.ProjFileOpen** *(G2frame, showProvenance=True)*

Read a GSAS-II project file and load into the G2 data tree

**GSASIIIO.ProjFileSave** *(G2frame)*

Save a GSAS-II project file

**GSASIIIO.PutG2Image** *(filename, Comments, Data, Npix, image)*

Write an image as a python pickle - might be better as an .edf file?

**GSASIIIO.ReadImages** *(G2frame, imagefile)*

Read one or more images from a file and put them into the Tree using image importers. Called only in AutoIntFrame.OnTimerLoop().

**ToDo:** Images are most commonly read in `GSASIIdataGUI.GSASII.OnImportGeneric()` which is called from `GSASIIdataGUI.GSASII.OnImportImage()` it would be good if these routines used a common code core so that changes need to be made in only one place.

**Parameters**

- **G2frame (wx.Frame)** – main GSAS-II Frame and data object.
- **imagefile (str)** – name of image file

**Returns** a list of the id’s of the IMG tree items created

---

1.5. **GSAS-II GUI Support Routines**
GSASIIIO.\texttt{SaveIntegration}(G2frame, PickId, data, Overwrite=False)
Save image integration results as powder pattern(s)

GSASIIIO.\texttt{XYsave}(G2frame, XY, labelX=‘X’, labelY=‘Y’, names=[])
Save XY table data

GSASIIIO.\texttt{objectScan}(data, tag, indexStack=[])
Recursively scan an object looking for unexpected data types. This is used in debug mode to scan .gpx files for objects we did not intend to be there.

GSASIIIO.\texttt{sfloat}(S)
Convert a string to float. An empty field or a unconvertable value is treated as zero

GSASIIIO.\texttt{sint}(S)
Convert a string to int. An empty field is treated as zero

GSASIIIO.\texttt{striphist}(var, insChar=“”)
strip a histogram number from a var name

GSASIIIO.\texttt{trim}(val)
Simplify a string containing leading and trailing spaces as well as newlines, tabs, repeated spaces etc. into a shorter and more simple string, by replacing all ranges of whitespace characters with a single space.

\begin{description}
\item[Parameters] val (str) – the string to be simplified
\item[Returns] the (usually) shortened version of the string
\end{description}

\texttt{class ReadMarCCDFrame.marFrame}(File, byteOrd='<', IFD={})
A class to extract correct mar header and image info from a MarCCD file

\begin{description}
\item[Parameters]
\begin{itemize}
\item File (str) – file object [from open()]
\item byteOrd – ‘<’ (default) or ‘>’
\item IFD (dict) – ?
\end{itemize}
\end{description}

1.5.3 \textit{GSASIIpy3: Python 3.x Routines}

Module to hold python 3-compatible code, to keep it separate from code that will break with \_\_future\_\_ options.

GSASIIpy3.\texttt{FormatPadValue}(val, maxdigits=None)
Format a float to fit in maxdigits[0] spaces with maxdigits[1] after decimal.

\begin{description}
\item[Parameters]
\begin{itemize}
\item val (float) – number to be formatted.
\item maxdigits (list) – the number of digits & places after decimal to be used for display of the number (defaults to [10,2]).
\end{itemize}
\item[Returns] a string with exactly maxdigits[0] characters (except under error conditions), but last character will always be a space
\end{description}

GSASIIpy3.\texttt{FormatSigFigs}(val, maxdigits=10, sigfigs=5, treatAsZero=1e-20)
Format a float to use maxdigits or fewer digits with sigfigs significant digits showing (if room allows).

\begin{description}
\item[Parameters]
\begin{itemize}
\item val (float) – number to be formatted.
\item maxdigits (int) – the number of digits to be used for display of the number (defaults to 10).
\end{itemize}
\end{description}
• **sigfigs** (int) – the number of significant figures to use, if room allows

• **treatAsZero** (float) – numbers that are less than this in magnitude are treated as zero.
  
  Defaults to 1.0e-20, but this can be disabled if set to None.

**Returns** a string with <= maxdigits characters (I hope).

**GSASIIpy3.FormatValue** (*val*, *maxdigits=None*)

Format a float to fit in at most *maxdigits*[0] spaces with *maxdigits*[1] after decimal. Note that this code has been hacked from FormatSigFigs and may have unused sections.

**Parameters**

• **val** (float) – number to be formatted.

• **maxdigits** (list) – the number of digits, places after decimal and ‘f’ or ‘g’ to be used for display of the number (defaults to [10,2,’f’]).

**Returns** a string with <= maxdigits characters (usually).

**GSASIIpy3.FormulaEval** (*string*)

Evaluates a algebraic formula into a float, if possible. Works properly on fractions e.g. 2/3 only with python 3.0+ division.

Expressions such as 2/3, 3*pi, sin(45)/2, 2*sqrt(2), 2**10 can all be evaluated.

**Parameters**

• **string** (str) – Character string containing a Python expression to be evaluated.

**Returns** the value for the expression as a float or None if the expression does not evaluate to a valid number.

# 1.6 GSAS-II GUI Submodules

## 1.6.1 GSASIIdataGUI: Main GSAS-II GUI

Module that defines GUI routines and classes for the main GUI Frame (window) and the main routines that define the GSAS-II tree panel and much of the data editing panel.

**class** **GSASIIdataGUI.G2DataWindow** (*parent*)

Create the data item window as well as the menu. Note that the same core menu items are used in all menus, but different items may be added depending on what data tree item (and for phases, the phase tab).

Note that while the menus are created here, the binding for the menus is done later in various GSASII*GUI modules, where the functions to be called are defined.

Use of the dataWindow scrolled panel:

dataWindow has a “master” vertical BoxSizer: find it with G2frame.dataWindow.GetSizer() and always use it. A call to dataWindow.SetSizer() should not be needed.

When placing a widget in the sizer that has its own scrolling (e.g. G2G.GSNoteBook, anything else?) that one widget should be placed in the sizer as

```
G2frame.dataWindow.GetSizer().Add(G2frame.<obj>,1.wx.ALL|wx.EXPAND)
```

[is wx.ALL superfluous here?] so that it consumes the full size of the panel and so that the NoteBook widget does the scrolling.

For other uses, one will likely place a bunch of widgets and (other [sub-]sizers) into the master sizer. In this case, DO NOT use wx.EXPAND, as this will result in the widget resizing/repositioning as the window
resizes. Possible exceptions might be for widgets going into a fixed-size panel that is inside the dataWindow (probably not being done). A call to Sizer.Fit(dataWindow) will do bad things, though a call to SubSizer.Fit(dataWindow.subpanel) could make sense.

Initial GUI draws to dataWindow will go through GSASIIdataGUI.SelectDataTreeItem(), which is called after any changes to data tree selection. SelectDataTreeItem places items in dataWindow or calls that do that. Before it calls those routines, it calls

G2frame.dataWindow.ClearData()

which deletes the contents of the master sizer. After the contents are posted a call is made to

G2frame.dataWindow.SetDataSize()

which repaints the window. For routines [such as GSASIIpwdGUI.UpdatePeakGrid()] that are called repeatedly to update the entire contents of dataWindow themselves, it is important to add calls to

G2frame.dataWindow.ClearData()

and

G2frame.dataWindow.SetDataSize()

at the beginning and end respectively to clear and refresh. This is not needed for GSNoteBook repaints, which seem to be working mostly automatically. If there is a problem, a call like

wx.CallAfter(G2frame.phaseDisplay.SendSizeEvent)

might be needed. There are some calls to G2frame.dataWindow.SendSizeEvent() that may be doing the same thing.

ClearData()

Initializes the contents of the dataWindow panel

OnResize(event)

Used for grids to match ScrolledWindow size

PostfillDataMenu(empty=False)

Add the help menu to the menus associated with data tree items.

PrefillDataMenu(menu, empty=False)

Create the “standard” part of data frame menus & add the dataWindow menu headings. This menu duplicates the tree menu, but adds an extra help command for the current data item and a separator.

SetDataSize()

Sizes the contents of the dataWindow panel

class GSASIIdataGUI.GSASII(parent)

Define the main GSAS-II frame and its associated menu items.

    Parameters parent – reference to parent application

AddSimulatedPowder(ttArr, intArr, HistName, Lam1, Lam2)

Create a PWDR entry for a computed powder pattern

CheckNotebook()

Make sure the data tree has the minimally expected controls.

class CopyDialog(parent, title, text, data)

Creates a dialog for copying control settings between data tree items

EditProxyInfo(event)

Edit the proxy information used by subversion
**ErrorDialog** *(title, message, parent=None, wtype=<sphinx.ext.autodoc.importer._MockObject object>)*

Display an error message

**ExitMain**(event)

Called if exit selected or the main window is closed record last position of data & plot windows; saved to config.py file NB: not called if console window closed

**ExpandAll**(event)

Expand all tree items or those of a single type

**FillMainMenu**(menubar, addhelp=True)

Define contents of the main GSAS-II menu for the (main) data tree window. For the mac, this is also called for the data item windows as well so that the main menu items are data menu as well.

**GetFileList**(fileType, skip=None)

Appears unused. Note routine of same name in GSASIIpwdGUI

**GetHKLFdatafromTree**(HKLFname)

Returns single crystal data from GSASII tree

Parameters HKLFname (str) – a single crystal histogram name as obtained from GSASIIfstruct.GetHistogramNames()

Returns HKLFdata = single crystal data list of reflections

**GetHistogramNames**(hType)

Returns a list of histogram names found in the GSASII data tree Note routine GSASIIfstrIO. GetHistogramNames() also exists to get same info from GPX file.

Parameters hType (str) – list of histogram types

Returns list of histogram names

**GetHistogramNamesID**(hType)

Returns a list of histogram names found in the GSASII data tree and a lookup table of their Id values. Should replace GetHistogramNames since that will not be much faster (and there may be real speed gains from caching the Ids rather than keep searching for them).

N.B routine GSASIIfstrIO.GetHistogramNames() also exists to get same info, but from GPX file.

Parameters hType (str) – list of histogram types

Returns list of histogram names and a dict of histogram Ids keyed by histogram name.

**GetPWDRdatafromTree**(PWDRname)

Returns powder data from GSASII tree

Parameters PWDRname (str) – a powder histogram name as obtained from GSASIIfstruct.GetHistogramNames()

Returns PWDRdata = powder data dictionary with Powder data arrays, Limits, Instrument Parameters, Sample Parameters

**GetPhaseData**()

Returns a dict with defined phases. Note routine GSASIIfstrIO.GetPhaseData() also exists to get same info from GPX file.

**GetPhaseInfofromTree**()

Get the phase names and their rId values, also the histograms used in each phase.

Returns

(phaseRIdList, usedHistograms) where

1.6. **GSAS-II GUI Submodules**
• phaseRIdList is a list of random Id values for each phase
• usedHistograms is a dict where the keys are the phase names and the values for each key are a list of the histogram names used in each phase.

GetPhaseNames()
Returns a list of defined phases. Note routine GSASIIstrIO.GetPhaseNames() also exists to get same info from GPX file.

GetPowderIparm(rd, prevIparm, lastIparmfile, lastdatafile)
Open and read an instrument parameter file for a data file Returns the list of parameters used in the data tree

Parameters
• rd(obj) – the raw data (histogram) data object.
• prevIparm(str) – not used
• lastIparmfile(str) – Name of last instrument parameter file that was read, or a empty string.
• lastdatafile(str) – Name of last data file that was read.

Returns a list of two dicts, the first containing instrument parameters and the second used for TOF lookup tables for profile coeff.

GetUsedHistogramsAndPhasesfromTree()
Returns all histograms that are found in any phase and any phase that uses a histogram. This also assigns numbers to used phases and histograms by the order they appear in the file. Note routine GSASIIstrIO.GetUsedHistogramsAndPhases() also exists to get same info from GPX file.

Returns
(Histograms,Phases)
• Histograms = dictionary of histograms as {name:data,...}
• Phases = dictionary of phases that use histograms

MakeLSParmDict(seqHist=None)
Load all parameters used for computation from the tree into a dict of paired values [value, refine flag]. Note that this is different than the parmDict used in the refinement, which only has values.

Note that similar things are done in GSASIIIO.ExportBaseclass.loadParmDict() (from the tree) and GSASIIstrMain.Refine() and GSASIIstrMain.SeqRefine() (from a GPX file).

Parameters seqHist (dict) – defines a specific histogram to be loaded for a sequential refinement, if None (default) all are loaded. Note: at present this parameter is not used anywhere.

Returns
(parDict,varyList) where:
• parDict is a dict with values and refinement flags for each parameter and
• varyList is a list of variables (refined parameters).

MenuBinding(event)
Called when a menu is clicked upon; looks up the binding in table

MoveTreeItems(event)
Move tree items of a single type to the end of the tree
OnAddPhase (event)
Add a new, empty phase to the tree. Called by Data/Add Phase menu

OnColMetaTest (event)
Test the .par/*lbls pair for contents

OnDataDelete (event)
Delete one or more histograms from data tree. Called by the Data/DeleteData menu

OnDataTreeSelChanged (event)
Called when a data tree item is selected. May be called on item deletion as well.

OnDeletePhase (event)
Delete a phase from the tree. Called by Data/Delete Phase menu

OnDummyPowder (event)
Called in response to Import/Powder Data/Simulate menu item to create a Dummy powder diffraction data set.

Reads an instrument parameter file and then gets input from the user

OnExportPDF (event)
Save S(Q), G(R),... as selected by user

OnFileClose (event)
Clears the data tree in response to the File/New Project menu button. User is given option to save the project.

OnFileOpen (event, filename=None)
Gets a GSAS-II .gpx project file in response to the File/Open Project menu button

OnFileSave (event)
Save the current project in response to the File/Save Project menu button

OnFileSaveas (event)
Save the current project in response to the File/Save as menu button

OnGPXtreeItemActivated (event)
Called when a tree item is activated

OnGPXtreeItemCollapsed (event)
Called when a tree item is collapsed - all children will be collapsed

OnGPXtreeItemDelete (event)
Called when a tree item is deleted, inhibit the next tree item selection action

OnGPXtreeItemExpanded (event)
Called when a tree item is expanded

OnGPXtreeKeyDown (event)
Allows stepping through the tree with the up/down arrow keys

OnImageSum (event)
Sum together image data

OnImportGeneric (reader, readerlist, label, multiple=False, usedRanIdList=[], Preview=True, load2Tree=False)
Used for all imports, including Phases, datasets, images...

Called from GSASII.OnImportPhase(), GSASII.OnImportImage(), GSASII.OnImportSfact(), GSASII.OnImportPowder(), GSASII.OnImportSmallAngle() and .meth:`GSASII.OnImportReflectometry`
Uses reader_objects subclassed from GSASIIobj.ImportPhase, GSASIIobj.ImportStructFactor, GSASIIobj.ImportPowderData, GSASIIobj.ImportSmallAngleData GSASIIobj.ImportReflectometryData or GSASIIobj.ImportImage. If a specific reader is specified, only that method will be called, but if no reader is specified, every one that is potentially compatible (by file extension) will be tried on the file(s) selected in the Open File dialog.

Parameters

- **reader** *(reader_object)* – This will be a reference to a particular object to be used to read a file or None, if every appropriate reader should be used.

- **readerlist** *(list)* – a list of reader objects appropriate for the current read attempt. At present, this will be either self.ImportPhaseReaderlist, self.ImportSfactReaderlist self.ImportPowderReaderlist or self.ImportImageReaderlist (defined in _init_Imports from the files found in the path), but in theory this list could be tailored. Used only when reader is None.

- **label** *(str)* – string to place on the open file dialog: Open *label* input file

- **multiple** *(bool)* – True if multiple files can be selected in the file dialog. False is default. At present True is used only for reading of powder data.

- **usedRanIdList** *(list)* – an optional list of random Ids that have been used and should not be reused

- **Preview** *(bool)* – indicates if a preview of the file should be shown. Default is True, but set to False for image files which are all binary.

- **load2Tree** *(bool)* – indicates if the file should be loaded into the data tree immediately (used for images only). True only when called from OnImportImage(); causes return value to change to a list of True values rather than reader objects.

Returns a list of reader objects (rd_list) that were able to read the specified file(s). This list may be empty.

**OnImportImage**(event)
Called in response to an Import/Image/... menu item to read an image from a file. Like all the other imports, dict self.ImportMenuId is used to look up the specific reader item associated with the menu item, which will be None for the last menu item, which is the “guess” option where all appropriate formats will be tried.

A reader object is filled each time an image is read.

**OnImportPDF**(event)
Called in response to an Import/PDF G(R) Data/... menu item to read a PDF G(R) data set. dict self.ImportMenuId is used to look up the specific reader item associated with the menu item, which will be None for the last menu item, which is the “guess” option where all appropriate formats will be tried.

**OnImportPhase**(event)
Called in response to an Import/Phase/... menu item to read phase information. dict self.ImportMenuId is used to look up the specific reader item associated with the menu item, which will be None for the last menu item, which is the “guess” option where all appropriate formats will be tried.

**OnImportPowder**(event)
Called in response to an Import/Powder Data/... menu item to read a powder diffraction data set. dict self.ImportMenuId is used to look up the specific reader item associated with the menu item, which will be None for the last menu item, which is the “guess” option where all appropriate formats will be tried.

Also reads an instrument parameter file for each dataset.
OnImportReflectometry(event)
Called in response to an Import/Reflectometry Data/... menu item to read a reflectometry data set. dict self.ImportMenuId is used to look up the specific reader item associated with the menu item, which will be None for the last menu item, which is the “guess” option where all appropriate formats will be tried.

OnImportSfact(event)
Called in response to an Import/Structure Factor/... menu item to read single crystal datasets. dict self.ImportMenuId is used to look up the specific reader item associated with the menu item, which will be None for the last menu item, which is the “guess” option where all appropriate formats will be tried.

OnImportSmallAngle(event)
Called in response to an Import/Small Angle Data/... menu item to read a small angle diffraction data set. dict self.ImportMenuId is used to look up the specific reader item associated with the menu item, which will be None for the last menu item, which is the “guess” option where all appropriate formats will be tried.

OnMacroRecordStatus(event, setvalue=None)
Called when the record macro menu item is used which toggles the value. Alternately a value to be set can be provided. Note that this routine is made more complex because on the Mac there are lots of menu items (listed in self.MacroStatusList) and this loops over all of them.

OnMakePDFs(event)
Sets up PDF data structure filled with defaults; if found chemical formula is inserted so a default PDF can be made.

OnNewGSASII(event)
Gets a GSAS-II .gpx project file in response to the File/Open new window menu button. Runs only on Mac.

OnPowderFPA(event)
Perform FPA simulation/peak fitting

OnPreferences(event)
Edit the GSAS-II configuration variables

OnPwdrSum(event)
Sum or Average together powder data(?)

OnReadPowderPeaks(event)
Bound to menu Data/Read Powder Peaks

OnRefine(event)
Perform a refinement or a sequential refinement (depending on controls setting) Called from the Calculate/Refine menu.

OnRenameData(event)
Renames an existing phase. Called by Data/Rename Phase menu

OnSaveMultipleImg(event)
Select and save multiple image parameter and mask files

OnSeqRefine(event)
Perform a sequential refinement. Called from self.OnRefine (Which is called from the Calculate/Refine menu)

OnShowLSParms(event)
Displays a window showing all parameters in the refinement. Called from the Calculate/View LS Parms menu.

This could potentially be sped up by loading only the histogram that is needed for a sequential fit.
OpenPowderInstprm \( \text{(instfile)} \)
Read a GSAS-II (new) instrument parameter file

**Parameters**

- **instfile** \( \text{(str)} \) – name of instrument parameter file

PreviewFile \( \text{(filename)} \)
utility to confirm we have the right file

ReadPowderInstprm \( \text{instLines, bank, databanks, rd} \)
Read lines from a GSAS-II (new) instrument parameter file similar to \text{G2pwdGUI.OnLoad} If instprm file has multiple banks each with header #Bank n: . . . , this finds matching bank no. to load - problem with nonmatches?

Note that this routine performs a similar role to \text{GSASIIfiles.ReadPowderInstprm()}, but this will call a GUI routine for selection when needed. TODO: refactor to combine

**Parameters**

- **instLines** \( \text{(list)} \) – strings from GSAS-II parameter file; can be concatenated with ‘;’
- **bank** \( \text{(int)} \) – bank number to check when instprm file has ‘#BANK n: . . .’ strings when bank = n then use parameters; otherwise skip that set. Ignored if BANK n: not present. NB: this kind of instprm file made by a Save all profile command in Instrument Parameters

Return dict Inst instrument parameter dict if OK, or str: Error message if failed

ReadPowderIp parm \( \text{instfile, bank, databanks, rd} \)
Read a GSAS (old) instrument parameter file

**Parameters**

- **instfile** \( \text{(str)} \) – name of instrument parameter file
- **bank** \( \text{(int)} \) – the bank number read in the raw data file
- **databanks** \( \text{(int)} \) – the number of banks in the raw data file. If the number of banks in the data and instrument parameter files agree, then the sets of banks are assumed to match up and bank is used to select the instrument parameter file. If not and not TOF, the user is asked to make a selection.
- **rd** \( \text{(obj)} \) – the raw data (histogram) data object. This sets rd.instbank.

ResetPlots() This reloads the current tree item, often drawing a plot. It also refreshes any plots that have registered a refresh routine (see \text{G2plotNB.RegisterRedrawRoutine}) and deletes all plots that have not been refreshed and require one (see \text{G2plotNB.SetNoDelete}).

SaveTreeSetting() Save the current selected tree item by name (since the id will change)

SetDataSize() this routine is a placeholder until all \text{G2frame.SetDataSize} calls are replaced by \text{G2frame.dataWindow.SetDataSize}

SetTitleByGPX() Set the title for the two window frames

StartProject() Opens a GSAS-II project file & selects the 1st available data set to display (PWDR, HKLF, REFD or SASD)

class SumDialog \( \text{parent, title, text, dataType, data, dataList} \)
Allows user to supply scale factor(s) when summing data
OnFilter(event)
Read text from filter control and select entries that match.

onChar(event)
Respond to keyboard events in the Filter box

reloadFromGPX()
Deletes current data tree & reloads it from GPX file (after a refinement.) Done after events are completed to avoid crashes.

testSeqRefineMode()
Returns the list of histograms included in a sequential refinement or an empty list if a standard (non-sequential) refinement. Also sets Menu item status depending on mode

GSASIIdataGUI.GSASIImain(application)
Start up the GSAS-II GUI

GSASIIdataGUI.GUIpatches()
Misc fixes that only needs to be done when running a GUI

GSASIIdataGUI.GetDisplay(pos)
Gets display number (0=main display) for window position (pos). If pos outside all displays returns None

GSASIIdataGUI.GetGPXtreeDataNames(G2frame, dataTypes)
Finds all items in tree that match a 4 character prefix

Parameters
• **G2frame (wx.Frame)** – Data tree frame object
• **dataTypes (list)** – Contains one or more data tree item types to be matched such as ['IMG'] or ['PWDR', 'HKLF']

Returns a list of tree item names for the matching items

GSASIIdataGUI.GetGPXtreeItemId(G2frame, parentId, itemText)
Find the tree item that matches the text in itemText starting with parentId

Parameters
• **G2frame (wx.Frame)** – Data tree frame object
• **parentId (wx.TreeItemId)** – tree item to start search with
• **itemText (str)** – text for tree item

class GSASIIdataGUI.MergeDialog(parent, data)
HKL transformation & merge dialog

Parameters
• **parent (wx.Frame)** – reference to parent frame (or None)
• **data** – HKLF data

GSASIIdataGUI.SelectDataTreeItem(G2frame, item, oldFocus=None)
Called from GSASIIdataGUI.GSASI.OnDataTreeSelChanged() when a item is selected on the tree. Also called from GSASII.OnGPXtreeEndDrag, OnAddPhase – might be better to select item, triggering the the bind to SelectDataItem

Also Called in GSASIIphsGUI.UpdatePhaseData by OnTransform callback.

GSASIIdataGUI.SetDataMenuBar(G2frame, menu=None)
Set the menu for the data frame.

1.6. GSAS-II GUI Submodules

97
Note that data frame items do not have menus, for these (menu=None) display the standard main menu for the data tree window.

GSASIIdataGUI.ShowVersions()
Show the versions all of required Python packages, etc.

GSASIIdataGUI.UpdateComments(G2frame, data)
Place comments into the data window

GSASIIdataGUI.UpdateControls(G2frame, data)
Edit overall GSAS-II controls in main Controls data tree entry

GSASIIdataGUI.UpdateNotebook(G2frame, data)
Called when the data tree notebook entry is selected. Allows for editing of the text in that tree entry

GSASIIdataGUI.UpdatePWHKPlot(G2frame, kind, item)
Called when the histogram main tree entry is called. Displays the histogram weight factor, refinement statistics for the histogram and the range of data for a simulation.
Also invokes a plot of the histogram.

GSASIIdataGUI.UpdateSeqResults(G2frame, data, prevSize=None)
Called when the Sequential Results data tree entry is selected to show results from a sequential refinement.

Parameters

- **G2frame** (wx.Frame) – main GSAS-II data tree windows
- **data** (dict) – a dictionary containing the following items:
  - ‘histNames’ - list of histogram names in order as processed by Sequential Refinement
  - ‘varyList’ - list of variables - identical over all refinements in sequence note that this is the original list of variables, prior to processing constraints.
  - ‘variableLabels’ – a dict of labels to be applied to each parameter (this is created as an empty dict if not present in data).
  - keyed by histName - dictionaries for all data sets processed, which contains:
    - ‘variables’ - result[0] from leastsq call
    - ‘varyList’ - list of variables passed to leastsq call (not same as above)
    - ‘sig’ - esds for variables
    - ‘covMatrix’ - covariance matrix from individual refinement
    - ‘title’ - histogram name; same as dict item name
    - ‘newAtomDict’ - new atom parameters after shifts applied
    - ‘newCellDict’ - refined cell parameters after shifts to A0-A5 from Dij terms applied’

1.6.2 GSASIIphsGUI: Phase GUI

Module to create the GUI for display of phase information in the data display window when a phase is selected. Phase information is stored in one or more Phase Tree Item objects. Note that there are functions that respond to some tabs in the phase GUI in other modules (such as GSASIIdata).

Main routine here is UpdatePhaseData(), which displays the phase information (called from GSASIIdataGUI:SelectDataTreeItem()).
Other top-level routines are: `GetSpGrpfromUser()` (called locally only); `FindBondsDraw()` and `FindBondsDrawCell()` (called locally and in GSASIIplot); `SetPhaseWindow()` (called locally and in GSASIIddataGUI and GSASIIrestrGUI, multiple locations) to control scrolling.

**class** GSASIIPhsGUI.AddHatomDialog *(parent, Neigh, phase)*

H atom addition dialog. After `ShowModal()` returns, the results are found in dict `self.data`, which is accessed using `GetData()`.

**Parameters**

- `parent (wx.Frame)` – reference to parent frame (or None)
- `Neigh (dict)` – a dict of atom names with list of atom name, dist pairs for neighboring atoms
- `phase (dict)` – a dict containing the phase as defined by *Phase Tree Item*

**Draw (Neigh, phase)**

Creates the contents of the dialog. Normally called by `__init__()`.

**GetData ()**

Returns the values from the dialog

**OnOk (event)**

Called when the OK button is pressed

**class** GSASIIPhsGUI.DIFFaXcontrols *(parent, ctrlrs, parms=None)*

Solicit items needed to prepare DIFFaX control.dif file

**GSASIIPhsGUI.FindBondsDraw (data)**

Generally used routine where cell is from data

**GSASIIPhsGUI.FindBondsDrawCell (data, cell)**

uses numpy & masks - very fast even for proteins! allows different cell as input from seq. refinements

**class** GSASIIPhsGUI.RotationDialog *(parent)*

Get Rotate & translate matrix & vector - currently not used needs rethinking - possible use to rotate a group of atoms about some vector/origin + translation

**GSASIIPhsGUI.SetDrawingDefaults (drawingData)**

Add required items into `data['drawing']` array if not present. This does not add all the items in SetupDrawingData, but it seems that this is not a problem. Perhaps the two routines could be combined?

**class** GSASIIPhsGUI.SetUpFullrmcDialog *(parent, Name, Phase)*

Get from user the super cell size & selected histogram to make various files

**GetData ()**

Returns the values from the dialog

**class** GSASIIPhsGUI.SetUpRMCPProfileDialog *(parent, Name, Phase)*

Get from user the super cell size & selected histogram to make various files

**GetData ()**

Returns the values from the dialog

**class** GSASIIPhsGUI.SphereEnclosure *(parent, general, drawing, indx)*

Add atoms within sphere of enclosure to drawing

**Parameters**

- `parent (wx.Frame)` – reference to parent frame (or None)
- `general` – general data (includes drawing data)
- `atoms` – drawing atoms data
• *index* – list of selected atoms (may be empty)

```python
class GSASIIphsGUI.SymOpDialog(parent, SGData, New=True, ForceUnit=False)
Class to select a symmetry operator
```

*class* GSASIIphsGUI.TransformDialog(parent, phase, Trans=<sphinx.ext.autodoc.importer._MockObject
object>, Uvec=<sphinx.ext.autodoc.importer._MockObject
object>, Vvec=<sphinx.ext.autodoc.importer._MockObject
object>, ifMag=False, BNSlatt='')

Phase transformation \( X' = M^*(X-U)+V \)

**Parameters**

- `parent` (*wx.Frame*) – reference to parent frame (or None)
- `phase` – parent phase data

#NB: commonNames & commonTrans defined in GSASIIdataGUI = G2gd

```python
GSASIIphsGUI.UpdatePhaseData(G2frame, Item, data)
Create the data display window contents when a phase is clicked on in the main (data tree) window. Called
only from GSASIIdataGUI.SelectDataTreeItem(), which in turn is called from GSASIIdataGUI.
GSASII.OnDataTreeSelChanged() when a Phase tree item is selected. This creates all tabs on the page
and fills their contents. Routine OnPageChanged is called each time a tab is pressed and updates the contents of
the tab’s page.
```

**Parameters**

- `G2frame` (*wx.frame*) – the main GSAS-II frame object
- `Item` (*wx.TreeItemId*) – the tree item that was selected
- `data` (*dict*) – all the information on the phase in a dictionary

```python
class GSASIIphsGUI.UseMagAtomDialog(parent, Name, Atoms, atCodes, atMxyz, ifMag=True, ifOK=False, ifDelete=False)
Get user selected magnetic atoms after cell transformation
```

### 1.6.3 GSASIIdddataGUI: Phase Diffraction Data GUI

Module to create the GUI for display of diffraction data * phase information that is shown in the data display window
(when a phase is selected.)

```python
GSASIIdddataGUI.UpdateDData(G2frame, DData, data, hist='', Scroll=0)
Display the Diffraction Data associated with a phase (items where there is a value for each histogram and phase)
```

**Parameters**

- `G2frame` (*wx.frame*) – the main GSAS-II frame object
- `DData` (*wx.ScrolledWindow*) – notebook page to be used for the display
- `data` (*dict*) – all the information on the phase in a dictionary
- `hist` (*str*) – histogram name
- `Scroll` (*int*) – previous scroll position

### 1.6.4 GSASIIElemGUI: GUI to select and delete element lists

Module to select elements from a periodic table and to delete an element from a list of selected elements.
class GSASIIElemGUI.DeleteElement(parent, choice)
    Delete element from selected set widget

ElButton(name, pos)
    Needs a doc string

class GSASIIElemGUI.PickElement(parent, oneOnly=False, ifNone=False, ifMag=False, multiple=False)
    Makes periodic table widget for picking element. Modes: oneOnly if True element symbols are provided, otherwise select valence ifNone if True show None button ifMag if True present magnetic scatters only multiple if True multiple elements can be selected

ElButton(name, pos, tip, color)
    Creates an element button widget

class GSASIIElemGUI.PickElements(parent, list)
    Makes periodic table widget for picking elements - caller maintains element list

1.6.5 GSASIIconstrGUI: Constraint GUI routines

Used to define constraints and rigid bodies.

GSASIIconstrGUI.CheckAllScalePhaseFractions(G2frame)
    Check if scale factor and all phase fractions are refined without a constraint for all used histograms, if so, offer the user a chance to create a constraint on the sum of phase fractions

GSASIIconstrGUI.CheckScalePhaseFractions(G2frame, hist, histograms, phases)
    Check if scale factor and all phase fractions are refined without a constraint for histogram hist, if so, offer the user a chance to create a constraint on the sum of phase fractions

class GSASIIconstrGUI.ConstraintDialog(parent, title, text, data, separator='*', varname=", varyflag=False)
    Window to edit Constraint values

GSASIIconstrGUI.TransConstraints(G2frame, oldPhase, newPhase, Trans, Vec, atCodes)
    Add constraints for new magnetic phase created via transformation of old nuclear one NB: A = [G11,G22,G33,2*G12,2*G13,2*G23]

GSASIIconstrGUI.UpdateConstraints(G2frame, data)
    Called when Constraints tree item is selected. Displays the constraints in the data window

GSASIIconstrGUI.UpdateRigidBodies(G2frame, data)
    Called when Rigid bodies tree item is selected. Displays the rigid bodies in the data window

1.6.6 GSASIIimgGUI: Image GUI

Control image display and processing

class GSASIIimgGUI.AutoIntFrame(G2frame, PollTime=30.0)
    Creates a wx.Frame window for the Image AutoIntegration. The intent is that this will be used as a non-modal dialog window.

    Implements a Start button that morphs into a pause and resume button. This button starts a processing loop that is repeated every PollTime() seconds.

    Parameters
        • G2frame (wx.Frame) – main GSAS-II frame
• **PollTime** *(float)* – frequency in seconds to repeat calling the processing loop. (Default is 30.0 seconds.)

**EnableButtons** *(flag)*
Relabels and enable/disables the buttons at window bottom when auto-integration is running

**IntegrateImage** *(img, useTA=None, useMask=None)*
Integrates a single image. Ids for created PWDR entries (more than one is possible) are placed in G2frame.IntgOutList

**OnPause**
Respond to Pause, changes text on button/Status line, if needed Stops timer self.Pause should already be True

**OnTimerLoop** *(event)*
A method that is called every PollTime() seconds that is used to check for new files and process them. Integrates new images. Also optionally sets up and computes PDF. This is called only after the “Start” button is pressed (then its label reads “Pause”).

**ResetFromTable** *(dist)*
Sets integration parameters based on values from the lookup table

**SetSourceDir** *(event)*
Use a dialog to get a directory for image files

**ShowMatchingFiles** *(value, invalid=False, **kwargs)*
Find and show images in the tree and the image files matching the image file directory (self.params['readdir']) and the image file filter (self.params['filter']) and add this information to the GUI list box

**StartLoop**
Prepare to start autointegration timer loop. Save current Image params for use in future integrations also label the window so users understand what is being used

**CheckPDFprm** *(ShowContents=False)*
Read in the PDF (.pdfprm) parameter file and check for problems. If ShowContents is True, a formatted text version of some of the file contents is returned. If errors are found, the return string will contain the string “Error:” at least once.

**GSASIIimgGUI.CleanupMasks** *(data)*
If a mask creation is not completed, an empty mask entry is created in the masks array. This cleans them out. It is called when the masks page is first loaded and before saving them or after reading them in. This should also probably be done before they are used for integration.

**GSASIIimgGUI.DefineEvaluator** *(dlg)*
Creates a function that provides interpolated values for a given distance value

**GSASIIimgGUI.GetImageZ** *(G2frame, data, newRange=False)*

**class** **GSASIIimgGUI.ImgIntLstCtrl** *(parent, ID, pos=<sphinx.ext.autodoc.importer._MockObject object>, size=(1000, 200), style=0)*
Creates a custom ListCtrl for editing Image Integration parameters

**FillList** *(parms)*
Places the current parms into the table

**OnDouble** *(evt)*
respond to a double-click
class GSASIIimgGUI.IntegParmTable(parent, parms=None, IMfileList=None, readFileList=None)

Creates a dialog window with a table of integration parameters. `ShowModal()` will return `wx.ID_OK` if the process has been successful. In this case, `DefineEvaluator()` should be called to obtain a function that creates a dictionary with interpolated parameter values.

`ReadFiles(files)`
Reads a list of .imctrl files or a single .imtbl file

`ReadImageParmTable()`
Reads possibly edited values from the ListCtrl table and returns a list of values for each column.

GSASIIimgGUI.ReadControls(filename)
read an image controls (.imctrl) file

GSASIIimgGUI.ReadMask(filename)
Read a mask (.immask) file

GSASIIimgGUI.Read_imctrl(imctrl_file)
Read an image control file and record control parms into a dict, with some simple type conversions

GSASIIimgGUI.UpdateImageControls(G2frame, data, masks, useTA=None, useMask=None, IntegrateOnly=False)
Shows and handles the controls on the “Image Controls” data tree entry

GSASIIimgGUI.UpdateMasks(G2frame, data)
Shows and handles the controls on the “Masks” data tree entry

GSASIIimgGUI.UpdateStressStrain(G2frame, data)
Shows and handles the controls on the “Stress/Strain” data tree entry

GSASIIimgGUI.testColumnMetadata(G2frame)
Test the column-oriented metadata parsing, as implemented at 1-ID, by showing results when using a .par and .lbs file pair.

- Select a .par file, if more than one in selected dir.
- Select the .lbs file, if more than one matching .par file.
- Parse the .lbs file, showing errors if encountered; loop until errors are fixed.
- Search for an image or a line in the .par file and show the results when interpreted

See `GSASIIfiles.readColMetadata()` for more details.

1.6.7 GSASIIpwdGUI: Powder Pattern GUI routines

Used to define GUI controls for the routines that interact with the powder histogram (PWDR) data tree items.

GSASIIpwdGUI.CopyPlotCtrls(G2frame)
Global copy: Copy plot controls from current histogram to others.

GSASIIpwdGUI.CopySelectedHistItems(G2frame)
Global copy: Copy items from current histogram to others.

GSASIIpwdGUI.GetHistsLikeSelected(G2frame)
Get the histograms that match the current selected one: The histogram prefix and data type (PXC etc.), the number of wavelengths and the instrument geometry (Debye-Scherrer etc.) must all match. The current histogram is not included in the list.

Parameters

G2frame (wx.Frame) – pointer to main GSAS-II data tree
GSASIIpwdGUI.IsHistogramInAnyPhase (G2frame, histoName)
Tests a Histogram to see if it is linked to any phases. Returns the name of the first phase where the histogram is used.

GSASIIpwdGUI.OptimizePDF (G2frame, data, showFit=True, maxCycles=5)
Optimize the PDF to minimize the difference between G(r) and the expected value for low r (-4 pi r #density).

class GSASIIpwdGUI.RDFDialog (parent)

GSASIIpwdGUI.SetCopyNames (histName, dataType, addNames=[])
Determine the items in the sample parameters that should be copied, depending on the histogram type and the instrument type.

GSASIIpwdGUI.SetDefaultREFDModel ()
Fills in default items for the REFD Models dictionary which are defined as follows for each layer:

• Name: name of substance
• Thick: thickness of layer in Angstroms (not present for top & bottom layers)
• Rough: upper surface roughness for layer (not present for toplayer)
• Penetration: mixing of layer substance into layer above-is this needed?
• DenMul: multiplier for layer scattering density (default = 1.0)

Top layer defaults to vacuum (or air/any gas); can be substituted for some other substance.
Bottom layer default: infinitely thick Silicon; can be substituted for some other substance.

GSASIIpwdGUI.SetDefaultSASDModel ()
Fills in default items for the SASD Models dictionary

GSASIIpwdGUI.SetDefaultSubstances ()
Fills in default items for the SASD Substances dictionary

GSASIIpwdGUI.SetupSampleLabels (histName, dataType, histType)
Setup a list of labels and number formatting for use in labeling sample parameters. :param str histName: Name of histogram, ("PWDR ...") :param str dataType:

class GSASIIpwdGUI.SubCellsDialog (parent, title, controls, SGData, items, phaseDict)

GSASIIpwdGUI.UpdateBackground (G2frame, data)
respond to selection of PWDR background data tree item.

GSASIIpwdGUI.UpdateIndexPeaksGrid (G2frame, data)
respond to selection of PWDR Index Peak List data tree item.

GSASIIpwdGUI.UpdateInstrumentGrid (G2frame, data)
respond to selection of PWDR/SASD/REFD Instrument Parameters data tree item.

GSASIIpwdGUI.UpdateLimitsGrid (G2frame, data, plottype)
respond to selection of PWDR Limits data tree item.

GSASIIpwdGUI.UpdateModelsGrid (G2frame, data)
respond to selection of SASD Models data tree item.

GSASIIpwdGUI.UpdatePDFGrid (G2frame, data)
respond to selection of PWDR PDF data tree item.

GSASIIpwdGUI.UpdatePeakGrid (G2frame, data)
respond to selection of PWDR powder peaks data tree item.

GSASIIpwdGUI.UpdateREFDModelsGrid (G2frame, data)
respond to selection of REFD Models data tree item.
GSASIIpwdGUI.UpdateReflectionGrid(G2frame, data, HKLF=False, Name="")
respond to selection of PWDR Reflections data tree item by displaying a table of reflections in the data window.

GSASIIpwdGUI.UpdateSampleGrid(G2frame, data)
respond to selection of PWDR/SASD Sample Parameters data tree item.

GSASIIpwdGUI.UpdateSubstanceGrid(G2frame, data)
respond to selection of SASD/REFD Substance data tree item.

GSASIIpwdGUI.UpdateUnitCellsGrid(G2frame, data)
respond to selection of PWDR Unit Cells data tree item.

GSASIIpwdGUI.computePDF(G2frame, data)
Calls GSASIIpwd.CalcPDF() to compute the PDF and put into the data tree array. Called from OnComputePDF and OnComputeAllPDF and OnComputeAllPDF in GSASIIimgGUI.py

1.6.8 GSASIIrestrGUI: Restraint GUI routines

Used to define restraints.

GSASIIrestrGUI.GetSelectedRows(widget)
Returns a list of selected rows. Rows can be selected, blocks of cells or individual cells can be selected. The column for selected cells is ignored.

GSASIIrestrGUI.UpdateRestraints(G2frame, data, phaseName)
Respond to selection of the Restraints item on the data tree

1.6.9 GSASIIexprGUI: Expression Handling

This module defines a class for defining an expression in terms of values in a parameter dictionary via a wx.Dialog. The dialog creates a GSASII.ExpressionObj which is used to evaluate the expression against a supplied parameter dictionary.

The expression is parsed to find variables used in the expression and then the user is asked to assign parameters from the dictionary to each variable.

Default expressions are read from file DefaultExpressions.txt using GSASIIpath.LoadConfigFile().

class GSASIIexprGUI.AngleDialog(parent, Phases, parmDict, exprObj=None, header='Enter restraint expression here', wintitle='Expression Editor', VarLabel=None, depVarDict=None, ExtraButton=None, usedVars=[])

A wx.Dialog that allows a user to select a bond angle to be evaluated. What needs to be done here? Need phase info for atom 0. Select phase 1. Select 1st atom from dialog 2. Find neighbors & select two from dialog 3. Set up angle equation & save it - has to look like result from Show in above ExpressionDialog Use existing angle & esd calculate routines

class GSASIIexprGUI.BondDialog(parent, Phases, parmDict, exprObj=None, header='Enter restraint expression here', wintitle='Expression Editor', VarLabel=None, depVarDict=None, ExtraButton=None, usedVars=[])

A wx.Dialog that allows a user to select a bond length to be evaluated. What needs to be done here? Need phase info for atoms 0. Select phase 1. Select 1st atom from dialog 2. Find neighbors & select one from dialog 3. Set up distance equation & save it - has to look like result from Show in above ExpressionDialog Use existing bond & esd calculate routines
class GSASIIexprGUI.ExpressionDialog (parent, parmDict, exprObj=None, header='Enter restraint expression here', wintitle='Expression Editor', fit=True, VarLabel=None, depVarDict=None, ExtraButton=None, usedVars=[], wildCard=True)

A wx.Dialog that allows a user to input an arbitrary expression to be evaluated and possibly minimized.

To do this, the user assigns a new (free) or existing GSAS-II parameter to each parameter label used in the expression. The free parameters can optionally be designated to be refined. For example, is an expression is used such as:

\[ A \times \exp(-B/C) \]

then A, B and C can each be assigned as Free parameter with a user-selected value or to any existing GSAS-II variable in the parameter dictionary. As the expression is entered it is checked for validity.

After the ExpressionDialog object is created, use Show() to run it and obtain a GSASIIObj. ExpressionObj object with the user input.

Parameters

- **parent (wx.Frame)** – The parent of the Dialog. Can be None, but better is to provide the name of the Frame where the dialog is called.
- **parmDict (dict)** – a dict with defined parameters and their values. Each value may be a list with parameter values and a refine flag or may just contain the parameter value (non-float/int values in dict are ignored)
- **exprObj** – a GSASIIObj.ExpressionObj object with an expression and label assignments or None (default)
- **wintitle (str)** – String placed on title bar of dialog; defaults to “Expression Editor”
- **header (str)** – String placed at top of dialog to tell the user what they will do here; default is “Enter restraint expression here”
- **fit (bool)** – determines if the expression will be used in fitting (default=True). If set to False, refinement flags are not shown and Free parameters are not offered as an assignment option.
- **VarLabel (str)** – an optional variable label to include before the expression input. Ignored if None (default)
- **depVarDict (list)** – a dict of choices for the dependent variable to be fitted to the expression and their values. Ignored if None (default).
- **ExtraButton (list)** – a list with two terms that define [0]: the label for an extra button and [1] the callback routine to be used when the button is pressed. The button will only be enabled when the OK button can be used (meaning the equation/expression is valid). The default is None, meaning this will not be used.
- **usedVars (list)** – defines a list of previously used variable names. These names will not be reused as defaults for new free variables. (The default is an empty list).
- **wildCard (bool)** – If True (default), histogram names are converted to wildcard values, as is appropriate for the sequential refinement table

CheckVars ()
Check that appropriate variables are defined for each symbol used in self.expr

Returns a text error message or None if all needed input is present

GetDepVar ()
Returns the name of the dependent variable, when depVarDict is used.
OnChar(event)
Called as each character is entered. Cancels any running timer and starts a new one. The timer causes a check of syntax after 2 seconds without input. Disables the OK button until a validity check is complete.

OnChoice(event)
Respond to a selection of a variable type for a label in an expression

OnDepChoice(event)
Respond to a selection of a variable type for a label in an expression

OnValidate(event)
Respond to a press of the Validate button or when a variable is associated with a label (in OnChoice())

Repaint(exprObj)
Redisplay the variables and continue the validation

RestartTimer()
Cancels any running timer and starts a new one. The timer causes a check of syntax after 2 seconds unless there is further input. Disables the OK button until a validity check is complete.

SelectG2var(sel, var, parmList)
Offer a selection of a GSAS-II variable.

Parameters
  sel (int) – Determines the type of variable to be selected. Where 1 is used for Phase variables, and 2 for Histogram/Phase vars, 3 for Histogram vars and 4 for Global vars.

Returns
  a variable name or None (if Cancel is pressed)

Show(mode=True)
Call to use the dialog after it is created.

Returns
  None (On Cancel) or a new ExpressionObj

depVarDict = None
dict for dependent variables

dependentVar = None
name for dependent variable selection, when depVarDict is specified

expr = None
Expression as a text string

exprVarList = None
A list containing the variables utilized in the current expression. Placed into a GSASIIobj. ExpressionObj object when the dialog is closed with “OK”, saving any changes.

parmDict = None
A copy of the G2 parameter dict (parmDict) except only numerical values are included and only the value (not the vary flag, if present) is included.

setEvalResult(msg)
Show a string in the expression result area

showError(msg1, msg2=”, msg3=”)
Show an error message of 1 to 3 sections. The second section is shown in an equally-spaced font.

Parameters
  • msg1 (str) – msg1 is shown in a the standard font
  • msg2 (str) – msg2 is shown in a equally-spaced (wx.MODERN) font
  • msg3 (str) – msg3 is shown in a the standard font

1.6. GSAS-II GUI Submodules
usedVars = None
variable names that have been used and should not be reused by default
varName = None
Name assigned to each variable
varRefflag = None
Refinement flag for a variable (Free parameters only)
varSelect = None
A dict that shows the variable type for each label found in the expression.
  • If the value is None or is not defined, the value is not assigned.
  • If the value is 0, then the variable is “free” – a new refineable parameter.
  • Values above 1 determine what variables will be shown when the option is selected.
varValue = None
Value for a variable (Free parameters only)

GSASIIexprGUI.IndexParmDict (parmDict, wildcard)
Separate the parameters in parmDict into list of keys by parameter type.

Parameters
  • parmDict (dict) – a dict with GSAS-II parameters
  • wildcard (bool) – True if wildcard versions of parameters should be generated and added to the lists

Returns a dict of lists where key 1 is a list of phase parameters, 2 is histogram/phase parms, 3 is histogram parms and 4 are global parameters

GSASIIexprGUI.LoadDefaultExpressions()
Read a configuration file with default expressions from all files named DefaultExpressions.txt found in the path. Duplicates are removed and expressions are sorted alphabetically

1.6.10 GSASIIfpaGUI: Fundamental Parameters Routines

This module contains routines for getting Fundamental Parameters Approach (FPA) input, setting up for running the NIST XRD Fundamental Parameters Code, plotting the convolutors and computing a set of peaks generated by that code.

GSASIIfpaGUI.BBPSDDetector = [('lpsd_th2_angular_range', 3.0, 'Angular range observed by PSD'), ('lpsd_equitorial_divergence', 0.1, 'Equatorial divergence of the primary beam (degrees)')]
Additional FPA dict entries used in MakeTopasFPASizer() needed for Bragg Brentano instruments with linear (1-D) PSD detectors.

GSASIIfpaGUI.BBPointDetector = [('receiving_slit_width', 0.2, 'Width of receiving slit (mm)')]
Additional FPA dict entries used in MakeTopasFPASizer() needed for Bragg Brentano instruments with point detectors.

GSASIIfpaGUI.BraggBrentanoParms = [('divergence', 0.5, 'Bragg–Brentano divergence angle (deg)'), ('soller_angle', 2.0, 'Soller divergence angle for PSD'), ('tube_tails_width', 1.0, 'Tube tails width, in projection (mm)'), ('tube-tails_rel-I', 0.001, 'Tube tails fractional intensity (no units)')]
FPA dict entries used in MakeTopasFPASizer(). Tuple contains a dict key, a default value and a description. These are the parameters needed for all Bragg Brentano instruments

GSASIIfpaGUI.MakeSimSizer (G2frame, dlg)
Create a GUI to get simulation with parameters for Fundamental Parameters fitting.

Parameters
dlg (wx.Window) – Frame or Dialog where GUI will appear

Returns a sizer with the GUI controls
GSASIIfpagaGUI.MakeTopasFPASizer(*G2frame, FPdlg, mode, SetButtonStatus*)
Create a GUI with parameters for the NIST XRD Fundamental Parameters Code. Parameter input is modeled after Topas input parameters.

**Parameters**
- **FPdlg** (*wx.Window*) – Frame or Dialog where GUI will appear
- **mode** (*str*) – either ‘BBpoint’ or ‘BBPSD’ for Bragg-Brentano point detector or (linear) position sensitive detector
- **parmDict** (*dict*) – dict to place parameters. If empty, default values from globals BraggBrentanoParms, BBPointDetector & BBPSDDetector will be placed in the array.

**Returns** a sizer with the GUI controls

GSASIIfpaGUI.NISTparms = {}
Parameters in a nested dict, with an entry for each convolutor. Entries in those dicts have values in SI units (of course). NISTparms can be can be input directly or can be from created from parmDict by XferFPAsettings()

GSASIIfpaGUI.SetCu2Wave()
Set the parameters to the two-line Cu K alpha 1+2 spectrum

GSASIIfpaGUI.XferFPAsettings(*InpParms*)
convert Topas-type parameters to SI units for NIST and place in a dict sorted according to use in each convoluter

**Parameters** *InpParms* (*dict*) – a dict with Topas-like parameters, as set in MakeTopasFPASizer()

**Returns** a nested dict with global parameters and those for each convolution

GSASIIfpaGUI.doFPAcalc(*NISTpk, ttArr, twotheta, calcwid, step*)
Compute a single peak using a NIST profile object

**Parameters**
- **NISTpk** (*object*) – a peak profile computational object from the NIST XRD Fundamental Parameters Code, typically established from a call to SetupFPAcalc()
- **ttArr** (*np.Array*) – an evenly-spaced grid of two-theta points (degrees)
- **twotheta** (*float*) – nominal center of peak (degrees)
- **calcwid** (*float*) – width to perform convolution (degrees)
- **step** (*float*) – step size

GSASIIfpaGUI.parmDict = {'int': [0: 0.653817, 1: 0.346183], 'lwidth': [0: 0.501844, 1: 0.626579], ...}
Parameter dict used for reading Topas-style values. These are converted to SI units and placed into NISTparms

GSASIIfpaGUI.setupFPAcalc()
Create a peak profile object using the NIST XRD Fundamental Parameters Code.

**Returns** a profile object that can provide information on each convolution or compute the composite peak shape.

GSASIIfpaGUI.simParms = {}
Parameters to set range for pattern simulation
1.7 GSAS-II Structure Submodules

1.7.1 GSASIIstrMain: main structure routine

GSASIIstrMain.BestPlane(PlaneData)
Needs a doc string

GSASIIstrMain.DisAglTor(DATData)
Needs a doc string

GSASIIstrMain.PrintDistAngle(DisAglCtls, DisAglData, out=<_io.TextIOWrapper
name='<stdout>' mode='w' encoding='UTF-8'>)
Print distances and angles

Parameters

• DisAglCtls (dict) – contains distance/angle radii usually defined using
  GSASIIctrlGUI.DisAglDialog()

• DisAglData (dict) – contains phase data: Items ‘OrigAtoms’ and ‘TargAtoms’ con-
tain the atoms to be used for distance/angle origins and atoms to be used as targets. Item
‘SGData’ has the space group information (see Space Group object)

• out (file) – file object for output. Defaults to sys.stdout.

GSASIIstrMain.Refine(GPXfile, dlg=None, makeBack=True, refPlotUpdate=None)
Global refinement – refines to minimize against all histograms

GSASIIstrMain.RefineCore(Controls, Histograms, Phases, restraintDict, rigidbodyDict, parmDict,
varyList, calcControls, pawleyLookup, ifSeq, printFile, dlg, refPlotUp-
date=None)
Core optimization routines, shared between SeqRefine and Refine

Returns 5-tuple of ifOk (bool), Rvals (dict), result, covMatrix, sig

GSASIIstrMain.RetDistAngle(DisAglCtls, DisAglData)
Compute and return distances and angles

Parameters

• DisAglCtls (dict) – contains distance/angle radii usually defined using
  GSASIIctrlGUI.DisAglDialog()

• DisAglData (dict) – contains phase data: Items ‘OrigAtoms’ and ‘TargAtoms’ con-
tain the atoms to be used for distance/angle origins and atoms to be used as targets. Item
‘SGData’ has the space group information (see Space Group object)

Returns

AtomLabels,DistArray,AngArray where:

AtomLabels is a dict of atom labels, keys are the atom number

DistArray is a dict keyed by the origin atom number where the value is a list of distance entries.
The value for each distance is a list containing:

0) the target atom number (int);
1) the unit cell offsets added to x,y & z (tuple of int values)
2) the symmetry operator number (which may be modified to indicate centering and center of
   symmetry)
3) an interatomic distance in Å (float)
4) an uncertainty on the distance in Å or 0.0 (float)

**AngArray** is a dict keyed by the origin (central) atom number where the value is a list of angle entries. The value for each angle entry consists of three values:

0) a distance item reference for one neighbor (int)
1) a distance item reference for a second neighbor (int)
2) a angle, uncertainty pair; the s.u. may be zero (tuple of two floats)

The AngArray distance reference items refer directly to the index of the items in the DistArray item for the list of distances for the central atom.

```
GSASIIstrMain.SeqRefine(GPXfile, dlg, refPlotUpdate=None)
Perform a sequential refinement – cycles through all selected histograms, one at a time
```

```
GSASIIstrMain.main()
Called to run a refinement when this module is executed
```

```
GSASIIstrMain.phaseCheck(phaseVary, Phases, histogram)
Removes unused parameters from phase varylist if phase not in histogram #TODO - implement “Fix FXU” for seq refinement here - done?
```

### 1.7.2 GSASIIstrMath - structure math routines

```
GSASIIstrMath.ApplyRBModelDervs(dFdvDict, parmDict, rigidbodyDict, Phase)
Needs a doc string
```

```
GSASIIstrMath.ApplyRBModels(parmDict, Phases, rigidbodyDict, Update=False)
Takes RB info from RBModels in Phase and RB data in rigidbodyDict along with current RB values in parmDict & modifies atom contents (xyz & Uij) of parmDict
```

```
GSASIIstrMath.ApplyXYZshifts(parmDict, varyList)
takes atom x,y,z shift and applies it to corresponding atom x,y,z value
```

**Parameters**

- **parmDict** *(dict)* – parameter dictionary
- **varyList** *(list)* – list of variables (not used!)

**Returns** newAtomDict - dictionary of new atomic coordinate names & values; key is parameter shift name

```
GSASIIstrMath.Dict2Values(parmdict, varylist)
Use before call to leastsq to setup list of values for the parameters in parmdict, as selected by key in varylist
```

```
GSASIIstrMath.GetAbsorb(refl, im, hfx, calcControls, parmDict)
Needs a doc string
```

```
GSASIIstrMath.GetAbsorbDerv(refl, im, hfx, calcControls, parmDict)
Needs a doc string
```

```
GSASIIstrMath.GetAtomFXU(pfx, calcControls, parmDict)
Needs a doc string
```

```
GSASIIstrMath.GetAtomSSFXU(pfx, calcControls, parmDict)
Needs a doc string
```

```
GSASIIstrMath.GetFobsSq(Histograms, Phases, parmDict, calcControls)
Compute the observed structure factors for Powder histograms and store in reflection array Multiprocessing support added
```

### 1.7. GSAS-II Structure Submodules
GSASIIstrMath.GetHStrainShift (refl, im, SGData, phfx, hfx, calcControls, parmDict)
   Needs a doc string

GSASIIstrMath.GetHStrainShiftDerv (refl, im, SGData, phfx, hfx, calcControls, parmDict)
   Needs a doc string

GSASIIstrMath.GetIntensityCorr (refl, im, uniq, G, g, pf, phfx, hfx, SGData, calcControls, parmDict)
   Needs a doc string

GSASIIstrMath.GetIntensityDerv (refl, im, wave, uniq, G, g, pf, phfx, hfx, SGData, calcControls, parmDict)
   Needs a doc string

GSASIIstrMath.GetNewCellParms (parmDict, varyList)
   Compute unit cell tensor terms from varied Aij and Dij values. Terms are included in the dict only if Aij or Dij is varied.

GSASIIstrMath.GetPrefOri (uniq, G, g, phfx, hfx, SGData, calcControls, parmDict)
   March-Dollase preferred orientation correction

GSASIIstrMath.GetPrefOriDerv (refl, im, uniq, G, g, phfx, hfx, SGData, calcControls, parmDict)
   Needs a doc string

GSASIIstrMath.GetPwdrExt (refl, im, pf, phfx, hfx, calcControls, parmDict)
   Needs a doc string

GSASIIstrMath.GetPwdrExtDerv (refl, im, pf, phfx, hfx, calcControls, parmDict)
   Needs a doc string

GSASIIstrMath.GetReflPos (refl, im, wave, A, pf, hfx, calcControls, parmDict)
   Needs a doc string

GSASIIstrMath.GetReflPosDerv (refl, im, wave, A, pf, hfx, calcControls, parmDict)
   Needs a doc string

GSASIIstrMath.GetSampleSigGam (refl, im, wave, G, GB, SGData, hfx, phfx, calcControls, parmDict)
   Needs a doc string

GSASIIstrMath.GetSampleSigGamDerv (refl, im, wave, G, GB, SGData, hfx, phfx, calcControls, parmDict)
   Needs a doc string

GSASIIstrMath.HessRefine (values, HistoPhases, parmDict, varylist, calcControls, pawleyLookup, dlg)
   Loop over histograms and compute derivatives of the fitting model (M) with respect to all parameters. For each histogram, the Jacobian matrix, dMdV, with dimensions (n by m) where n is the number of parameters and m is the number of data points in the histogram. The (n by n) Hessian is computed from each Jacobian and it is returned. This routine is used when refinement derivatives are selected as “analytic Hessian” in Controls.

   Returns Vec, Hess where Vec is the least-squares vector and Hess is the Hessian

GSASIIstrMath.MagStructureFactor2 (refDict, G, hfx, pf, SGData, calcControls, parmDict)
   Compute neutron magnetic structure factors for all h,k,l for phase puts the result, F^2, in each ref[8] in refList operates on blocks of 100 reflections for speed input:

   Parameters
   - refDict (dict) – where ‘RefList’ list where each ref = h,k,l, it,d,... ‘FF’ dict of form factors - filed in below
   - G (np.array) – reciprocal metric tensor
   - pf (str) – phase id string
• **SGData** (*dict*) – space group info. dictionary output from SpcGroup

• **calcControls** (*dict*) –

• **ParmDict** (*dict*) –

  **Returns** copy of new refList - used in calculating numerical derivatives

**GSASIIstrMath.MagStructureFactorDerv**(*refDict, G, hfx, pfx, SGData, calcControls, parmDict*)

Compute nonmagnetic structure factor derivatives on blocks of reflections in magnetic structures - for powders/nontwins only input:

**Parameters**

• **refDict** (*dict*) – where ‘RefList’ list where each ref = h,k,l,it,d,… ‘FF’ dict of form factors - filled in below

• **G** (*np.array*) – reciprocal metric tensor

• **hfx** (*str*) – histogram id string

• **pfx** (*str*) – phase id string

• **SGData** (*dict*) – space group info. dictionary output from SpcGroup

• **calcControls** (*dict*) –

• **parmDict** (*dict*) –

  **Returns** dict *dFdvDict*: dictionary of derivatives

**GSASIIstrMath.MagStructureFactorDerv2**(*refDict, G, hfx, pfx, SGData, calcControls, parmDict*)

Compute magnetic structure factor derivatives numerically - for powders/nontwins only input:

**Parameters**

• **refDict** (*dict*) – where ‘RefList’ list where each ref = h,k,l,it,d,… ‘FF’ dict of form factors - filled in below

• **G** (*np.array*) – reciprocal metric tensor

• **hfx** (*str*) – histogram id string

• **pfx** (*str*) – phase id string

• **SGData** (*dict*) – space group info. dictionary output from SpcGroup

• **calcControls** (*dict*) –

• **parmDict** (*dict*) –

  **Returns** dict *dFdvDict*: dictionary of magnetic derivatives

**GSASIIstrMath.SCExtinction**(*ref, im, phfx, hfx, pfx, calcControls, parmDict, varyList*)

Single crystal extinction function; returns extinction & derivative

**GSASIIstrMath.SHP0cal**(*refl, im, g, phfx, hfx, SGData, calcControls, parmDict*)

spherical harmonics preferred orientation (cylindrical symmetry only)

**GSASIIstrMath.SHP0calDerv**(*refl, im, g, phfx, hfx, SGData, calcControls, parmDict*)

spherical harmonics preferred orientation derivatives (cylindrical symmetry only)

**GSASIIstrMath.SHTXcal**(*refl, im, g, pfx, hfx, SGData, calcControls, parmDict*)

Spherical harmonics texture

**GSASIIstrMath.SHTXcalDerv**(*refl, im, g, pfx, hfx, SGData, calcControls, parmDict*)

Spherical harmonics texture derivatives
GSASIIstrMath.\texttt{SStructureFactor}\((\text{refDict, } G, \text{hfx, pfx, SGData, SSGData, calcControls, parmDict})\)

Compute super structure factors for all \(h,k,l,m\) for phase - no twins puts the result, \(F^2\), in each ref[9] in refList works on blocks of 32 reflections for speed input:

**Parameters**

- \texttt{refDict (dict)} – where ‘RefList’ list where each ref = \(h,k,l,m,\ldots\) ‘FF’ dict of form factors - filed in below
- \(G(np.array)\) – reciprocal metric tensor
- \texttt{pfx (str)} – phase id string
- \texttt{SGData (dict)} – space group info. dictionary output from SpcGroup
- \texttt{calcControls (dict)} –
- \texttt{ParmDict (dict)} –

GSASIIstrMath.\texttt{SStructureFactorDerv}\((\text{refDict, im, G, hfx, pfx, SGData, SSGData, calcControls, parmDict})\)

Compute super structure factor derivatives for all \(h,k,l,m\) for phase - no twins Only Fourier component are done analytically here input:

**Parameters**

- \texttt{refDict (dict)} – where ‘RefList’ list where each ref = \(h,k,l,m,\ldots\) ‘FF’ dict of form factors - filed in below
- \texttt{im (int)} – = 1 (could be eliminated)
- \(G(np.array)\) – reciprocal metric tensor
- \texttt{hfx (str)} – histogram id string
- \texttt{pfx (str)} – phase id string
- \texttt{SGData (dict)} – space group info. dictionary output from SpcGroup
- \texttt{SSGData (dict)} – super space group info.
- \texttt{calcControls (dict)} –
- \texttt{ParmDict (dict)} –

**Returns** dict \(dFdvDict\): dictionary of derivatives

GSASIIstrMath.\texttt{SStructureFactorDerv2}\((\text{refDict, im, G, hfx, pfx, SGData, SSGData, calcControls, parmDict})\)

Compute super structure factor derivatives for all \(h,k,l,m\) for phase - no twins input:

**Parameters**

- \texttt{refDict (dict)} – where ‘RefList’ list where each ref = \(h,k,l,m,\ldots\) ‘FF’ dict of form factors - filed in below
- \texttt{im (int)} – = 1 (could be eliminated)
- \(G(np.array)\) – reciprocal metric tensor
- \texttt{hfx (str)} – histogram id string
- \texttt{pfx (str)} – phase id string
- \texttt{SGData (dict)} – space group info. dictionary output from SpcGroup
- \texttt{SSGData (dict)} – super space group info.
- \texttt{calcControls (dict)} –
• ParmDict (dict) –

Returns dict dFdvDict: dictionary of derivatives

GSASIIstrMath.SStructureFactorDervTw (refDict, im, G, hfx, pfx, SGData, SSGData, calcControls, parmDict)

Needs a doc string

GSASIIstrMath.SStructureFactorTw (refDict, G, hfx, pfx, SGData, SSGData, calcControls, parmDict)

Compute super structure factors for all h,k,l,m for phase - twins only puts the result, F^2, in each ref[8+im] in refList works on blocks of 32 reflections for speed input:

Parameters
• refDict (dict) – where ‘RefList’ list where each ref = h,k,l,m,it,d,… ‘FF’ dict of form factors - filed in below
• G (np.array) – reciprocal metric tensor
• pfx (str) – phase id string
• SGData (dict) – space group info. dictionary output from SpcGroup
• calcControls (dict) –
• ParmDict (dict) –

GSASIIstrMath.StructureFactor2 (refDict, G, hfx, pfx, SGData, calcControls, parmDict)

Compute structure factors for all h,k,l for phase puts the result, F^2, in each ref[8] in refList operates on blocks of 100 reflections for speed input:

Parameters
• refDict (dict) – where ‘RefList’ list where each ref = h,k,l,it,d,… ‘FF’ dict of form factors - filed in below
• G (np.array) – reciprocal metric tensor
• pfx (str) – phase id string
• SGData (dict) – space group info. dictionary output from SpcGroup
• calcControls (dict) –
• ParmDict (dict) –

GSASIIstrMath.SStructureFactorDerv2 (refDict, G, hfx, pfx, SGData, calcControls, parmDict)

Compute structure factor derivatives on blocks of reflections - for powders/nontwins only faster than StructureFactorDerv - correct for powders/nontwins!! input:

Parameters
• refDict (dict) – where ‘RefList’ list where each ref = h,k,l,it,d,… ‘FF’ dict of form factors - filed in below
• G (np.array) – reciprocal metric tensor
• hfx (str) – histogram id string
• pfx (str) – phase id string
• SGData (dict) – space group info. dictionary output from SpcGroup
• calcControls (dict) –
• parmDict (dict) –

1.7. GSAS-II Structure Submodules
Returns dict dFdvDict: dictionary of derivatives

GSASIIstrMath. StructureFactorDervTw2 (refDict, G, hfx, pfx, SGData, calcControls, parmDict)
Compute structure factor derivatives on blocks of reflections - for twins only faster than StructureFactorDervTw input:

Parameters

- refDict (dict) – where ‘RefList’ list where each ref = h,k,l,it,d,… ‘FF’ dict of form factors - filled in below
- G (np.array) – reciprocal metric tensor
- hfx (str) – histogram id string
- pfx (str) – phase id string
- SGData (dict) – space group info. dictionary output from SpcGroup
- calcControls (dict) –
- parmDict (dict) –

Returns dict dFdvDict: dictionary of derivatives

GSASIIstrMath. Values2Dict (parmdict, varylist, values)
Use after call to leastsq to update the parameter dictionary with values corresponding to keys in varylist

GSASIIstrMath. dervHKLF (Histogram, Phase, calcControls, varylist, parmDict, rigidbodyDict)
Loop over reflections in a HKLF histogram and compute derivatives of the fitting model (M) with respect to all parameters. Independent and dependant dM/dp arrays are returned to either dervRefine or HessRefine.

Returns

GSASIIstrMath. dervRefine (values, HistoPhases, parmDict, varylist, calcControls, pawleyLookup, dlg=None)
Loop over histograms and compute derivatives of the fitting model (M) with respect to all parameters. Results are returned in a Jacobian matrix (aka design matrix) of dimensions (n by m) where n is the number of parameters and m is the number of data points. This can exceed memory when m gets large. This routine is used when refinement derivatives are selected as “analytic Jacobian” in Controls.

Returns Jacobian numpy.array dMdv for all histograms concatenated

GSASIIstrMath. errRefine (values, HistoPhases, parmDict, varylist, calcControls, pawleyLookup, dlg=None)
Computes the point-by-point discrepancies between every data point in every histogram and the observed value. Used in the Jacobian, Hessian & numeric least-squares to compute function

Returns an np array of differences between observed and computed diffraction values.

GSASIIstrMath. getPowderProfile (parmdict, x, varylist, Histogram, Phases, calcControls, pawleyLookup, histogram=None)
Computes the powder pattern for a histogram based on contributions from all used phases

GSASIIstrMath. getPowderProfileDervMP (args)
Computes the derivatives of the computed powder pattern with respect to all refined parameters. Multiprocessing version.

GSASIIstrMath. penaltyDeriv (pNames, pVal, HistoPhases, calcControls, parmDict, varyList)
Compute derivatives on user-supplied and built-in restraint (penalty) functions

where pNames is list of restraint labels

returns pDerv with partial derivatives by variable# in varList and restraint# in pNames
(pDerv[variable#][restraint#])
GSASIIstrMath.\texttt{penaltyFxn}(\texttt{HistoPhases}, \texttt{calcControls}, \texttt{parmDict}, \texttt{varyList})
Compute user-supplied and built-in restraint functions

\section*{1.7.3 \texttt{GSASIIstrIO}: structure I/O routines}

Contains routines for reading from GPX files and printing to the .LST file. Used for refinements and in G2scriptable. Should not contain any wxpython references as this should be able to be used in non-GUI settings.

\texttt{GSASIIstrIO.GPXBackup}(\texttt{GPXfile}, \texttt{makeBack=True}) makes a backup of the specified .gpx file

\begin{itemize}
  \item \texttt{GPXfile (str)} – full .gpx file name
  \item \texttt{makeBack (bool)} – if True (default), the backup is written to a new file; if False, the last backup is overwritten
\end{itemize}

\textbf{Returns} the name of the backup file that was written

\texttt{GSASIIstrIO.GetAllPhaseData}(\texttt{GPXfile}, \texttt{PhaseName}) Returns the entire dictionary for PhaseName from GSASII gpx file

\begin{itemize}
  \item \texttt{GPXfile (str)} – full .gpx file name
  \item \texttt{PhaseName (str)} – phase name
\end{itemize}

\textbf{Returns} phase dictionary or None if PhaseName is not present

\texttt{GSASIIstrIO.GetConstraints}(\texttt{GPXfile}) Read the constraints from the GPX file and interpret them

called in \texttt{ReadCheckConstraints()}, \texttt{GSASIIstrMain.Refine()} and \texttt{GSASIIstrMain.SequentialRefine()}. \texttt{GSASIIstrIO.GetControls}(\texttt{GPXfile}) Returns dictionary of control items found in GSASII gpx file

\begin{itemize}
  \item \texttt{GPXfile (str)} – full .gpx file name
\end{itemize}

\textbf{Returns} dictionary of control items

\texttt{GSASIIstrIO.GetFprime}(\texttt{controlDict}, \texttt{Histograms}) Needs a doc string

\texttt{GSASIIstrIO.GetFullGPX}(\texttt{GPXfile}) Returns complete contents of GSASII gpx file. Used in \texttt{GSASIIscriptable.LoadDictFromProjFile()}. \texttt{Parameters} \texttt{GPXfile (str)} – full .gpx file name

\textbf{Returns} Project, nameList, where

\begin{itemize}
  \item Project (dict) is a representation of gpx file following the GSAS-II tree structure for each item: key = tree name (e.g. ‘Controls’, ‘Restraints’, etc.), data is dict
  \item nameList (list) has names of main tree entries & subentries used to reconstruct project file
\end{itemize}

\texttt{GSASIIstrIO.GetHistogramData}(\texttt{Histograms}, \texttt{Print=True}, \texttt{pFile=None}) needs a doc string

\section*{1.7. GSAS-II Structure Submodules}
**GSASIIstrIO.GetHistogramNames**(GPXfile, hTypes)

Returns a list of histogram names found in a GSAS-II .gpx file that match specified histogram types. Names are returned in the order they appear in the file.

**Parameters**

- **GPXfile**(str) – full .gpx file name
- **hTypes**(str) – list of histogram types

**Returns** list of histogram names (types = PWDR & HKLF)

**GSASIIstrIO.GetHistogramPhaseData**(Phases, Histograms, Print=True, pFile=None, resetRefList=True)

Loads the HAP histogram/phase information into dicts

**Parameters**

- **Phases**(dict) – phase information
- **Histograms**(dict) – Histogram information
- **Print**(bool) – prints information as it is read
- **pFile**(file) – file object to print to (the default, None causes printing to the console)
- **resetRefList**(bool) – Should the contents of the Reflection List be initialized on loading. The default, True, initializes the Reflection List as it is loaded.

**Returns** (hapVary,hapDict,controlDict) * hapVary: list of refined variables * hapDict: dict with refined variables and their values * controlDict: dict with fixed parameters

**GSASIIstrIO.GetHistograms**(GPXfile, hNames)

Returns a dictionary of histograms found in GSASII gpx file

**Parameters**

- **GPXfile**(str) – full .gpx file name
- **hNames**(str) – list of histogram names

**Returns** dictionary of histograms (types = PWDR & HKLF)

**GSASIIstrIO.GetPawleyConstr**(SGLaue, PawleyRef, im, pawleyVary)

need a doc string

**GSASIIstrIO.GetPhaseData**(PhaseData, RestraintDict={}, rbIds={}, Print=True, pFile=None, seqRef=False)

Setup the phase information for a structural refinement, used for regular and sequential refinements, optionally printing information to the .lst file (if Print is True)

**GSASIIstrIO.GetPhaseNames**(GPXfile)

Returns a list of phase names found under ‘Phases’ in GSASII gpx file

**Parameters** **GPXfile**(str) – full .gpx file name

**Returns** list of phase names

**GSASIIstrIO.GetRestraints**(GPXfile)

Read the restraints from the GPX file. Throws an exception if not found in the .GPX file

**GSASIIstrIO.GetRigid Bodies**(GPXfile)

Read the rigid body models from the GPX file

**GSASIIstrIO.GetRigidBodyModels**(rigidbodyDict, Print=True, pFile=None)

need a doc string
**GSASIIstrIO.GetSeqResult** *(GPXfile)*

Returns the sequential results table information from a GPX file. Called at the beginning of **GSASIIstrMain.SeqRefine()**

**Parameters**

- **GPXfile** *(str)* – full .gpx file name

**Returns**
a dict containing the sequential results table

**GSASIIstrIO.GetUsedHistogramsAndPhases** *(GPXfile)*

Returns all histograms that are found in any phase and any phase that uses a histogram. This also assigns numbers to used phases and histograms by the order they appear in the file.

**Parameters**

- **GPXfile** *(str)* – full .gpx file name

**Returns**

`(Histograms,Phases)`

- **Histograms** = dictionary of histograms as `{name:data,...}`
- **Phases** = dictionary of phases that use histograms

**GSASIIstrIO.IndexGPX** *(GPXfile, read=False)*

Create an index to a GPX file, optionally the file into memory. The byte size of the GPX file is saved. If this routine is called again, and if this size does not change, indexing is not repeated since it is assumed the file has not changed (this can be overridden by setting read=True).

**Parameters**

- **GPXfile** *(str)* – full .gpx file name

**Returns**

`Project,nameList` if `read=False`, where

- **Project** *(dict)* is a representation of gpx file following the GSAS-II tree structure for each item: key = tree name (e.g. ‘Controls’, ‘Restraints’, etc.), data is dict
- **nameList** *(list)* has names of main tree entries & subentries used to reconstruct project file

**GSASIIstrIO.PrintRestraints** *(cell, SGData, AtPtrs, Atoms, AtLookup, textureData, phaseRest, pFile)*

needs a doc string

**GSASIIstrIO.ProcessConstraints** *(constList)*

Interpret the constraints in the constList input into a dictionary, etc. All **GSASIIobj.G2VarObj** objects are mapped to the appropriate phase/hist/atoms based on the object internals (random Ids). If this can’t be done (if a phase has been deleted, etc.), the variable is ignored. If the constraint cannot be used due to too many dropped variables, it is counted as ignored.

**Parameters**

- **constList** *(list)* – a list of lists where each item in the outer list specifies a constraint of some form, as described in the **GSASIIobj.Constraint definition**.

**Returns**
a tuple of `(constDict,fixedList,ignored)` where:

- **constDict** *(list of dicts)* contains the constraint relationships
- **fixedList** *(list)* contains the fixed values for each type of constraint.
- **ignored** *(int)* counts the number of invalid constraint items (should always be zero!)

**GSASIIstrIO.ReadCheckConstraints** *(GPXfile, seqHist=None)*

Load constraints and related info and return any error or warning messages This is done from the GPX file rather than the tree.

### 1.7. GSAS-II Structure Submodules
Parameters seqHist (dict) – defines a specific histogram to be loaded for a sequential refinement, if None (default) all are loaded.

GSASIIstrIO.SaveUpdatedHistogramsAndPhases (GPXfile, Histigrams, Phases, RigidBodies, CovData)

Save phase and histogram information into “pseudo-gpx” files. The phase information is overwritten each time this is called, but histogram information is appended after each sequential step.

Parameters
- GPXfile (str) – full .gpx file name
- Histigrams (dict) – dictionary of histograms as {name:data,...}
- Phases (dict) – dictionary of phases that use histograms
- RigidBodies (dict) – dictionary of rigid bodies
- CovData (dict) – dictionary of refined variables, varyList, & covariance matrix

GSASIIstrIO.SetHistogramData (parmDict, sigDict, Histigrams, FFtables, Print=True, pFile=None)

needs a doc string

GSASIIstrIO.SetHistogramPhaseData (parmDict, sigDict, Phases, Histigrams, FFtables, Print=True, pFile=None)

needs a doc string

GSASIIstrIO.SetPhaseData (parmDict, sigDict, Phases, RBIds, covData, RestraintDict=None, pFile=None)

Called after a refinement to transfer parameters from the parameter dict to the phase(s) information read from a GPX file. Also prints values to the .lst file

GSASIIstrIO.SetRigidBodyModels (parmDict, sigDict, rigidbodyDict, pFile=None)

needs a doc string

GSASIIstrIO.SetSeqResult (GPXfile, Histigrams, SeqResult)

Places the sequential results information into a GPX file after a refinement has been completed. Called at the end of GSASIIstrMain.SeqRefine()

Parameters GPXfile (str) – full .gpx file name

GSASIIstrIO.SetUsedHistogramsAndPhases (GPXfile, Histigrams, Phases, RigidBodies, CovData, makeBack=True)

Updates gpxfile from all histograms that are found in any phase and any phase that used a histogram. Also updates rigid body definitions. This is used for non-sequential fits, but not for sequential fitting.

Parameters
- GPXfile (str) – full .gpx file name
- Histigrams (dict) – dictionary of histograms as {name:data,...}
- Phases (dict) – dictionary of phases that use histograms
- RigidBodies (dict) – dictionary of rigid bodies
- CovData (dict) – dictionary of refined variables, varyList, & covariance matrix
- makeBack (bool) – True if new backup of .gpx file is to be made; else use the last one made

GSASIIstrIO.SetupSeqSavePhases (GPXfile)

Initialize the files used to save intermediate results from sequential fits.
GSASIstrIO.ShowBanner(pFile=None)
Print authorship, copyright and citation notice

GSASIstrIO.ShowControls(Controls, pFile=None, SeqRef=False)
Print controls information

GSASIstrIO.cellFill(pfx, SGData, parmDict, sigDict)
Returns the filled-out reciprocal cell (A) terms and their uncertainties from the parameter and sig dictionaries.

Parameters
- **pfx** *(str)* – parameter prefix (“n::”, where n is a phase number)
- **SGdata** *(dict)* – a symmetry object
- **parmDict** *(dict)* – a dictionary of parameters
- **sigDict** *(dict)* – a dictionary of uncertainties on parameters

Returns
A, sigA where each is a list of six terms with the A terms

GSASIstrIO.cellVary(pfx, SGData)
Creates equivalences for a phase based on the Laue class. Returns a list of A tensor terms that are non-zero.

GSASIstrIO.getBackupName(GPXfile, makeBack)
Get the name for the backup .gpx file name

Parameters
- **GPXfile** *(str)* – full .gpx file name
- **makeBack** *(bool)* – if True the name of a new file is returned, if False the name of the last file that exists is returned

Returns
the name of a backup file

GSASIstrIO.getCellEsd(pfx, SGData, A, covData)
needs a doc string

GSASIstrIO.gpxSize = -1
Global variables used in IndexGPX() to see if file has changed (gpxSize) and to index where to find each 1st-level tree item in the file.

### 1.8 GSASIlmapvars: Parameter constraints

Module to implements algebraic contraints, parameter redefinition and parameter simplification contraints.

#### 1.8.1 Types of constraints

There are four ways to specify constraints, as listed below. Note that parameters are initially stored in the main section of the GSAS-II data tree under heading Constraints. This dict has four keys, ‘Hist’, ‘HAP’, ‘Global’, and ‘Phase’, each containing a list of constraints. An additional set of constraints are generated for each phase based on symmetry considerations by calling GSASIstrIO.GetPhaseData().

Note that in the constraints, as stored in the GSAS-II data tree, parameters are stored as GSASIObject.G2VarObj objects, as these objects allow for changes in numbering of phases, histograms and atoms. When they are interpreted (in GSASIstrIO.ProcessConstraints()), references to numbered objects are resolved using the appropriate random ids and the parameter object is converted to a string of form ph:hst::VARNAM:at.
Alternate parameters (New Var)

Parameter redefinition (“New Var” constraints) is done by creating an expression that relates several parameters:

\[
\begin{align*}
Mx1 \cdot Px + My1 \cdot Py + \ldots \\
Mx2 \cdot Px + Mz2 \cdot Pz + \ldots
\end{align*}
\]

where \( P_j \) is a GSAS-II parameter name and \( M_{jk} \) is a constant (float) multiplier. Alternately, multipliers \( M_{jk} \) can contain a formula (str) that will be evaluated prior to the start of the refinement. In a formula, GSAS-II parameters will be replaced by the value of the parameter before the formula is evaluated, so 'np.cos(0::Ax:2)' is a valid multiplier. At present, only phase (atom/cell) parameters are available for use in a formula, but this can be expanded if needed.

This type of constraint describes an alternate degree of freedom where parameter \( Px \) and \( Py \), etc. are varied to keep their ratio fixed according the expression. A new variable parameter is assigned to each degree of freedom when refined. An example where this can be valuable is when two parameters, \( P1 \) and \( P2 \), have similar values and are highly correlated. It is often better to create a new variable, \( Ps = P1 + P2 \), and refine \( Ps \). In the later stages of refinement, a second variable, \( Pd = P1 - P2 \), can be defined and it can be seen if refining \( Pd \) is supported by the data. Another use will be to define parameters that express “irrep modes” in terms of the fundamental structural parameters.

These “New Var” constraints are stored as described for type “f” in the constraint definitions table.

Constrained parameters (Const)

A constraint on a set of variables can be supplied in the form of a linear algebraic equation:

\[
\begin{align*}
Nx1 \cdot Px + Ny1 \cdot Py + \ldots &= C1 \\
&\vdots
\end{align*}
\]

where \( C_n \) is a constant (float), where \( P_j \) is a GSAS-II parameter name, and where \( N_{jk} \) is a constant multiplier (float) or a formula (str) that will be evaluated prior to the start of the refinement. In a formula, GSAS-II parameters will be replaced by the value of the parameter before the formula is evaluated, so 'np.cos(0::Ax:2)' is a valid multiplier. At present, only phase (atom/cell) parameters are available for use in a formula, but this can be expanded if needed.

These equations set an interdependence between parameters. Common uses of parameter constraints are to set rules that decrease the number of parameters, such as restricting the sum of fractional occupancies for atoms that share a site to sum to unity, thus reducing the effective number of variables by one. Likewise, the Uiso value for a H atom “riding” on a C, N or O atom can be related by a fixed offset (the so called B+1 “rule”).

A “Const” constraint is stored as described for type “c” in the constraint definitions table.

Equivalenced parameters (Equiv)

A simplified way to set up a constraint equation is to define an equivalence, which can be of form:

\[
\begin{align*}
C1 \cdot P1 &= C2 \cdot Py \\
&\vdots
\end{align*}
\]

or:

\[
\begin{align*}
C1 \cdot P1 &= C2 \cdot P2 = C3 \cdot P3 = \ldots
\end{align*}
\]

where \( C_n \) is a constant (float), where \( P_j \) is a GSAS-II parameter name. This means that parameters \( Py \) (or \( P2 \) and \( P3 \)) are determined from (or “slaved” to) parameter \( P1 \). Alternately, equivalences can be created with StoreEquivalence() where the multipliers can be a formula (str) that will be evaluated prior to the start of the refinement. In a formula, GSAS-II parameters will be replaced by the value of the parameter before the formula is evaluate, so 'np.cos(0::Ax:2)' is a valid multiplier. At present, only phase (atom/cell) parameters are available
for use in a formula, but this can be expanded if needed. Note that the latter constraint expression is conceptually identical to defining constraint equations. In practice, however, equivalenced parameters are processed in a different and more direct manner than constraint equations. The previous set of equalities could also be written in this way as a set of constraint equations:

\[
\begin{align*}
C_1 \times P_1 - C_2 \times P_2 &= 0 \\
C_1 \times P_1 - C_3 \times P_3 &= 0 \\
&\ldots
\end{align*}
\]

The first parameter (P1 above) is considered the independent variable and the remaining parameters are dependent variables. The dependent variables are set from the independent variable. An example of how this may be used would be if, for example, a material has a number of O atoms, all in fairly similar bonding environments and the diffraction data are sparse, one may reduce the complexity of the model by defining Uiso for the first O atom to be identical to the remaining atoms. The results of this refinement will be simpler to understand than if a set of constraint equations is used because the refined parameter will be the independent variable, which will be as ph::Uiso:n, corresponding to the first O atom.

A parameter can be used in multiple equivalences as independent variable, but if a parameter is used as both a dependent and independent variable or a parameter is used in equivalences and in “New Var” or “Const” constraints, this create conflicts that cannot be resolved within the equivalences implementation but can be handled as constraint equations. The equivalences that violate this are discovered in `CheckEquivalences()` and then `MoveConfEquiv()` is used to change these equivalences to “Const” equations.

Equivalenced parameters (“EQUIV” constraints), when defined by users, are stored as described for type “e” in the constraint definitions table. Other equivalences are generated by symmetry prior to display or refinement in `GSASIIstrIO.GetPhaseData()`. These are not stored.

**Fixed parameters (Hold)**

When parameters are refined where a single refinement flag determines that several variables are refined at the same time (examples are: cell parameters, atom positions, anisotropic displacement parameters, magnetic moments,…) it can be useful to specify that a specific parameter should not be varied. These will most commonly be generated due to symmetry, but under specific conditions, there may be other good reasons to constrain a parameter.

A “Hold” constraint is stored as described for type “h” in the constraint definitions table.

### 1.8.2 Constraint Processing

When constraints will be used or edited, they are processed using a series of calls:

- First all of the stored constraints are appended into a single list. They are initially stored in separate lists only to improve their creation and display in the GUI.
- Then `InitVars()` is used to initialize the global variables in this module (`GSASIImapvars`).
- Then `GSASIIstrIO.ProcessConstraints()` is used to initially process the constraints, as described below.
- Symmetry-generated equivalences are then created in `GSASIIstrIO.GetPhaseData()`, which also calls `GSASIIstrIO.cellVary()` and for Pawley refinements `GSASIIstrIO.GetPawleyConstr()`. These are entered directly into this module’s globals using `StoreEquivalence()`.
- Constraints/equivalences are then checked for possible conflicts with `CheckConstraints()`, this requires grouping the constraints, as described below.
- In refinements, `GenerateConstraints()` is then called to create the constraints that will be used, see below for
• For debugging constraints, `VarRemapShow()` can be called after `GenerateConstraints()` to display the generated constraints.

**Constraint Reorganization (ProcessConstraints())**

`GSASIIstrIO.ProcessConstraints()` is used to initially process the constraints. This does these things:

1. The “Hold”, “Const” and “New Var” expressions are split between two paired lists, `constDictList` and `fixedList` which are set:
   
   - For “Hold” entries a dict with a single entry is placed in `constDictList` where the key is the parameter name (associated value is 0.0) and `fixedList` gets a value of 0.0.
   - For “Const” entries, a dict with multiple entries is placed in `constDictList` where the key is the parameter name and the value is the multiplier for the parameter, while `fixedList` gets a string value corresponding to the constant value for the expression.
   - For “New Var” entries, a dict with multiple entries is placed in `constDictList` where the key is the parameter name and the value is the multiplier for the parameter; an additional key “_vary” is given the value of True or False depending on the refinement flag setting. The corresponding entry in `fixedList` is None

   The output from this will have this form where the first entry is a “Const”, the second is a “New Var” and the final is a “Hold”.

   ```python
   constrDict = {
   '0:12:Scale': 2.0, '0:14:Scale': 4.0, '0:13:Scale': 3.0, '0:0:Scale': -0.5,
   '2::C(10,6,1)': 1.0, '1::C(10,6,1)': 1.0, '_vary':True},
   '0::A0': 0.0}
   fixedList = ['5.0', None, '0']
   ```

2. Equivalences are stored using `StoreEquivalence()` into this module’s globals (`arrayList`, `invarrayList`, `indParmList`, `dependentParmList` and `symGenList`)

**Parameter Grouping (GenerateConstraints())**

Functions `CheckConstraints()` and `GenerateConstraints()` are used to process the parameter equivalences and constraint lists created in `ProcessConstraints()`. The former is used to generate error messages and the latter to generate the internal information used to apply the constraints.

Initially, in both a list of parameters that are fixed and those used in constraint relations are tabulated in `CheckEquivalences()`. The equivalence relations are the scanned for the following potential problems:

1. a parameter is used as a dependent variable in more than one equivalence relation
2. a parameter is fixed and used in an equivalence relation either as a dependent or independent variable
3. a parameter is used as a dependent variable in one equivalence relation and as an independent variable in another
4. a parameter is used in in an equivalence relation (either as a dependent or independent variable) and is used in a constraint expression
5. a parameter is not defined in a particular refinement, but is used in an equivalence relation
6. a parameter uses a wildcard for the histogram number (sequential refinements)

Cases 1 & 2 above cannot be corrected, and result in errors. Cases 3 & 4 are potentially corrected with `MoveConfEquiv()`, as described below. Case 5 causes the equivalence to be dropped. Case 6 causes the current histogram number to be substituted for the wildcard.
For cases 3 & 4, MoveConfEquiv() is used to change these equivalences into “Const” equations. This can potentially mean that additional equivalences will be problematic, so if there are changes made by MoveConfEquiv(), CheckEquivalences() is repeated. If any problem cases are noted, the refinement cannot be performed.

Constraint expressions (“Const” and “New Var”) are sorted into groups so that each group contains the minimum number of entries that ensures each parameter is referenced in only one group in GroupConstraints(). This is done by scanning the list of dicts in constDictList one by one and making a list of parameters used in that constraint expression. Any expression that contains a parameter in is in that list is added to the current group and those parameters are added to this list of parameters. The list of ungrouped expressions is then scanned again until no more expressions are added to the current group. This process is repeated until every expression has been placed in a group. Function GroupConstraints() returns two lists of lists. The first has, for each group, a list of the indices in constDictList that comprise the group (there can be only one). The second list contains, for each group, the unique parameter names in that group.

Each constraint group is then processed. First, wildcard parameters are renamed (in a sequential refinement). Any fixed parameters that are used in constraints are noted as errors. The number of refined parameters and the number of parameters that are not defined in the current refinement are also noted. It is fine if all parameters in a group are not defined or all are not varied, but if some are defined and others not or some are varied and others not, this constitutes an error.

The contents of each group is then examined. Groups with a single parameter (holds) are ignored. Then for each group, the number of parameters in the group (Np) and the number of expressions in the group (Nc) are counted and for each expression. If Nc > Np, then the constraint is overdetermined, which also constitutes an error.

The parameter multipliers for each expression are then assembled:

\[
\begin{align*}
M_{1a} \times P_1 + M_{2a} \times P_2 + \ldots + M_{ka} \times P_k \\
M_{1b} \times P_1 + M_{2b} \times P_2 + \ldots + M_{kb} \times P_k \\
\ldots \\
M_{1j} \times P_1 + M_{2j} \times P_2 + \ldots + M_{kj} \times P_k
\end{align*}
\]

From this it becomes possible to create a Nc x Np matrix, which is called the constraint matrix:

\[
\begin{pmatrix}
M_{1a} & M_{2a} & \ldots & M_{ka} \\
M_{1b} & M_{2b} & \ldots & M_{kb} \\
\ldots \\
M_{1j} & M_{2j} & \ldots & M_{kj}
\end{pmatrix}
\]

When Nc<Np, then additional rows need to be added to the matrix and to the vector that contains the value for each row (fixedList) where values are None for New Vars and a constant for fixed values. This then can describe a system of Np simultaneous equations:

\[
\begin{pmatrix}
M_{1a} & M_{2a} & \ldots & M_{ka} \\
M_{1b} & M_{2b} & \ldots & M_{kb} \\
\ldots \\
M_{1j} & M_{2j} & \ldots & M_{kj}
\end{pmatrix}
\begin{pmatrix}
P_1 \\
P_2 \\
\ldots \\
P_k
\end{pmatrix} =
\begin{pmatrix}
C_1 \\
C_2 \\
\ldots \\
C_k
\end{pmatrix}
\]

This is done by creating a square matrix from the group using _FillArray() with parameter FillDiagonals=False (the default). Any unspecified rows are left as all zero. The first Nc rows in the array are then coverted to row-echelon form using _RowEchelon(). This will create an Exception if any two rows are linearly dependent (which means that no matter what values are used for the remaining rows, that the matrix will be singular). _FillArray() is then called with parameter FillDiagonals=True, which again creates a square matrix but where unspecified rows are zero except for the diagonal elements. The Gram-Schmidt process, implemented in GramSchmidtOrtho(), is used to find orthonormal unit vectors for the remaining Np-Nc rows of the
matrix. This will fail with a ConstraintException if this is not possible (singular matrix) or the result is placed in constrArr as a numpy array.

Rows in the matrix corresponding to “New Var” constraints and those that were generated by the Gram-Schmidt process are provided with parameter names (this can be specified if a “New Var” entry by using a "_name" element in the constraint dict, but at present this is not implemented.) Names are generated using :param:`paramPrefix` which is set to "::constr", plus a number to make the new parameter name unique. :global:`genVarLookup` provides a lookup table, where the names of the parameters related to this new parameter can be looked up easily.

Finally the parameters used as input to the constraint are placed in this module’s :global:`dependentParmList` and the constraint matrix is placed in into :data:`arrayList`. This can be used to compute the initial values for “New Var” parameters. The inverse of the constraint matrix is placed in :data:`invarrayList` and a list of the “New Var” parameters and a list of the fixed values (as str’s) is placed in :data:`indParmList`. A lookup table for fixed values as floats is placed in :data:`fixedDict`. Finally the appropriate entry in :data:`symGenList` is set to False to indicate that this is not a symmetry generated constraint.

1.8.3 Externally-Accessible Routines

To define a set of constrained and unconstrained relations, one defines a list of dictionary defining constraint parameters and their values, a list of fixed values for each constraint and a list of parameters to be varied. In addition, one uses :func:`StoreEquivalence()` to define parameters that are equivalent. Use :func:` EvaluateMultipliers()` to convert formula-based constraint/ equivalence multipliers to numbers and then use :func:`CheckConstraints()` to check that the input is internally consistent and finally :func:`GroupConstraints()` and :func:` GenerateConstraints()` to generate the internally used tables. Routine :func:`Map2Dict()` is used to initialize the parameter dictionary and routine :func:`Dict2Map()`, :func:`Dict2Deriv()`, and :func:`ComputeDepESD()` are used to apply constraints. Routine :func:`VarRemapShow()` is used to print out the constraint information, as stored by :func:`GenerateConstraints()`. Further information on each routine is below:

**InitVars()** This is optionally used to clear out all defined previously defined constraint information

**StoreEquivalence()** To implement parameter redefinition, one calls StoreEquivalence. This should be called for every set of equivalence relationships. There is no harm in using StoreEquivalence with the same independent variable:

```python
StoreEquivalence('x',('y',))
StoreEquivalence('x',('z',))
```

or equivalently

```python
StoreEquivalence('x',('y','z'))
```

The latter will run more efficiently. Note that mixing independent and dependent variables would require assignments, such as

```python
StoreEquivalence('x',('y',))
StoreEquivalence('y',('z',))
```

would require that equivalences be applied in a particular order and thus is implemented as a constraint equation rather than an equivalence.

Use :func:`StoreEquivalence` before calling :func:`GenerateConstraints` or :func:`CheckConstraints`

**CheckConstraints()** check that input in internally consistent

**GenerateConstraints()** generate the internally used tables from constraints and equivalences
**EvaluateMultipliers()**  Convert any string-specified (formula-based) multipliers to numbers. Call this before using `CheckConstraints()` or `GenerateConstraints()`. At present, the code may pass only the dict for phase (atom/cell) parameters, but this could be expanded if needed.

**Map2Dict()**  To determine values for the parameters created in this module, one calls Map2Dict. This will not apply constraints.

**Dict2Map()**  To take values from the new independent parameters and constraints, one calls Dict2Map and set the parameter array, thus applying constraints.

**Dict2Deriv()**  Use Dict2Deriv to determine derivatives on independent parameters from those on dependent ones.

**ComputeDepESD()**  Use ComputeDepESD to compute uncertainties on dependent variables.

**VarRemapShow()**  To show a summary of the parameter remapping, one calls VarRemapShow.

### 1.8.4 Global Variables

**dependentParmList:** a list containing group of lists of parameters used in the group. Note that parameters listed in dependentParmList should not be refined as they will not affect the model

**indParmList:** a list containing groups of Independent parameters defined in each group. This contains both parameters used in parameter redefinitions as well as names of generated new parameters.

**arrayList:** a list containing group of relationship matrices to relate parameters in dependentParmList to those in indParmList. Unlikely to be used externally.

**invarrayList:** a list containing group of relationship matrices to relate parameters in indParmList to those in dependentParmList. Unlikely to be used externally.

**fixedVarList:** a list of parameters that have been ‘fixed’ by defining them as equal to a constant (::var: = 0). Note that the constant value is ignored at present. These parameters are later removed from varyList which prevents them from being refined. Unlikely to be used externally.

**fixedDict:** a dictionary containing the fixed values corresponding to parameter equations. The dict key is an ascii string, but the dict value is a float. Unlikely to be used externally.

**symGenList:** a list of boolean values that will be True to indicate that a constraint (only equivalences) is generated by symmetry (or Pawley overlap)

**problemVars:** a list containing parameters that show up in constraints producing errors

### 1.8.5 Routines/variables

Note that parameter names in GSAS-II are strings of form `<ph#>;<hst#>:<nam>` or `<ph#>::<nam>:<at#>`.

**GSASIImapvars: CheckConstraints** *(varyList, constrDict, fixedList)*

Takes a list of relationship entries comprising a group of constraints and checks for inconsistencies such as conflicts in parameter/variable definitions and or inconsistently varied parameters.

**Parameters**

- **varyList** *(list)* – a list of parameters names that will be varied
- **constrDict** *(dict)* – a list of dicts defining relationships/constraints (as created in `GSASIIstrIO.ProcessConstraints()` and documented in `GroupConstraints()`)
- **fixedList** *(list)* – a list of values specifying a fixed value for each dict in constrDict. Values are either strings that can be converted to floats or `None` if the constraint defines a new parameter rather than a constant.

1.8.  **GSASIImapvars: Parameter constraints** 127
Returns
two strings:
  • the first lists conflicts internal to the specified constraints
  • the second lists conflicts where the varyList specifies some parameters in a constraint, but not all

If there are no errors, both strings will be empty

**GSASIImapvars. CheckEquivalences**(constrDict, varyList, parmDict=None, SeqHist=None)
Process equivalence constraints, looking for conflicts such as where a parameter is used in both an equivalence and a constraint expression or where chaining is done (A->B and B->C). When called during refinements, parmDict is defined, and for sequential refinement SeqHist ia also defined.

  • parmDict is used to remove equivalences where a parameter is not present in a refinement
  • SeqHist is used to rename wild-card parameter names in sequential refinements to use the current histogram.

**GSASIImapvars. ComputeDepESD**(covMatrix, varyList, parmDict)
Compute uncertainties for dependent parameters from independent ones returns a dictionary containing the esd values for dependent parameters

**exception** **GSASIImapvars. ConstraintException**
Defines an Exception that is used when an exception is raised processing constraints

**GSASIImapvars. Dict2Deriv**(varyList, derivDict, dMdv)
Compute derivatives for Independent Parameters from the derivatives for the original parameters

**Parameters**
  • **varyList** (list) – a list of parameters names that will be varied
  • **derivDict** (dict) – a dict containing derivatives for parameter values keyed by the parameter names.
  • **dMdv** (list) – a Jacobian, as a list of np.array containing derivatives for dependent parameter computed from derivDict

**GSASIImapvars. Dict2Map**(parmDict, varyList)
Applies the constraints defined using **StoreEquivalence()**, **GroupConstraints()** and **GenerateConstraints()** by changing values in a dict containing the parameters. This should be done before the parameters are used for any computations

**Parameters**
  • **parmDict** (dict) – a dict containing parameter values keyed by the parameter names. This will contain updated values for both dependent and independent parameters after Dict2Map is called. It will also contain some unexpected entries of every constant value {'0':0.0} & {'1.0':1.0}, which do not cause any problems.
  • **varyList** (list) – a list of parameters names that will be varied

**GSASIImapvars. EvaluateMultipliers**(constList, *dicts)
Convert multipliers for constraints and equivalences that are specified as strings into values. The strings can specify values in the parameter dict as well as normal Python functions, such as “2*np.cos(0::Ax:2/2.)”

**Parameters**
  • **constList** (list) – a list of dicts containing constraint expressions
  • *dict1* – one or more dicts containing GSAS-II parameters and their values can be specified
The `GenerateConstraints` function takes a list of relationship entries comprising a group of constraints and builds the relationship lists and their inverse and stores them in global parameters. It also checks for internal conflicts or inconsistencies in parameter/variable definitions.

### Parameters

- `varyList` (list) – a list of parameters names (strings of form `<ph>:<hst>:<nam>` that will be varied. Note that this is changed here.
- `constrDict` (dict) – a list of dicts defining relationships/constraints (as defined in `GroupConstraints()`)
- `fixedList` (list) – a list of values specifying a fixed value for each dict in `constrDict`. Values are either strings that can be converted to floats, float values or None if the constraint defines a new parameter.
- `parmDict` (dict) – a dict containing all parameters defined in current refinement.
- `SeqHist` (int) – number of current histogram, when used in a sequential refinement. None (default) otherwise. Wildcard parameter names are set to the current histogram, when found if not None.

### Functions

- **`GetDependentVars()`**
  Return a list of dependent variables: e.g. parameters that are constrained in terms of other parameters

- **`GetIndependentVars()`**
  Return a list of independent variables: e.g. parameters that are slaved to other parameters by constraints

- **`GetSymEquiv()`**
  Return the automatically generated (equivalence) relationships.

- **`GramSchmidtOrtho(a, nkeep=0)`**
  Use the Gram-Schmidt process (http://en.wikipedia.org/wiki/Gram-Schmidt) to find orthonormal unit vectors relative to first row.

  - **input:** arrayin: a 2-D non-singular square array
  - **returns:** a orthonormal set of unit vectors as a square array

- **`GroupConstraints(constrDict)`**
  divide the constraints into groups that share no parameters.

  - **Parameters**
    - `constrDict` (dict) – a list of dicts defining relationships/constraints

  ```python
  constrDict = [{{<constr1>}}, {{<constr2>}}, ...]
  ```

  where `{<constr1>}` is `{‘var1’: mult1, ‘var2’: mult2, ... }

  - **Returns**
    - two lists of lists:
• a list of grouped contraints where each constraint grouped contains a list of indices for constraint constrDict entries
• a list containing lists of parameter names contained in each group

GSASIImapvars.InitVars()
Initializes all constraint information

GSASIImapvars.Map2Dict(parmDict, varyList)
Create (or update) the Independent Parameters from the original set of Parameters
Removes dependent variables from the varyList

This should be done once, after the constraints have been defined using StoreEquivalence(), GroupConstraints() and GenerateConstraints() and before any parameter refinement is done. This completes the parameter dictionary by defining independent parameters and it satisfies the constraint equations

Parameters

• parmDict (dict) – a dict containing parameter values keyed by the parameter names. This will contain updated values for both dependent and independent parameters after Dict2Map is called. It will also contain some unexpected entries of every constant value {‘0’:0.0} & {‘1.0’:1.0}, which do not cause any problems.
• varyList (list) – a list of parameters names that will be varied

GSASIImapvars.MoveConfEquiv(constrDict, fixedList)
Address conflicts in Equivalence constraints by creating an constraint equation that has the same action as the equivalence and removing the Equivalence

GSASIImapvars.PrintIndependentVars(parmDict, varyList, sigDict, PrintAll=False, pFile=None)
Print the values and uncertainties on the independent parameters

GSASIImapvars.StoreEquivalence(independentVar, dependentList, symGen=True)
Takes a list of dependent parameter(s) and stores their relationship to a single independent parameter (independentVar).

Called with user-supplied constraints by GSASIIstrIO.ProcessConstraints, with Pawley constraints from :func:`GSASIIstrIO.GetPawleyConstr()`, with Unit Cell constraints from GSASIIstrIO.cellVary() with symmetry-generated atom constraints from GSASIIstrIO.GetPhaseData()

Parameters

• independentVar (str) – name of master parameter that will be used to determine the value to set the dependent variables
• dependentList (list) – a list of parameters that will set from independentVar. Each item in the list can be a string with the parameter name or a tuple containing a name and multiplier: [‘::parm1’, (‘::parm2’, .5),

GSASIImapvars.VarKeys(constr)
Finds the keys in a constraint that represent parameters e.g. eliminates any that start with ‘_’

Parameters constr (dict) – a single constraint entry of form:

```
{‘var1’: mult1, ‘var2’: mult2,... ‘_notVar’: val,...}
```

(see GroupConstraints())

Returns a list of keys where any keys beginning with ‘_’ are removed.
GSASIImapvars.VarRemapShow(varyList, inputOnly=False)
List out the saved relationships. This should be done after the constraints have been defined using
StoreEquivalence(), GroupConstraints() and GenerateConstraints().

Returns a string containing the details of the contraint relationships

GSASIImapvars.arrayList = []
a list of of relationship matrices that map model parameters in each constraint group (in
dependentParmList) to generated (New Var) parameters.

GSASIImapvars.consNum = 0
The number to be assigned to the next constraint to be created

GSASIImapvars.dependentParmList = []
a list of lists where each item contains a list of parameters in each constraint group. note that parameters listed
in dependentParmList should not be refined directly.

GSASIImapvars.dependentVars = []
A list of dependent variables, taken from (dependentParmList).

GSASIImapvars.fixedDict = {}
A dict lookup-table containing the fixed values corresponding to defined parameter equations. Note the key is
the original ascii string and the value in the dict is a float.

GSASIImapvars.fixedVarList = []
List of parameters that should not be refined.

GSASIImapvars.genVarLookup = {}
provides a list of parameters that are related to each generated parameter

GSASIImapvars.indParmList = []
a list of lists where each item contains a list for each constraint group with fixed values for constraint equations
and names of generated (New Var) parameters.

GSASIImapvars.independentVars = []
A list of dependent variables, taken from (indParmList).

GSASIImapvars.inarrayList = []
a list of of inverse-relationship matrices that map constrained values and generated (New Var) parameters (in
indParmList) to model parameters (in dependentParmList).

GSASIImapvars.paramPrefix = '::constr'
A prefix for generated parameter names

GSASIImapvars.problemVars = []
a list of parameters causing errors

GSASIImapvars.symGenList = []
A list of flags that if True indicates a constraint was generated by symmetry

1.9 GSASIIimage: Image calc module

Ellipse fitting & image integration

GSASIIimage.EdgeFinder(image, data)
this makes list of all x,y where I>edgeMin suitable for an ellipse search? Not currently used but might be useful
in future?

GSASIIimage.Fill2ThetaAzimuthMap(masks, TA, tam, image)
Needs a doc string
GSASIIimage.FitDetector(rings, varyList, parmDict, Print=True, covar=False)
Fit detector calibration parameters

Parameters

- **rings** (*np.array*) – vector of ring positions
- **varyList** (*list*) – calibration parameters to be refined
- **parmDict** (*dict*) – all calibration parameters
- **Print** (*bool*) – set to True (default) to print the results
- **covar** (*bool*) – set to True to return the covariance matrix (default is False)

Returns: [chisq,vals,sigList] unless covar is True, then [chisq,vals,sigList,coVarMatrix] is returned

GSASIIimage.FitMultiDist(rings, varyList, parmDict, Print=True, covar=False)
Fit detector calibration parameters with multi-distance data

Parameters

- **rings** (*np.array*) – vector of ring positions (x,y,dist,d-space)
- **varyList** (*list*) – calibration parameters to be refined
- **parmDict** (*dict*) – calibration parameters
- **Print** (*bool*) – set to True (default) to print the results
- **covar** (*bool*) – set to True to return the covariance matrix (default is False)

Returns: [chisq,vals,sigDict] unless covar is True, then [chisq,vals,sigDict,coVarMatrix] is returned

GSASIIimage.FitStrSta(Image, StrSta, Controls)
Needs a doc string

GSASIIimage.FitStrain(rings, ρ0, dset, wave, phi, StaType)
Needs a doc string

GSASIIimage.GetAzm(x, y, data)
Give azimuth value for detector x,y position; calibration info in data

GSASIIimage.GetDetXYfromThAzm(Th, Azm, data)
Computes a detector position from a 2theta angle and an azimuthal angle (both in degrees) - apparently not used!

GSASIIimage.GetDetectorXY(dsp, azm, data)
Get detector x,y position from d-spacing (dsp), azimuth (azm,deg) & image controls dictionary (data) it seems to be only used in plotting

GSASIIimage.GetDsp(x, y, data)
Give d-spacing value for detector x,y position; calibration info in data

GSASIIimage.GetEllipse(dsp, data)
uses Dandelin spheres to find ellipse or hyperbola parameters from detector geometry as given in image controls dictionary (data) and a d-spacing (dsp)

GSASIIimage.GetEllipse2(tth, dxy, dist, cent, tilt, phi)
uses Dandelin spheres to find ellipse or hyperbola parameters from detector geometry on output radii[0] (b-minor axis) set < 0. for hyperbola

GSASIIimage.GetTth(x, y, data)
Give 2-theta value for detector x,y position; calibration info in data

GSASIIimage.GetTthAzm(x, y, data)
Give 2-theta, azimuth values for detector x,y position; calibration info in data
GSASII image. **GetTthAzmDsp** *(x, y, data)*
Computes a 2theta, etc. from a detector position and calibration constants - checked OK for ellipses & hyperbola.

**Returns** np.array(tth, azm, G, dsp) where tth is 2theta, azm is the azimuthal angle, G is ? and dsp is the d-space

GSASII image. **GetTthAzmG** *(x, y, data)*
Give 2-theta, azimuth & geometric corr. values for detector x,y position; calibration info in data - only used in integration

GSASII image. **ImageCalibrate** *(G2frame, data)*
Called to perform an initial image calibration after points have been selected for the inner ring.

GSASII image. **ImageCompress** *(image, scale)*
Reduces size of image by selecting every n’th point param: image array: original image param: scale int: intervsl between selected points returns: array: reduced size image

GSASII image. **ImageIntegrate** *(image, data, masks, blkSize=128, returnN=False, useTA=None, useMask=None)*
Integrate an image; called from OnIntegrateAll and OnIntegrate in G2imgGUI

GSASII image. **ImageLocalMax** *(image, w, Xpix, Ypix)*
Needs a doc string

GSASII image. **ImageRecalibrate** *(G2frame, ImageZ, data, masks, getRingsOnly=False)*
Called to repeat the calibration on an image, usually called after calibration is done initially to improve the fit.

**Parameters**
- **G2frame** – The top-level GSAS-II frame or None, to skip plotting
- **ImageZ np.Array** – the image to calibrate
- **data dict** – the Controls dict for the image
- **masks dict** – a dict with masks

**Returns** a list containing vals, varyList, sigList, parmDict, covar or rings (with an array of x, y, and d-space values) if getRingsOnly is True or an empty list, in case of an error

GSASII image. **Make2ThetaAzimuthMap** *(data, iLim, jLim)*
Needs a doc string

GSASII image. **calcFij** *(omg, phi, azm, th)*
Uses parameters as defined by Bob He & Kingsley Smith, Adv. in X-Ray Anal. 41, 501 (1997)

**Parameters**
- **omg** – his omega = sample omega rotation; 0 when incident beam ll sample surface, 90 when perp. to sample surface
- **phi** – his phi = sample phi rotation; usually = 0, axis rotates with omg.
- **azm** – his chi = azimuth around incident beam
- **th** – his theta = theta

GSASII image. **checkEllipse** *(Zsum, distSum, xSum, ySum, dist, x, y)*
Needs a doc string

GSASII image. **makeMat** *(Angle, Axis)*
Make rotation matrix from Angle and Axis

**Parameters**
- **Angle float** – in degrees
• **Axis** *(int)* – 0 for rotation about x, 1 for about y, etc.

```python
GSASIIimage.makeRing(dsp, ellipse, pix, reject, scalex, scaley, image, mul=1)
```

Needs a doc string

```python
GSASIIimage.peneCorr(tth, dep, dist=0.0, azm=0.0)
```

Needs a doc string

```python
GSASIIimage.pointInPolygon(pXY, xy)
```

Needs a doc string

### 1.10 GSASIImath: computation module

Routines for least-squares minimization and other stuff

```python
GSASIImath.AV2Q(A, V)
```

convert angle (radians) & vector to quaternion \( q=r+ai+bj+ck \)

```python
GSASIImath.AVdeg2Q(A, V)
```

convert angle (degrees) & vector to quaternion \( q=r+ai+bj+ck \)

```python
GSASIImath.ApplyModulation(data, tau)
```

Applies modulation to drawing atom positions & Uij's for given tau

```python
GSASIImath.ApplySeqData(data, seqData)
```

Applies result from seq. refinement to drawing atom positions & Uij's

```python
GSASIImath.AtomTLS2UIJ(atomData, atPtrs, Amat, rbObj)
```

default doc string

```python
Parameters name(type) – description

Returns type name: description
```

```python
GSASIImath.BessIn(nmax, x)
```

compute modified Bessel function \( I(n,x) \) from scipy routines & recurrence relation returns sequence of \( I(n,x) \) for \( n \) in range \([-nmax...0...nmax]\)

```python
Parameters

• **nmax** *(integer)* – maximal order for \( I(n,x) \)
• **x** *(float)* – argument for \( I(n,x) \)

Returns numpy array \([I(-nmax,x)...I(0,x)...I(nmax,x)]\)
```

```python
GSASIImath.BessJn(nmax, x)
```

compute Bessel function \( J(n,x) \) from scipy routine & recurrence relation returns sequence of \( J(n,x) \) for \( n \) in range \([-nmax...0...nmax]\)

```python
Parameters

• **nmax** *(integer)* – maximal order for \( J(n,x) \)
• **x** *(float)* – argument for \( J(n,x) \)

Returns numpy array \([J(-nmax,x)...J(0,x)...J(nmax,x)]\)
```

```python
GSASIImath.ChargeFlip(data, reflDict, pgbar)
```

default doc string

```python
Parameters name(type) – description

Returns type name: description
```
GSASIImath.Den2Vol (Elements, density)
converts density to molecular volume

Parameters

- **Elements (dict)** – elements in molecular formula; each must contain
  **Num**: number of atoms in formula
  **Mass**: at. wt.

- **density (float)** – material density in gm/cm^3

Returns float volume: molecular volume in A^3

GSASIImath.El2EstVol (Elements)
Estimate volume from molecular formula; assumes atom volume = 10A^3

Parameters **Elements (dict)** – elements in molecular formula; each must contain
**Num**: number of atoms in formula

Returns float volume: estimate of molecular volume in A^3

GSASIImath.El2Mass (Elements)
compute molecular weight from Elements

Parameters **Elements (dict)** – elements in molecular formula; each must contain
**Num**: number of atoms in formula

Returns float mass: molecular weight.

GSASIImath.FillAtomLookUp (atomData, indx)
create a dictionary of atom indexes with atom IDs as keys

Parameters **atomData (list)** – Atom table to be used

Returns dict atomLookUp: dictionary of atom indexes with atom IDs as keys

GSASIImath.FindAtomIndexByIDs (atomData, loc, IDs, Draw=True)
finds the set of atom array indices for a list of atom IDs. Will search either the Atom table or the drawAtom table.

Parameters

- **atomData (list)** – Atom or drawAtom table containing coordinates, etc.

- **loc (int)** – location of atom id in atomData record

- **IDs (list)** – atom IDs to be found

- **Draw (bool)** – True if drawAtom table to be searched; False if Atom table is searched

Returns list indx: atom (or drawAtom) indices

GSASIImath.Fourier4DMap (data, reflDict)
default doc string

Parameters **name (type)** – description

Returns type name: description

GSASIImath.FourierMap (data, reflDict)
default doc string

Parameters **name (type)** – description

Returns type name: description

GSASIImath.GetAngleSig (Oatoms, Atoms, Amat, SGData, covData=[])
default doc string
<table>
<thead>
<tr>
<th>Function</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSASIImath.GetAtomCoordsByID(pId, parmDict, AtLookup, indx)</td>
<td>default doc string</td>
</tr>
<tr>
<td>GSASIImath.GetAtomFracByID(pId, parmDict, AtLookup, indx)</td>
<td>default doc string</td>
</tr>
<tr>
<td>GSASIImath.GetAtomItemsById(atomData, atomLookUp, IdList, itemLoc, numItems=1)</td>
<td>gets atom parameters for atoms using atom IDs</td>
</tr>
<tr>
<td>GSASIImath.GetAtomsById(atomData, atomLookUp, IdList)</td>
<td>gets a list of atoms from Atom table that match a set of atom IDs</td>
</tr>
</tbody>
</table>

**Parameters**
- **atomData (list)** – Atom table to be used
- **atomLookUp (dict)** – dictionary of atom indexes with atom IDs as keys
- **IdList (list)** – atom IDs to be found
- **itemLoc (int)** – pointer to desired 1st item in an atom table entry
- **numItems (int)** – number of items to be retrieved

**Returns** type name: description

<table>
<thead>
<tr>
<th>Function</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSASIImath.GetDATSig(Oatoms, Atoms, Amat, SGData, covData={})</td>
<td>default doc string</td>
</tr>
<tr>
<td>GSASIImath.GetDistSig(Oatoms, Atoms, Amat, SGData, covData={})</td>
<td>default doc string</td>
</tr>
<tr>
<td>GSASIImath.GetSHCoeff(pId, parmDict, SHkeys)</td>
<td>default doc string</td>
</tr>
</tbody>
</table>

**Parameters**
- **name (type)** – description

**Returns** type name: description
GSASIImath.GetTorsionSig(Oatoms, Atoms, Amat, SGData, covData={})

default doc string

Parameters name(type) – description

Returns type name: description

GSASIImath.GetXYZDist(xyz, XYZ, Amat)

gets distance from position xyz to all XYZ, xyz & XYZ are np.array and are in crystal coordinates; Amat is crystal to Cart matrix

Parameters name(type) – description

Returns type name: description

GSASIImath.HessianLSQ(func, x0, Hess, args=(), ftol=1.49012e-08, xtol=1e-06, maxcyc=0, lamda=-3, Print=False, refPlotUpdate=None)

Minimize the sum of squares of a function \( f \) evaluated on a series of values \( y \): \[ \sum_{y=0}^{N_{obs}} y = 0 \] \( f(y, \text{args}) \) where \( x = \text{argmin}(\sum_{y=0}^{N_{obs}} (\text{func}(y))^2, \text{axis} = 0) \)

Parameters

- func (function) – callable method or function should take at least one (possibly length N vector) argument and returns M floating point numbers.
- x0 (np.ndarray) – The starting estimate for the minimization of length N
- Hess (function) – callable method or function A required function or method to compute the weighted vector and Hessian for func. It must be a symmetric NxN array
- args (tuple) – Any extra arguments to func are placed in this tuple.
- ftol (float) – Relative error desired in the sum of squares.
- xtol (float) – Relative tolerance of zeros in the SVD solution in nl.pinv.
- maxcyc (int) – The maximum number of cycles of refinement to execute, if -1 refine until other limits are met (ftol, xtol)
- lamda (int) – initial Marquardt lambda=10**lamda
- Print (bool) – True for printing results (residuals & times) by cycle

Returns

\((x, \text{cov}_x, \text{infodict})\) where

- \(x\) : ndarray The solution (or the result of the last iteration for an unsuccessful call).
- \(\text{cov}_x\) : ndarray Uses the fjac and ipvt optional outputs to construct an estimate of the jacobian around the solution. None if a singular matrix encountered (indicates very flat curvature in some direction). This matrix must be multiplied by the residual standard deviation to get the covariance of the parameter estimates – see curve_fit.
- \(\text{infodict}\) : dict a dictionary of optional outputs with the keys:
  - 'fvec' : the function evaluated at the output
  - 'num cyc':
  - 'nfev':
  - 'lamMax':
  - 'psing':

1.10. GSASIImath: computation module
GSAS-II Developers Documentation, Release version 4249

- 'SVD0':

```python
GSASIImath.HessianSVD(func, x0, Hess, args=(), ftol=1.49012e-08, xtol=1e-06, maxcyc=0, lamda=-3,
Print=False, refPlotUpdate=None)
```

Minimize the sum of squares of a function \( f \) evaluated on a series of values \( y \): \( \sum_{y=0}^{N_{obs}} f(y, \text{args}) \) where \( x = \text{argmin}(\sum_{y=0}^{N_{obs}} (func(y)^2, axis = 0)) \)

**Parameters**

- **func** *(function)* – callable method or function should take at least one (possibly length N vector) argument and returns M floating point numbers.
- **x0** *(np.ndarray)* – The starting estimate for the minimization of length N
- **Hess** *(function)* – callable method or function A required function or method to compute the weighted vector and Hessian for func. It must be a symmetric N x N array
- **args** *(tuple)* – Any extra arguments to func are placed in this tuple.
- **ftol** *(float)* – Relative error desired in the sum of squares.
- **xtol** *(float)* – Relative tolerance of zeros in the SVD solution in nl.pinv.
- **maxcyc** *(int)* – The maximum number of cycles of refinement to execute, if -1 refine until other limits are met (ftol, xtol)
- **Print** *(bool)* – True for printing results (residuals & times) by cycle

**Returns**

\( (x, \text{cov}_x, \text{infodict}) \) where

- **x** : ndarry The solution (or the result of the last iteration for an unsuccessful call).
- **cov_x** : ndarry Uses the fjac and ipvt optional outputs to construct an estimate of the Jacobian around the solution. \text{None} if a singular matrix encountered (indicates very flat curvature in some direction). This matrix must be multiplied by the residual standard deviation to get the covariance of the parameter estimates – see curve_fit.
- **infodict** : dict a dictionary of optional outputs with the keys:
  - 'fvec': the function evaluated at the output
  - 'num cyc':
  - 'nfev':
  - 'lamMax':0.
  - 'psing':
  - 'SVD0':

GSASIImath.MagMod(glTau, XYZ, modQ, MSSdata, SGData, SSGData)

this needs to make magnetic moment modulations & magnitudes as fn of gTau points

GSASIImath.Modulation(H, HP, nWaves, Fmod, Xmod, Umod, glTau, glWt)

H: array nRefBlk x ops X hklt HP: array nRefBlk x ops X hklt proj to hkl nWaves: list number of waves for frac, pos, uij & mag Fmod: array 2 x atoms x waves (sin,cos terms) Xmod: array atoms X 3 X ngl Umod: array atoms x 3 x 3 x ngl glTau, glWt: arrays Gauss-Lorentzian pos & wts

GSASIImath.ModulationDeriv(H, HP, Hij, nWaves, waveShapes, Fmod, Xmod, UmodAB, SCtauF, SCtauX, SCtauU, glTau, glWt)

Compute Fourier modulation derivatives H: array ops X hklt proj to hkl HP: array ops X hklt proj to hkl Hij: array 2pi^2[a^2h^2 b^2k^2 c^2l^2 a*b*hk a*c*hl b*c*kl] of projected hklm to hkl space
GSASIImath.\texttt{ModulationTw}(H, HP, nWaves, Fmod, Xmod, Umod, glTau, glWt)

H: array nRefBlk x tw x ops X hkl HP: array nRefBlk x tw x ops X hkl proj to hkl Fmod: array 2 x atoms x waves (sin,cos terms) Xmod: array atoms X ngl X 3 Umod: array atoms x ngl x 3x3 glTau,glWt: arrays

GSASIImath.\texttt{NCScattDen}(\texttt{Elements}, vol, wave=0.0)

Estimate neutron scattering density from molecular formula & volume; ignores valence, but includes anomalous effects

\textbf{Parameters}

- **\texttt{Elements} (dict)** – elements in molecular formula; each element must contain Num: number of atoms in formula Z: atomic number
- **\texttt{vol (float)}** – molecular volume in A^3
- **\texttt{wave (float)}** – optional wavelength in A

\textbf{Returns}

- float rho: scattering density in 10^10cm^{-2}; if wave > 0 the includes \( f' \) contribution
- float mu: if wave>0 absorption coeff in cm^{-1}; otherwise 0
- float fpp: if wave>0 \( f'' \) in 10^10cm^{-2}; otherwise 0

GSASIImath.\texttt{OmitMap}(data, reflDict, pgbar=None)

\texttt{default doc string}

\textbf{Parameters}

- name (type) – description

\textbf{Returns}

- type name: description

GSASIImath.\texttt{PeaksEquiv}(data, Ind)

Find the equivalent map peaks for those selected. Works on the contents of data[‘Map Peaks’].

\textbf{Parameters}

- **data** – the phase data structure
- **Ind (list)** – list of selected peak indices

\textbf{Returns}

- augmented list of peaks including those related by symmetry to the ones in Ind

GSASIImath.\texttt{PeaksUnique}(data, Ind)

Finds the symmetry unique set of peaks from those selected. Works on the contents of data[‘Map Peaks’].

\textbf{Parameters}

- **data** – the phase data structure
- **Ind (list)** – list of selected peak indices

\textbf{Returns}

- the list of symmetry unique peaks from among those given in Ind

GSASIImath.\texttt{Q2AV}(Q)

\begin{itemize}
  \item convert quaternion to angle (radians 0-2\pi) & normalized vector \( q = r + ai + bj + ck \)
\end{itemize}

GSASIImath.\texttt{Q2AVdeg}(Q)

\begin{itemize}
  \item convert quaternion to angle (degrees 0-360) & normalized vector \( q = r + ai + bj + ck \)
\end{itemize}

GSASIImath.\texttt{Q2Mat}(Q)

\begin{itemize}
  \item make rotation matrix from quaternion \( q = r + ai + bj + ck \)
\end{itemize}

GSASIImath.\texttt{RotateRBXYZ}(Bmat, Cart, oriQ)

\begin{itemize}
  \item rotate & transform cartesian coordinates to crystallographic ones no translation applied. To be used for numerical derivatives
\end{itemize}

\textbf{Parameters}

- name (type) – description

1.10. \textit{GSASIImath: computation module}
Returns type name: description

GSASIImath.**SSChargeFlip** *(data, reflDict, pgbar)*

default doc string

Parameters name *(type)* – description

Returns type name: description

GSASIImath.**SearchMap** *(generalData, drawingData, Neg=False)*

Does a search of a density map for peaks meeting the criterion of peak height is greater than mapData['cutOff']/100 of mapData['rhoMax'] where mapData is data['General']['mapData']; the map is also in mapData.

Parameters

- **generalData** – the phase data structure; includes the map
- **drawingData** – the drawing data structure
- **Neg** – if True then search for negative peaks (i.e. H-atoms & neutron data)

Returns

(peaks,mags,dzeros) where

- peaks: ndarray x,y,z positions of the peaks found in the map
- mags: ndarray the magnitudes of the peaks
- dzeros: ndarray the distance of the peaks from the unit cell origin
- dcent: ndarray the distance of the peaks from the unit cell center

GSASIImath.**SetMolCent** *(model, RBData)*

default doc string

Parameters name *(type)* – description

Returns type name: description

GSASIImath.**TLS2Uij** *(xyz, g, Amat, rbObj)*

default doc string

Parameters name *(type)* – description

Returns type name: description

GSASIImath.**UpdateMCSAxyz** *(Bmat, MCSA)*

default doc string

Parameters name *(type)* – description

Returns type name: description

GSASIImath.**UpdateRBUIJ** *(Bmat, Cart, RBObj)*

default doc string

Parameters name *(type)* – description

Returns type name: description

GSASIImath.**UpdateRBXYZ** *(Bmat, RBObj, RBData, RBType)*

default doc string

Parameters name *(type)* – description

Returns type name: description
GSASIImath.ValEsd (value, esd=0, nTZ=False)
Format a floating point number with a given level of precision or with in crystallographic format with a “esd”, as value(esd). If esd is negative the number is formatted with the level of significant figures appropriate if abs(esd) were the esd, but the esd is not included. if the esd is zero, approximately 6 significant figures are printed. nTZ=True causes “extra” zeros to be removed after the decimal place. for example:

- “1.235(3)” for value=1.2346 & esd=0.003
- “1.235(3)e4” for value=12346. & esd=30
- “1.235(3)e6” for value=0.12346e7 & esd=3000
- “1.235” for value=1.2346 & esd=-0.003
- “1.240” for value=1.2395 & esd=-0.003
- “1.24” for value=1.2395 & esd=-0.003 with nTZ=True
- “1.23460” for value=1.2346 & esd=0.0

Parameters
- value (float) – number to be formatted
- esd (float) – uncertainty or if esd < 0, specifies level of precision to be shown e.g. esd=-0.01 gives 2 places beyond decimal
- nTZ (bool) – True to remove trailing zeros (default is False)

Returns value(esd) or value as a string

GSASIImath.Vol2Den (Elements, volume)
converts volume to density

Parameters
- Elements (dict) – elements in molecular formula; each must contain Num: number of atoms in formula Mass: at. wt.
- volume (float) – molecular volume in A^3

Returns float density: material density in gm/cm^3

GSASIImath.XScattDen (Elements, vol, wave=0.0)
Estimate X-ray scattering density from molecular formula & volume; ignores valence, but includes anomalous effects

Parameters
- Elements (dict) – elements in molecular formula; each element must contain Num: number of atoms in formula Z: atomic number
- vol (float) – molecular volume in A^3
- wave (float) – optional wavelength in A

Returns float rho: scattering density in 10^10cm^-2; if wave > 0 the includes f’ contribution
Returns float mu: if wave>0 absorption coeff in cm^-1; otherwise 0
Returns float fpp: if wave>0 f” in 10^10cm^-2; otherwise 0

GSASIImath.adjHKLmax (SGData, Hmax)
default doc string

Parameters name (type) – description
Returns  type name: description

GSASIIImath.anneal(func, x0, args=(), schedule='fast', T0=None, Tf=1e-12, maxeval=None, maxaccept=None, maxiter=400, feps=1e-06, quench=1.0, c=1.0, lower=-100, upper=100, dwell=50, slope=0.9, ranStart=False, ranRange=0.1, autoRan=False, dlg=None)

Minimize a function using simulated annealing.

Schedule is a schedule class implementing the annealing schedule. Available ones are ‘fast’, ‘cauchy’, ‘boltzmann’

Parameters

- **func** (*callable*) – f(x, *args) Function to be optimized.
- **x0** (*ndarray*) – Initial guess.
- **args** (*tuple*) – Extra parameters to **func**.
- **schedule** (*base_schedule*) – Annealing schedule to use (a class).
- **T0** (*float*) – Initial Temperature (estimated as 1.2 times the largest cost-function deviation over random points in the range).
- **Tf** (*float*) – Final goal temperature.
- **maxeval** (*int*) – Maximum function evaluations.
- **maxaccept** (*int*) – Maximum changes to accept.
- **maxiter** (*int*) – Maximum cooling iterations.
- **feps** (*float*) – Stopping relative error tolerance for the function value in last four coolings.
- **quench, c** (*float*) – Parameters to alter fast_sa schedule.
- **lower, upper** (*float/ndarray*) – Lower and upper bounds on x.
- **dwell** (*int*) – The number of times to search the space at each temperature.
- **slope** (*float*) – Parameter for log schedule
- **ranStart=False** (*bool*) – True for set 10% of ranges about x

Returns

(xmin, Jmin, T, feval, iters, accept, retval) where

- **xmin** (*ndarray*): Point giving smallest value found.
- **Jmin** (*float*): Minimum value of function found.
- **T** (*float*): Final temperature.
- **feval** (*int*): Number of function evaluations.
- **iters** (*int*): Number of cooling iterations.
- **accept** (*int*): Number of tests accepted.
- **retval** (*int*): Flag indicating stopping condition:
  - 0: Points no longer changing
  - 1: Cooled to final temperature
  - 2: Maximum function evaluations
  - 3: Maximum cooling iterations reached
Notes: Simulated annealing is a random algorithm which uses no derivative information from the function being optimized. In practice it has been more useful in discrete optimization than continuous optimization, as there are usually better algorithms for continuous optimization problems.

Some experimentation by trying the difference temperature schedules and altering their parameters is likely required to obtain good performance.

The randomness in the algorithm comes from random sampling in numpy. To obtain the same results you can call numpy.random.seed with the same seed immediately before calling scipy.optimize.anneal.

We give a brief description of how the three temperature schedules generate new points and vary their temperature. Temperatures are only updated with iterations in the outer loop. The inner loop is over range(dwell), and new points are generated for every iteration in the inner loop. (Though whether the proposed new points are accepted is probabilistic.)

For readability, let d denote the dimension of the inputs to func. Also, let x_old denote the previous state, and k denote the iteration number of the outer loop. All other variables not defined below are input variables to scipy.optimize.anneal itself.

In the ‘fast’ schedule the updates are

\[
\begin{align*}
    u & \sim \text{Uniform}(0, 1, \text{size}=d) \\
    y & = \text{sgn}(u - 0.5) \times T \times ((1 + 1/T) \times \text{abs}(2u - 1) - 1.0) \\
    xc & = y \times \text{upper} - \text{lower} \\
    x_{\text{new}} & = x_{\text{old}} + xc \\
    T_{\text{new}} & = T_0 \times \exp(-c \times k^{\text{quench}})
\end{align*}
\]

GSASIImath.calcRamaEnergy(phi, psi, Coeff=[])
Computes pseudo potential energy from a pair of torsion angles and a numerical description of the potential energy surface. Used to create penalty function in LS refinement: 
\[
    \text{Eval}(\phi, \psi) = C[0] \times \exp(-V/1000)
\]
where 
\[
    V = -C[3] \times (\phi - C[1])^2 - C[4] \times (\psi - C[2])^2 - 2 \times (\phi - C[1]) \times (\psi - C[2])
\]

Parameters

• phi (float) – first torsion angle (\(\phi\))
• psi (float) – second torsion angle (\(\psi\))
• Coeff (list) – pseudo potential coefficients

Returns list (sum,Eval): pseudo-potential difference from minimum & value; sum is used for penalty function.

GSASIImath.calcTorsionEnergy(TOR, Coeff=[])
default doc string

Parameters name (type) – description

Returns type name: description

GSASIImath.findOffset(SGData, A, Fhkl)
default doc string

Parameters name (type) – description

Returns type name: description

GSASIImath.findSSOffset(SGData, SSGData, A, Fhklm)
default doc string
Parameters name (type) – description

Returns type name: description

GSASIImath.getAngSig(VA, VB, Amat, SGData, covData={})
default doc string

Parameters name (type) – description

Returns type name: description

GSASIImath.getAtomXYZ(atoms, cx)
default doc string

Parameters name (type) – description

Returns type name: description

GSASIImath.getCWgam(ins, pos)
get CW peak profile gamma

Parameters

• ins (dict) – instrument parameters with at least ‘X’, ‘Y’ & ‘Z’ as values only
• pos (float) – 2-theta of peak

Returns float getCWgam: peak gamma

GSASIImath.getCWgamDeriv(pos)
get derivatives of CW peak profile gamma wrt X, Y & Z

Parameters pos (float) – 2-theta of peak

Returns list getCWgamDeriv: d(gam)/dX & d(gam)/dY

GSASIImath.getCWSig(ins, pos)
get CW peak profile sigma^2

Parameters

• ins (dict) – instrument parameters with at least ‘U’, ‘V’, & ‘W’ as values only
• pos (float) – 2-theta of peak

Returns float getCWSig: peak sigma^2

GSASIImath.getCWSigDeriv(pos)
get derivatives of CW peak profile sigma^2 wrt U, V & W

Parameters pos (float) – 2-theta of peak

Returns list getCWSigDeriv: d(sig^2)/dU, d(sig)/dV & d(sig)/dW

GSASIImath.getDensity(generalData)
calculate crystal structure density

Parameters generalData (dict) – The General dictionary in Phase

Returns float density: crystal density in gm/cm^3

GSASIImath.getDistDerv(Oxyz, Txyz, Amat, Tunit, Top, SGData)
default doc string

Parameters name (type) – description

Returns type name: description
GSASIImath.getMass (generalData)
Computes mass of unit cell contents

Parameters generalData (dict) – The General dictionary in Phase

Returns float mass: Crystal unit cell mass in AMU.

GSASIImath.getMeanWave (Parms)
returns mean wavelength from Instrument parameters dictionary

ParametersParms (dict) – Instrument parameters; must contain: Lam: single wavelength or Lam1, Lam2: Ka1, Ka2 radiation wavelength I(L2)/I(L1): Ka2/Ka1 ratio

Returns float wave: mean wavelength

GSASIImath.getRBTransMat (X, Y)
Get transformation for Cartesian axes given 2 vectors X will be parallel to new X-axis; X cross Y will be new Z-axis & (X cross Y) cross Y will be new Y-axis Useful for rigid body axes definition

Parameters

• X (array) – normalized vector
• Y (array) – normalized vector

Returns array M: transformation matrix
use as XYZ’ = np.inner(M,XYZ) where XYZ are Cartesian

GSASIImath.getRamaDeriv (XYZ, Amat, Coeff)
Computes numerical derivatives of torsion angle pair pseudo potential with respect of crystallographic atom coordinates of the 5 atom sequence

Parameters

• XYZ (nparray) – crystallographic coordinates of 5 atoms
• Amat (nparray) – crystal to cartesian transformation matrix
• Coeff (list) – pseudo potential coefficients

Returns list (deriv) derivatives of pseudopotential with respect to 5 atom crystallographic xyz coordinates.

GSASIImath.getRestAngle (XYZ, Amat)
default doc string

Parameters name (type) – description

Returns type name: description

GSASIImath.getRestChiral (XYZ, Amat)
default doc string

Parameters name (type) – description

Returns type name: description

GSASIImath.getRestDeriv (Func, XYZ, Amat, ops, SGData)
default doc string

Parameters name (type) – description

Returns type name: description

GSASIImath.getRestDist (XYZ, Amat)
default doc string

1.10. GSASIImath: computation module
Parameters name (type) – description

Returns type name: description

GSASIImath.getRestPlane (XYZ, Amat)
default doc string

Parameters name (type) – description

Returns type name: description

GSASIImath.getRestPolefig (ODFl, SamSym, Grid)
default doc string

Parameters name (type) – description

Returns type name: description

GSASIImath.getRestPolefigDerv (HKL, Grid, SHCoeff)
default doc string

Parameters name (type) – description

Returns type name: description

GSASIImath.getRestRama (XYZ, Amat)
Computes a pair of torsion angles in a 5 atom string

Parameters

• XYZ (nparray) – crystallographic coordinates of 5 atoms
• Amat (nparray) – crystal to cartesian transformation matrix

Returns list (phi, psi) two torsion angles in degrees

GSASIImath.getRestTorsion (XYZ, Amat)
default doc string

Parameters name (type) – description

Returns type name: description

GSASIImath.getRho (xyz, mapData)
get scattering density at a point by 8-point interpolation
param xyz: coordinate to be probed
param: mapData: dict of map data

Returns density at xyz

GSASIImath.getRhos (XYZ, rho)
get scattering density at an array of point by 8-point interpolation this is faster than getRho which is only used for single points. However, getRhos is replaced by scipy.ndimage.interpolation.map_coordinates which does a better job & is just as fast. Thus, getRhos is unused in GSAS-II at this time. param xyz: array coordinates to be probed
Nx3 param: rho: array copy of map (NB: don’t use original!)

Returns density at xyz

GSASIImath.getSyXYZ (XYZ, ops, SGData)
default doc

Parameters name (type) – description

Returns type name: description

GSASIImath.getTOFalpha (ins, dsp)
get TOF peak profile alpha
Parameters

- `ins (dict)` – instrument parameters with at least ‘alpha’ as values only
- `dsp (float)` – d-spacing of peak

Returns float getTOFalpha: peak alpha

GSASImath.getTOFalphaDeriv(dsp)
get derivatives of TOF peak profile beta wrt alpha

Parameters `dsp (float)` – d-spacing of peak

Returns float getTOFalphaDeriv: d(alp)/d(alpha)

GSASImath.getTOFbeta(ins, dsp)
get TOF peak profile beta

Parameters

- `ins (dict)` – instrument parameters with at least ‘beat-0’, ‘beta-1’ & ‘beta-q’ as values only
- `dsp (float)` – d-spacing of peak

Returns float getTOFbeta: peak beat

GSASImath.getTOFbetaDeriv(dsp)
get derivatives of TOF peak profile beta wrt beta-0, beta-1, & beat-q

Parameters `dsp (float)` – d-spacing of peak

Returns list getTOFbetaDeriv: d(beta)/d(beat-0), d(beta)/d(beta-1) & d(beta)/d(beta-q)

GSASImath.getTOFgamma(ins, dsp)
get TOF peak profile gamma

Parameters

- `ins (dict)` – instrument parameters with at least ‘X’, ‘Y’ & ‘Z’ as values only
- `dsp (float)` – d-spacing of peak

Returns float getTOFgamma: peak gamma

GSASImath.getTOFgammaDeriv(dsp)
get derivatives of TOF peak profile gamma wrt X, Y & Z

Parameters `dsp (float)` – d-spacing of peak

Returns list getTOFgammaDeriv: d(gam)/dX & d(gam)/dY

GSASImath.getTOFsig(ins, dsp)
get TOF peak profile sigma^2

Parameters

- `ins (dict)` – instrument parameters with at least ‘sig-0’, ‘sig-1’ & ‘sig-q’ as values only
- `dsp (float)` – d-spacing of peak

Returns float getTOFsig: peak sigma^2

GSASImath.getTOFsigDeriv(dsp)
get derivatives of TOF peak profile sigma^2 wrt sig-0, sig-1, & sig-q

Parameters `dsp (float)` – d-spacing of peak

Returns list getTOFsigDeriv: d(sig0)/d(sig-0), d(sig)/d(sig-1) & d(sig)/d(sig-q)
GSAS-II Developers Documentation, Release version 4249

GSASIImath.getTorsionDeriv (XYZ, Amat, Coeff)
default doc string

Parameters  name (type) – description

Returns  type name: description

GSASIImath.getVCov (varyNames, varyList, covMatrix)
obtain variance-covariance terms for a set of variables. NB: the varyList and covMatrix were saved by the last least squares refinement so they must match.

Parameters

• varyNames (list) – variable names to find v-cov matric for
• varyList (list) – full list of all variables in v-cov matrix
• covMatrix (nparray) – full variance-covariance matrix from the last least squares refinement

Returns  nparray vcov: variance-covariance matrix for the variables given in varyNames

GSASIImath.getWave (Parms)
returns wavelength from Instrument parameters dictionary

Parameters Parms (dict) – Instrument parameters; must contain: Lam: single wavelength or Lam1: Ka1 radiation wavelength

Returns  float wave: wavelength

GSASIImath.invQ (Q)
get inverse of quaternion q=r+ai+bj+ck; q* = r-ai-bj-ck

GSASIImath.makeQuat (A, B, C)
Make quaternion from rotation of A vector to B vector about C axis

Parameters  A, B, C (np.array) – Cartesian 3-vectors

Returns  quaternion & rotation angle in radians q=r+ai+bj+ck

GSASIImath.makeWaves (waveTypes, FSSdata, XSSdata, USSdata, MSSdata, Mast)
waveTypes: array nAtoms: ‘Fourier’, ‘ZigZag’ or ‘Block’ FSSdata: array 2 x atoms x waves (sin,cos terms) XSSdata: array 2x3 x atoms X waves (sin,cos terms) USSdata: array 2x6 x atoms X waves (sin,cos terms) MSSdata: array 2x3 x atoms X waves (sin,cos terms)
Mast: array orthogonalization matrix for Uij

GSASIImath.makeWavesDerv (ngl, waveTypes, FSSdata, XSSdata, USSdata, Mast)
Only for Fourier waves for fraction, position & adp (probably not used for magnetism) FSSdata: array 2 x atoms x waves (sin,cos terms) XSSdata: array 2x3 x atoms X waves (sin,cos terms) USSdata: array 2x6 x atoms X waves (sin,cos terms) Mast: array orthogonalization matrix for Uij

GSASIImath.mcsaSearch (data, RBdata, reflType, reflData, covData, pgbar, start=True)
default doc string

Parameters  name (type) – description

Returns  type name: description

GSASIImath.normQ (QA)
get length of quaternion & normalize it q=r+ai+bj+ck

GSASIImath.pinv (a, rcond=1e-15)
Compute the (Moore-Penrose) pseudo-inverse of a matrix. Modified from numpy.linalg.pinv; assumes a is
Hessian & returns no. zeros found Calculate the generalized inverse of a matrix using its singular-value decomposition (SVD) and including all large singular values.

**Parameters**

- `a (array)` – (M, M) array_like - here assumed to be LS Hessian Matrix to be pseudo-inverted.
- `rcond (float)` – Cutoff for small singular values. Singular values smaller (in modulus) than `rcond * largest_singular_value` (again, in modulus) are set to zero.

**Returns**  
`B : (M, M) ndarray` The pseudo-inverse of `a`

**Raises:** `LinAlgError` If the SVD computation does not converge.

**Notes:** The pseudo-inverse of a matrix `A`, denoted `A⁺`, is defined as: “the matrix that ‘solves’ [the least-squares problem] `Ax = b`,” i.e., if `x̂` is said solution, then `A⁺` is that matrix such that `x̂ = A⁺b`.

It can be shown that if `Q₁ΣQ₂ᵀ = A` is the singular value decomposition of `A`, then `A⁺ = Q₂Σ⁺Q₁ᵀ`, where `Q₁,₂` are orthogonal matrices, `Σ` is a diagonal matrix consisting of `A`'s so-called singular values, (followed, typically, by zeros), and then `Σ⁺` is simply the diagonal matrix consisting of the reciprocals of `A`'s singular values (again, followed by zeros). [1]


```
GSASIImath.printRho (SGLaue, rho, rhoMax)
```

default doc string

**Parameters**  
name (**type**) – description

**Returns**  
type name: description

```
GSASIImath.prodQQ (QA, QB)
```

Grassman quaternion product QA,QB quaternions; q=r+ai+bj+ck

```
GSASIImath.prodQVQ (Q, V)
```

compute the quaternion vector rotation qvq⁻¹ = v' q=r+ai+bj+ck

```
GSASIImath.randomAVdeg (r0, r1, r2, r3)
```

create random angle (deg),vector from 4 random number in range (-1,1)

```
GSASIImath.randomQ (r0, r1, r2, r3)
```

create random quaternion from 4 random numbers in range (-1,1)

```
GSASIImath.setPeakparms (Parms,Parms2,pos,mag,ifQ=False,useFit=False)
```

set starting peak parameters for single peak fits from plot selection or auto selection

**Parameters**

- `Parms (dict)` – instrument parameters dictionary
- `Parms2 (dict)` – table lookup for TOF profile coefficients
- `pos (float)` – peak position in 2-theta, TOF or Q (ifQ=True)
- `mag (float)` – peak top magnitude from pick
- `ifQ (bool)` – True if pos in Q
- `useFit (bool)` – True if use fitted CW Parms values (not defaults)

**Returns**  
list XY: peak list entry: for CW: [pos,0,mag,1,sig,0,gam,0] for TOF: [pos,0,mag,1,alp,0,bet,0,sig,0,gam,0] NB: mag refinement set by default, all others off
GSASIImath.sortArray(data, pos, reverse=False)
data is a list of items sort by pos in list; reverse if True

GSASIImath.wavekE(wavekE)
Convert wavelength to energy & vise versa
:param float wavekE: wavelength in A or energy in kE
:return float wavekE: the other one

1.11 GSASIIindex: Cell Indexing Module

Cell indexing program: variation on that of A. Coehlo includes cell refinement from peak positions

GSASIIindex.A2values(ibrav, A)
needs a doc string

GSASIIindex.DoIndexPeaks(peaks, controls, bravais, dlg, ifX20=True)
needs a doc string

GSASIIindex.FitHKL(ibrav, peaks, A, Pwr)
needs a doc string

GSASIIindex.FitHKLT(difC, ibrav, peaks, A, Z, Zref)
needs a doc string

GSASIIindex.FitHKLTSS(difC, ibrav, peaks, A, V, Vref, Z, Zref)
needs a doc string

GSASIIindex.FitHKLZ(wave, ibrav, peaks, A, Z, Zref)
needs a doc string

GSASIIindex.FitHKLZSS(wave, ibrav, peaks, A, V, Vref, Z, Zref)
needs a doc string

GSASIIindex.IndexPeaks(peaks, HKL)
needs a doc string

GSASIIindex.IndexSSPeaks(peaks, HKL)
needs a doc string

GSASIIindex.TestData()
needs a doc string

GSASIIindex.Values2A(ibrav, values)
needs a doc string

GSASIIindex.calc_M20(peaks, HKL, ifX20=True)
needs a doc string

GSASIIindex.calc_M20SS(peaks, HKL)
needs a doc string

GSASIIindex.findBestCell(dlg, ncMax, A, Ntries, ibrav, peaks, V1, ifX20=True)
needs a doc string

GSASIIindex.getDmax(peaks)
needs a doc string

GSASIIindex.getDmin(peaks)
needs a doc string
GSASIIindex.**halfCell**(ibrav, A, peaks)
   needs a doc string

GSASIIindex.**monoCellReduce**(ibrav, A)
   needs a doc string

GSASIIindex.**oddPeak**(indx, peaks)
   needs a doc string

GSASIIindex.**ran2axis**(k, N)
   needs a doc string

GSASIIindex.**ranAbyR**(Bravais, A, k, N, ranFunc)
   needs a doc string

GSASIIindex.**ranAbyV**(Bravais, dmin, dmax, V)
   needs a doc string

GSASIIindex.**ranaxis**(dmin, dmax)
   needs a doc string

GSASIIindex.**rancell**(Bravais, dmin, dmax)
   needs a doc string

GSASIIindex.**refinePeaks**(peaks, ibrav, A, ifX20=True)
   needs a doc string

GSASIIindex.**refinePeaksT**(peaks, difC, ibrav, A, Zero, ZeroRef)
   needs a doc string

GSASIIindex.**refinePeaksTSS**(peaks, difC, Inst, SGData, SSGData, maxH, ibrav, A, vec, vecRef, Zero, ZeroRef)
   needs a doc string

GSASIIindex.**refinePeaksZ**(peaks, wave, ibrav, A, Zero, ZeroRef)
   needs a doc string

GSASIIindex.**refinePeaksZSS**(peaks, wave, Inst, SGData, SSGData, maxH, ibrav, A, vec, vecRef, Zero, ZeroRef)
   needs a doc string

GSASIIindex.**rotOrthoA**(A)
   needs a doc string

GSASIIindex.**scaleAbyV**(A, V)
   needs a doc string

GSASIIindex.**sortM20**(cells)
   needs a doc string

GSASIIindex.**swapMonoA**(A)
   needs a doc string

### 1.12 GSASIIplot: plotting routines

This module performs all visualization using matplotlib and OpenGL graphics. The following plotting routines are defined:
<table>
<thead>
<tr>
<th>plotting routine</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>PlotPatterns()</td>
<td>Powder pattern plotting</td>
</tr>
<tr>
<td>PublishRietveldPlot()</td>
<td>Create publication-quality Rietveld plots from PlotPatterns() plot</td>
</tr>
<tr>
<td>PlotImage()</td>
<td>Plots of 2D detector images</td>
</tr>
<tr>
<td>PlotPeakWidths()</td>
<td>Plot instrument broadening terms as function of 2-theta/TOF</td>
</tr>
<tr>
<td>PlotCovariance()</td>
<td>Show covariance terms in 2D</td>
</tr>
<tr>
<td>PlotStructure()</td>
<td>Crystal structure plotting with balls, sticks, lines, ellipsoids, polyhedra and magnetic moments</td>
</tr>
<tr>
<td>Plot1DSngl()</td>
<td>1D stick plots of structure factors</td>
</tr>
<tr>
<td>PlotSngl()</td>
<td>Structure factor plotting</td>
</tr>
<tr>
<td>Plot3DSngl()</td>
<td>3D Structure factor plotting</td>
</tr>
<tr>
<td>PlotDeltSig()</td>
<td>Normal probability plot (powder or single crystal)</td>
</tr>
<tr>
<td>PlotISFG()</td>
<td>PDF analysis: displays I(Q), S(Q), F(Q) and G(r)</td>
</tr>
<tr>
<td>PlotCalib()</td>
<td>CW or TOF peak calibration</td>
</tr>
<tr>
<td>PlotXY()</td>
<td>Simple plot of xy data</td>
</tr>
<tr>
<td>PlotXYZ()</td>
<td>Simple contour plot of xyz data</td>
</tr>
<tr>
<td>PlotAAProb()</td>
<td>Protein “quality” plot</td>
</tr>
<tr>
<td>PlotStrain()</td>
<td>Plot of strain data, used for diagnostic purposes</td>
</tr>
<tr>
<td>PlotSASDSzdDist()</td>
<td>Small angle scattering size distribution plot</td>
</tr>
<tr>
<td>PlotPowderLines()</td>
<td>Plot powder pattern as a stick plot (vertical lines)</td>
</tr>
<tr>
<td>PlotSizeStrainPO()</td>
<td>Plot 3D mustrain/size/preferred orientation figure</td>
</tr>
<tr>
<td>PlotTexture()</td>
<td>Pole figure, inverse pole figure plotting</td>
</tr>
<tr>
<td>ModulationPlot()</td>
<td>Plots modulation information</td>
</tr>
<tr>
<td>PlotTorsion()</td>
<td>Plots MC torsion angles</td>
</tr>
<tr>
<td>PlotRama()</td>
<td>Ramachandran of energetically allowed regions for dihedral angles in protein</td>
</tr>
<tr>
<td>PlotSelectedSequence()</td>
<td>Plot one or more sets of values selected from the sequential refinement table</td>
</tr>
<tr>
<td>PlotIntegration()</td>
<td>Rectified plot of 2D image after image integration with 2-theta and azimuth as coordinates</td>
</tr>
<tr>
<td>PlotTRImage()</td>
<td>test plot routine</td>
</tr>
<tr>
<td>PlotRigidBody()</td>
<td>show rigid body structures as balls &amp; sticks</td>
</tr>
<tr>
<td>PlotLayers()</td>
<td>show layer structures as balls &amp; sticks</td>
</tr>
<tr>
<td>PlotFPAConvolutors()</td>
<td>plots the convolutors from Fundamental Parameters</td>
</tr>
</tbody>
</table>

These plotting routines place their graphics in the GSAS-II Plot Window, which contains a G2PlotNoteBook tabbed panel allowing multiple plots to be viewed. Methods G2PlotNoteBook.addMpl() (2-D matplotlib), G2PlotNoteBook.add3D() (3-D matplotlib), and G2PlotNoteBook.addOgl() (OpenGL) are used to create tabbed plot objects to hold plots of the following classes: G2PlotMpl (2-D matplotlib), G2Plot3D (3-D matplotlib), and G2PlotOgl (OpenGL). Note that two G2PlotNoteBook methods are potentially used to determine how plot updates after a refinement are handled:

<table>
<thead>
<tr>
<th>class method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G2PlotNoteBook. RegisterRedrawRoutine</td>
<td>This specifies a function to redraw the plot after the data tree has been reloaded. Be sure this updates data objects with new values from the tree, when needed.</td>
</tr>
<tr>
<td>G2PlotNoteBook. SetNoDelete()</td>
<td>Use this to indicate that a plot does not need to be updated after a refinement and should not be closed.</td>
</tr>
</tbody>
</table>

These two methods define the following attributes (variables) in the plot tab classes:
<table>
<thead>
<tr>
<th>variable</th>
<th>default</th>
<th>use</th>
</tr>
</thead>
<tbody>
<tr>
<td>replotFunction</td>
<td>None</td>
<td>Defines a routine to be called to update the plot after a refinement (unless None). Use <code>G2PlotNoteBook.RegisterRedrawRoutine()</code> to define this (and replotArgs &amp; replotKWArgs). Plotting functions that take significant time to complete should probably not use this.)</td>
</tr>
<tr>
<td>replotArgs</td>
<td>[]</td>
<td>Defines the positional arguments to be supplied to the replotFunction function or method.</td>
</tr>
<tr>
<td>replotKWArgs</td>
<td>{}</td>
<td>Defines the keyword arguments to be supplied to the replotFunction function or method.</td>
</tr>
<tr>
<td>plotRequiresRedraw</td>
<td>True</td>
<td>If set to True, after a refinement, the plot will be closed (in <code>GSASIIdataGUI.GSASII.CleanupOldPlots()</code>) if it was not updated after the refinement. Set this to False using <code>G2PlotNoteBook.SetNoDelete()</code> for plots that should not be deleted or do not change based on refinement results.</td>
</tr>
<tr>
<td>plotInvalid</td>
<td>False</td>
<td>Used to track if a plot has been updated. Set to False in <code>G2PlotNoteBook.FindPlotTab()</code> when a plot is drawn. After a refinement is completed, method <code>GSASIIdataGUI.GSASII.ResetPlots()</code> sets plotInvalid to False for all plots before any routines are called.</td>
</tr>
</tbody>
</table>

Note that the plot toolbar is customized with `GSASIItoolbar`.

`GSASIIplot.ComputeArc`(angI, angO, wave, azm0=0, azm1=362)

Computes arc/ring arrays in with inner and outer radii from angI,angO and beginning and ending azimuths azm0,azm1 (optional). Returns the inner and outer ring/arc arrays.

`GSASIIplot.CopyRietveldPlot`(G2frame, Pattern, Plot, Page, figure)

Copy the contents of the Rietveld graph from the plot window to another mpl figure which can be on screen or can be a file for hard copy. Uses values from Pattern to also generate a delta/sigma plot below the main figure, since the weights are not available from the plot.

Parameters

- **Pattern**(list) – histogram object from data tree
- **Plot**(mpl.axes) – The axes object from the Rietveld plot
- **Page**(wx.Panel) – The tabbed panel for the Rietveld plot
- **figure**(mpl.figure) – The figure object from the Rietveld plot

`class GSASIIplot.G2Plot3D`(parent, id=-1, dpi=None, **kwargs)

Creates a 3D Matplotlib plot in the GSAS-II graphics window

`class GSASIIplot.G2PlotMpl`(parent, id=-1, dpi=None, publish=None, **kwargs)

Creates a Matplotlib 2-D plot in the GSAS-II graphics window

`class GSASIIplot.G2PlotNoteBook`(parent, id=-1, G2frame=None)

create a tabbed panel to hold a GSAS-II graphics window

- **Delete**(name)
  delete a tabbed page

- **FindPlotTab**(label, Type, newImage=True, publish=None)
  Open a plot tab for initial plotting, or raise the tab if it already exists. Set a flag (Page.plotInvalid) that it has been redrawn. Record the name of the this plot in self.lastRaisedPlotTab

- **GetTabIndex**(label)
  Look up a tab label and return the index in the notebook (this appears to be independent to the order it is dragged to – at least in Windows) as well as the associated wx.Panel

### 1.12. **GSASIIplot**: plotting routines
An exception is raised if the label is not found

**InvokeTreeItem**(*pid*)
This is called to select an item from the tree using the `self.allowZoomReset` flag to prevent a reset to the zoom of the plot (where implemented)

**OnNotebookKey**(event)
Called when a keystroke event gets picked up by the notebook window rather than the child. This is not expected, but somehow it does sometimes on the Mac and perhaps Linux.

Assume that the page associated with the currently displayed tab has a child, `.canvas`; give that child the focus and pass it the event.

**OnPageChanged**(event)
respond to someone pressing a tab on the plot window. Called when a plot tab is clicked. On some platforms (Mac for sure) this is also called when a plot is created or selected with `.SetSelection()` or `.SetFocus()`.

(removed) The `self.skipPageChange` is used variable is set to suppress repeated replotting.

**RaisePageNoRefresh**(Page)
Raises a plot tab without triggering a refresh via `OnPageChanged`

**RegisterRedrawRoutine**(name, routine=None, args=(), kwargs={})
Save information to determine how to redraw a plot:
- **param** str name: label on tab of plot
- **param** Object routine: a function to be called
- **param** args: a list of positional parameters for the function
- **param** kwargs: a dict with keyword parameters for the function

**Rename**(oldName, newName)
rename a tab

**SetNoDelete**(name)
Indicate that a plot does not need to be redrawn

**SetSelectionNoRefresh**(plotNum)
Raises a plot tab without triggering a refresh via `OnPageChanged`

**add3D**(name=“”)
Add a tabbed page with a 3D plot

**addMpl**(name=“”, publish=None)
Add a tabbed page with a matplotlib plot

**addOgl**(name=“”)
Add a tabbed page with an openGL plot

**clear**()
clear all pages from plot window

class **GSASIIplot.G2PlotOgl**(parent, id=-1, dpi=None, **kwargs)
Creates an OpenGL plot in the GSAS-II graphics window

class **GSASIIplot.GSASIItoolbar**(plotCanvas, publish=None)
Override the matplotlib toolbar so we can add more icons

**OnArrow**(event)
reposition limits to scan or zoom by button press

**OnHelp**(event)
Respond to press of help button on plot toolbar

**OnKey**(event)
Provide user with list of keystrokes defined for plot as well as an alternate way to access the same functionality
get_zoompan()

Return “ZOOM” if Zoom is active, “PAN” if Pan is active, or None if neither.

reset_zoompan()

Turns off Zoom or Pan mode, if on. Ignored if neither is set.

GSASIIplot.MPLsubplots(figure, nrows=1, ncols=1, sharex=False, sharey=False, squeeze=True, subplot_kw=None, gridspec_kw=None)

Add a set of subplots to this figure.

This is a copy of Figure.figure.subplots from matplotlib.figure.py. Included here because this appears only in later versions of matplotlib. When v2.2 is standard, this should be removed from GSAS-II.

Parameters

- **nrows, ncols (int)** – default: 1 Number of rows/cols of the subplot grid.
- **sharex, sharey (bool/str)** – True/False or {'none', 'all', 'row', 'col'}, default: False

Controls sharing of properties among x (sharex) or y (sharey) axes:

- True or 'all': x- or y-axis will be shared among all subplots.
- False or 'none': each subplot x- or y-axis will be independent.
- 'row': each subplot row will share an x- or y-axis.
- 'col': each subplot column will share an x- or y-axis.

When subplots have a shared x-axis along a column, only the x tick labels of the bottom subplot are visible. Similarly, when subplots have a shared y-axis along a row, only the y tick labels of the first column subplot are visible.

- **squeeze (bool)** – default: True

- If True, extra dimensions are squeezed out from the returned axis object:
  - if only one subplot is constructed (nrows=ncols=1), the resulting single Axes object is returned as a scalar.
  - for Nx1 or 1xN subplots, the returned object is a 1D numpy object array of Axes objects are returned as numpy 1D arrays.
  - for NxM, subplots with N>1 and M>1 are returned as a 2D arrays.

- If False, no squeezing at all is done: the returned Axes object is always a 2D array containing Axes instances, even if it ends up being 1x1.

- **subplot_kw (dict)** – default: {} Dict with keywords passed to the matplotlib.figure.Figure.add_subplot() call used to create each subplots.

- **gridspec_kw (dict)** – default: {} Dict with keywords passed to the matplotlib.gridspec.GridSpec constructor used to create the grid the subplots are placed on.

Returns

A single Axes object or array of Axes objects with the added axes. The dimensions of the resulting array can be controlled with the squeeze keyword, see above.

See also: pyplot.subplots: pyplot API; docstring includes examples.

GSASIIplot.ModulationPlot(G2frame, data, atom, ax, off=0)

Needs a description

GSASIIplot.OnStartMask(G2frame)

Initiate the start of a Frame or Polygon map, etc. Called from a menu command (GSASIIimgGUI) or from

1.12. **GSASIIplot: plotting routines**
OnImPlotKeyPress. Variable G2frame.MaskKey contains a single letter (‘f’ or ‘p’, etc.) that determines what type of mask is created.

**Parameters**

- **G2frame** (*wx.Frame*) – The main GSAS-II tree “window”

**GSASIIplot.OnStartNewDzero** *(G2frame)*

Initiate the start of adding a new d-zero to a strain data set

**Parameters**

- **G2frame** (*wx.Frame*) – The main GSAS-II tree “window”
- **eventkey** (*str*) – a single letter (‘a’) that triggers the addition of a d-zero.

**GSASIIplot.Plot1DSngl** *(G2frame, newPlot=False, hklRef=None, Super=0, Title=False)*

1D Structure factor plotting package - displays reflections as sticks proportional to F, F**2**, etc. as requested

**GSASIIplot.Plot3DSngl** *(G2frame, newPlot=False, Data=None, hklRef=None, Title=False)*

3D Structure factor plotting package - displays reflections as rings proportional to F, F**2**, etc. as requested as 3D array

**GSASIIplot.PlotAAProb** *(G2frame, resNames, Probs1, Probs2, Title=”, thresh=None, pickHandler=None)*

Needs a description

**GSASIIplot.PlotBeadModel** *(G2frame, Atoms, defaults, PDBtext)*

Bead model plotting package. For bead models from SHAPES

**GSASIIplot.PlotCalib** *(G2frame, Inst, XY, Sigs, newPlot=False)*

Plot of CW or TOF peak calibration

**GSASIIplot.PlotCovariance** *(G2frame, Data)*

Plots the covariance matrix. Also shows values for parameters and their standard uncertainties (esd’s) or the correlation between variables.

**GSASIIplot.PlotDeltSig** *(G2frame, kind, PatternName=None)*

Produces normal probability plot for a powder or single crystal histogram

**GSASIIplot.PlotExposedImage** *(G2frame, newPlot=False, event=None)*

General access module for 2D image plotting

**GSASIIplot.PlotFPAConvolutors** *(G2frame, NISTpk)*

Plot the convolutions used for the current peak computed with **GSASIIfpGUI.doFPACalc()**

**GSASIIplot.PlotISFG** *(G2frame, data, newPlot=False, plotType=”, peaks=None)*

Plotting package for PDF analysis; displays I(Q), S(Q), F(Q) and G(r) as single or multiple plots with waterfall and contour plots as options

**GSASIIplot.PlotImage** *(G2frame, newPlot=False, event=None, newImage=True)*

Plot of 2D detector images as contoured plot. Also plot calibration ellipses, masks, etc. Plots whatever is in G2frame.ImageZ

**Parameters**

- **G2frame** (*wx.Frame*) – main GSAS-II frame
- **newPlot** (*bool*) – if newPlot is True, the plot is reset (zoomed out, etc.)
- **event** – matplotlib mouse event (or None)
- **newImage** (*bool*) – If True, the Figure is cleared and redrawn

**GSASIIplot.PlotIntegration** *(G2frame, newPlot=False, event=None)*

Plot of 2D image after image integration with 2-theta and azimuth as coordinates
GSASIIplot.PlotLayers (G2frame, Layers, laySeq, defaults)
Layer plotting package. Can show layer structures as balls & sticks

GSASIIplot.PlotPatterns (G2frame, newPlot=False, plotType=’PWDR’, data=None, extraKeys=[], refineMode=False)
Powder pattern plotting package - displays single or multiple powder patterns as intensity vs 2-theta, q or TOF. Can display multiple patterns as “waterfall plots” or contour plots. Log I plotting available.

Note that plotting information will be found in: G2frame.PatternId (contains the tree item for the current histogram)
G2frame.PickId (contains the actual selected tree item (can be child of histogram)
G2frame.HKL (used for tool tip display of hkl for selected phase reflection list)

GSASIIplot.PlotPeakWidths (G2frame, PatternName=None)
Plotting of instrument broadening terms as function of 2-theta Seen when “Instrument Parameters” chosen from powder pattern data tree. Parameter PatternName allows the PWDR to be referenced as a string rather than a wx tree item, defined in G2frame.PatternId.

GSASIIplot.PlotPowderLines (G2frame)
plotting of powder lines (i.e. no powder pattern) as sticks

GSASIIplot.PlotRama (G2frame, phaseName, Rama, RamaName, Names=[], PhiPsi=[], Coeff=[]) 
needs a doc string

GSASIIplot.PlotRigidBody (G2frame, rbType, AtInfo, rbData, defaults)
RB plotting package. Can show rigid body structures as balls & sticks

GSASIIplot.PlotSASDPairDist (G2frame)
Needs a description

GSASIIplot.PlotSASDSizesDist (G2frame)
Needs a description

GSASIIplot.PlotSelectedSequence (G2frame, ColumnList, TableGet, SelectX, fitnum=None, fitvals=None)
Plot a result from a sequential refinement

Parameters
- G2frame (wx.Frame) – The main GSAS-II tree “window”
- ColumnList (list) – list of int values corresponding to columns selected as y values
- TableGet (function) – a function that takes a column number as argument and returns the column label, the values and there ESDs (or None)
- SelectX (function) – a function that returns a selected column number (or None) as the X-axis selection

GSASIIplot.PlotSizeStrainPO (G2frame, data, hist=",\n\nStart=False)
Plot 3D mustrain/size/preferred orientation figure. In this instance data is for a phase

GSASIIplot.PlotSngl (G2frame, newPlot=False, Data=None, hklRef=None, Title="")
Structure factor plotting package - displays zone of reflections as rings proportional to F, F**2, etc. as requested

GSASIIplot.PlotStrain (G2frame, data, newPlot=False)
plot of strain data, used for diagnostic purposes

GSASIIplot.PlotStructure (G2frame, data, firstCall=False)
Crystal structure plotting package. Can show structures as balls, sticks, lines, thermal motion ellipsoids and polyhedra. Magnetic moments shown as black/red arrows according to spin state
GSASIIplot.PlotTRImage(G2frame, tax, tay, taz, newPlot=False)
   a test plot routine - not normally used

GSASIIplot.PlotTexture(G2frame, data, Start=False)
Pole figure, inverse pole figure plotting. dict generalData contains all phase info needed which is in data

GSASIIplot.PlotTorsion(G2frame, phaseName, Torsion, TorName, Names=[], Angles=[], Coeff=[]) needs a doc string

GSASIIplot.PlotXY(G2frame, XY, XY2=[], labelX='X', labelY='Y', newPlot=False, Title='', lines=False, names=[], names2=[], vertLines=[])
simple plot of xy data
   Parameters
   • G2frame (wx.Frame) – The main GSAS-II tree “window”
   • XY (list) – a list of X,Y array pairs; len(X) = len(Y)
   • XY2 (list) – a secondary list of X,Y pairs
   • labelX (str) – label for X-axis
   • labelY (str) – label for Y-axis
   • newPlot (bool) – =True if new plot is to be made
   • Title (str) – title for plot
   • lines (bool) – = True if lines desired for XY plot; XY2 always plotted as lines
   • names (list) – legend names for each XY plot as list a of str values
   • names2 (list) – legend names for each XY2 plot as list a of str values
   • vertLines (list) – lists of vertical line x-positions; can be one for each XY

   Returns nothing

GSASIIplot.PlotXYZ(G2frame, XY, Z, labelX='X', labelY='Y', newPlot=False, Title='', zrange=None, color=None, buttonHandler=None)
simple contour plot of xyz data
   Parameters
   • G2frame (wx.Frame) – The main GSAS-II tree “window”
   • XY (list) – a list of X,Y arrays
   • Z (list) – a list of Z values for each X,Y pair
   • labelX (str) – label for X-axis
   • labelY (str) – label for Y-axis
   • newPlot (bool) – =True if new plot is to be made
   • Title (str) – title for plot
   • zrange (list) – [zmin,zmax]; default=None to use limits in Z
   • color (str) – one of mpl.cm.dated.keys(); default=None to use G2frame.ContourColor

   Returns nothing

GSASIIplot.PublishRietveldPlot(G2frame, Pattern, Plot, Page)
   Show a customizable “Rietveld” plot and export as a publication-quality file. Will only work when a single pattern is displayed.
Parameters

- **G2Frame** *(wx.Frame)* – the main GSAS-II window
- **Pattern** *(list)* – list of np.array items with obs, calc (etc.) diffraction pattern
- **Plot** *(mpl.axes)* – axes of the graph in plot window
- **Page** *(wx.Panel)* – tabbed panel containing the plot

**GSASIIplot.ReplotPattern** *(G2frame, newPlot, plotType, PatternName=None, PickName=None)*

This does the same as PlotPatterns except that it expects the information to be plotted (pattern name, item picked in tree + eventually the reflection list) to be passed as names rather than references to wx tree items, defined as class entries

**GSASIIplot.ToggleMultiSpotMask** *(G2frame)*

Turns on and off MultiSpot selection mode; displays a subtitle on plot the is cleared by the next PlotImage call

**GSASIIplot.UpdatePolygon** *(pick, event, polygon)*

Update a polygon (or frame) in response to the location of the mouse. Delete the selected point if moved on top of another. With right button add a point after the current button.

**GSASIIplot.Write2csv** *(fil, dataItems, header=False)*

Write a line to a CSV file

Parameters

- **fil** *(object)* – file object
- **dataItems** *(list)* – items to write as row in file
- **header** *(bool)* – True if all items should be written with quotes (default is False)

**GSASIIplot.changePlotSettings** *(G2frame, Plot)*

Code in development to allow changes to plot settings prior to export of plot with “floppy disk” button

**GSASIIplot.onLegendPick** *(event)*

When a line in the legend is selected, find the matching line in the plot and then highlight it by adding/enlarging markers. Set up a timer to make a reset after delay selected in SetupLegendPick

### 1.13 GSASII powder calculation module

**GSASIIpwd.Absorb** *(Geometry, MuR, Tth, Phi=0, Psi=0)*

Calculate sample absorption


: param float MuR: absorption coeff * sample thickness/2 or radius

: param Tth: 2-theta scattering angle - can be numpy array

: param float Phi: flat plate tilt angle - future

: param float Psi: flat plate tilt axis - future

**GSASIIpwd.AbsorbDerv** *(Geometry, MuR, Tth, Phi=0, Psi=0)*

needs a doc string

**GSASIIpwd.CalcPDF** *(data, inst, limits, xdata)*

Computes I(Q), S(Q) & G(r) from Sample, Bkg, etc. diffraction patterns loaded into dict xdata; results are placed in xdata. Calculation parameters are found in dicts data and inst and list limits. The return value is at present an empty list.

**GSASIIpwd.Dict2Values** *(parmdict, varylist)*

Use before call to leastsq to setup list of values for the parameters in parmdict, as selected by key in varylist
**GSASIIpwd.DoPeakFit** *(FitPgm, Peaks, Background, Limits, Inst, Inst2, data, fixback=None, prevVaryList=[], oneCycle=False, controls=None, dlg=None)*

Called to perform a peak fit, refining the selected items in the peak table as well as selected items in the background.

**Parameters**

- **FitPgm** (*str*) – type of fit to perform. At present this is ignored.
- **Peaks** (*list*) – a list of peaks. Each peak entry is a list with 8 values: four values followed by a refine flag where the values are: position, intensity, sigma (Gaussian width) and gamma (Lorentzian width). From the Histogram/”Peak List” tree entry, dict item “peaks”
- **Background** (*list*) – describes the background. List with two items. Item 0 specifies a background model and coefficients. Item 1 is a dict. From the Histogram/Background tree entry.
- **Limits** (*list*) – min and max x-value to use
- **Inst** (*dict*) – Instrument parameters
- **Inst2** (*dict*) – more Instrument parameters
- **data** (*numpy.array*) – a 5xn array. data[0] is the x-values, data[1] is the y-values, data[2] are weight values, data[3], [4] and [5] are calc, background and difference intensities, respectively.
- **fixback** (*array*) – fixed background values
- **prevVaryList** (*list*) – Used in sequential refinements to override the variable list. Defaults as an empty list.
- **oneCycle** (*bool*) – True if only one cycle of fitting should be performed
- **controls** (*dict*) – a dict specifying two values, Ftol = controls[‘min dM/M’] and deriv-Type = controls[‘deriv type’]. If None default values are used.
- **dlg** (*wx.Dialog*) – A dialog box that is updated with progress from the fit. Defaults to None, which means no updates are done.

**GSASIIpwd.GetAsfMean** *(ElList, Sthl2)*

Calculate various scattering factor terms for PDF calcs

**Parameters**

- **ElList** (*dict*) – element dictionary contains scattering factor coefficients, etc.
- **Sthl2** (*np.array*) – numpy array of sin theta/lambda squared values

**Returns** *mean(f^2), mean(f)^2, mean(compton)*

**GSASIIpwd.GetNumDensity** *(ElList, Vol)*

needs a doc string

**GSASIIpwd.LorchWeight** *(Q)*

needs a doc string

**GSASIIpwd.MEMupdateReflData** *(prfName, data, reflData)*

Update reflection data with new Fosq. phase result from Dysnomia

**Parameters**

- **prfName** (*str*) – phase.mem file name
- **reflData** (*list*) – GSAS-II reflection data
GSASIIpwd.Oblique(ObCoeff, Tth)
currently assumes detector is normal to beam

GSASIIpwd.PhaseWtSum(G2frame, histo)
Calculate sum of phase mass*phase fraction for PWDR data (exclude magnetic phases)

Parameters
- G2frame – GSASII main frame structure
- histo(str) – histogram name

Returns sum(scale*mass) for phases in histo

GSASIIpwd.Polarization(Pola, Tth, Azm=0.0)
Calculate angle dependent x-ray polarization correction (not scaled correctly!)

Parameters
- Pola – polarization coefficient e.g 1.0 fully polarized, 0.5 unpolarized
- Azm – azimuthal angle e.g. 0.0 in plane of polarization
- Tth – 2-theta scattering angle - can be numpy array which (if either) of these is “right”?

Returns (pola, dpdPola) * pola = ((1-Pola)*npcosd(Azm)**2+Pola*npsind(Azm)**2)*npcosd(Tth)**2+(1-Pola)*npsind(Azm)**2+Pola*npcosd(Azm)**2 * dpdPola: derivative needed for least squares

GSASIIpwd.Ruland(RulCoff, wave, Q, Compton)
needs a doc string

GSASIIpwd.SetBackgroundParms(Background)
Loads background parameters into dicts/lists to create varylist & parmdict

GSASIIpwd.StackSim(Layers, ctrls, scale=0.0, background={}, limits=[], inst={}, profile={})
Simulate powder or selected area diffraction pattern from stacking faults using DIFFaX

Parameters
- Layers(dict) – dict with following items
  - ['Laue':'1', 'Cell':[False,1.,1.,1.,90.,90.,90.,1.],
   'Width':[[10.,10.],[False,False]],'Toler':0.01,'AtInfo':{},
   'Layers':[],'Stacking':[],'Transitions':[]]

- ctrls(str) – controls string to be written on DIFFaX controls.dif file
- scale(float) – scale factor
- background(dict) – background parameters
- limits(list) – min/max 2-theta to be calculated
- inst(dict) – instrument parameters dictionary
- profile(list) – powder pattern data

Note that parameters all updated in place

GSASIIpwd.SurfaceRough(SRA, SRB, Tth)
Suortti (J. Appl. Cryst, 5,325-331, 1972) surface roughness correction :param float SRA: Suortti surface roughness parameter :param float SRB: Suortti surface roughness parameter :param float Tth: 2-theta(deg) - can be numpy array
GSASIIpwd.**SurfaceRoughDerv**(SRA, SRB, Tth)
Suortti surface roughness correction derivatives:
- **param SRA**: Suortti surface roughness parameter (dimensionless)
- **param SRB**: Suortti surface roughness parameter (dimensionless)
- **param Tth**: 2-theta(deg)
- can be numpy array
- return list: [dydSRA, dydSRB] derivatives to be used for intensity derivative

GSASIIpwd.**TestData()**
need a doc string

GSASIIpwd.**Transmission**(Geometry, Abs, Diam)
Calculate sample transmission

**Parameters**
- **Abs** *(float)* — absorption coeff in cm\(^{-1}\)
- **Diam** *(float)* — sample thickness/diameter in mm

GSASIIpwd.**Values2Dict**(parmdict, varylist, values)
Use after call to leastsq to update the parameter dictionary with values corresponding to keys in varylist

GSASIIpwd.**abeles**(kz, depth, rho, irho=0, sigma=0)
Optical matrix form of the reflectivity calculation. O.S. Heavens, Optical Properties of Thin Solid Films
Reflectometry as a function of kz for a set of slabs.

**Parameters**
- **kz** — float[n] (1/Ang). Scattering vector, \(2\pi \sin(\theta)/\lambda\). This is \(\frac{1}{2}Q_z\).
- **depth** — float[m] (Ang). thickness of each layer. The thickness of the incident medium and substrate are ignored.
- **rho** — float[n,k] (1e-6/Ang^2) Real scattering length density for each layer for each kz
- **irho** — float[n,k] (1e-6/Ang^2) Imaginary scattering length density for each layer for each kz Note: absorption cross section \(\mu = 2i\rho/\lambda\) for neutrons
- **sigma** — float[m-1] (Ang) interfacial roughness. This is the roughness between a layer and the previous layer. The sigma array should have m-1 entries.

Slabs are ordered with the surface SLD at index 0 and substrate at index -1, or reversed if \(kz < 0\).

GSASIIpwd.**calcIncident**(Iparm, xdata)
nedd a doc string

**class** GSASIIpwd.**cauchy_gen**(\*args, **kwargs)
nedd a doc string

GSASIIpwd.**ellipseSize**(H, Sij, GB)
Implements \(r = 1/\sqrt{\sum(Sij(q,v)^2)}\) per note from Alexander Brady

GSASIIpwd.**ellipseSizeDerv**(H, Sij, GB)
nedd a doc string

GSASIIpwd.**factorize**(num)
Provide prime number factors for integer num :returns: dictionary of prime factors (keys) & power for each (data)

**class** GSASIIpwd.**fcjde_gen**(\*args, **kwargs)

**Parameters**
• $x$ – array -1 to 1
• $t$ – 2-theta position of peak
• $s$ – sum($S/L,H/L$); $S$: sample height, $H$: detector opening, $L$: sample to detector opening distance
• $dx$ – 2-theta step size in deg

Returns
for fcj.pdf
• $T = x*dx+t$
• $s = S/L+H/L$
• if $x < 0$:
  $\text{fcj.pdf} = \frac{1}{\sqrt{(\cos(T)\cos(t))^2-1}} - \frac{1}{s} \div |\cos(T)|$
• if $x >= 0$: fcj.pdf = 0

GSASIIpwd.getBackground((pfx, parmDict, bakType, dataType, xdata, fixedBkg={})
Computes the background from vars pulled from gpix file or tree.

GSASIIpwd.getBackgroundDeriv(hfx, parmDict, bakType, dataType, xdata)
needs a doc string

GSASIIpwd.getEpsVoigt(pos, alp, bet, sig, gam, xdata)
needs a doc string

GSASIIpwd.getFCJVoigt(pos, intens, sig, gam, shl, xdata)
  Compute the Finger-Cox-Jepcoat modified Voigt function for a CW powder peak by direct convolution. This version is not used.

GSASIIpwd.getFCJVoigt3(pos, sig, gam, shl, xdata)
  Compute the Finger-Cox-Jepcoat modified Pseudo-Voigt function for a CW powder peak in external Fortran routine

GSASIIpwd.getFWHM(pos, Inst)
  Compute total FWHM from Thompson, Cox & Hastings (1987), J. Appl. Cryst. 20, 79-83 via getgamFW(g,s).

Parameters
  • pos – float peak position in deg 2-theta or tof in musec
  • Inst – dict instrument parameters

Returns float total FWHM of pseudoVoigt in deg or musec

GSASIIpwd.getHKLMpeak(dmin, Inst, SGData, SSGData, Vec, maxH, A)
needs a doc string

GSASIIpwd.getHKLpeak(dmin, SGData, A, Inst=None, nodup=False)
  Generates allowed by symmetry reflections with $d >= dmin$ NB: GenHLKf & checkMagextc return True for extinct reflections

Parameters
  • dmin – minimum d-spacing
  • SGData – space group data obtained from SpcGroup
  • A – lattice parameter terms A1-A6
  • Inst – instrument parameter info

1.13. GSASII powder calculation module
Returns HKLs: np.array hkl, etc for allowed reflections

GSASIIpwd.getPeakProfile\(\) \(\text{dataType, parmDict, xdata, varyList, bakType}\)
Computes the profile for a powder pattern

GSASIIpwd.getPeakProfileDeriv\(\) \(\text{dataType, parmDict, xdata, varyList, bakType}\)
needs a doc string

GSASIIpwd.getPsVoigt\(\) \(\text{pos, sig, gam, xdata}\)
needs a doc string

GSASIIpwd.getWidthsCW\(\) \(\text{pos, sig, gam, shl}\)
Compute the peak widths used for computing the range of a peak for constant wavelength data. On low-angle side, 50 FWHM are used, on high-angle side 75 are used, low angle side extended for axial divergence (for peaks above 90 deg, these are reversed.)

GSASIIpwd.getWidthsTOF\(\) \(\text{pos, alp, bet, sig, gam}\)
Compute the peak widths used for computing the range of a peak for constant wavelength data. 50 FWHM are used on both sides each extended by exponential coeff.

GSASIIpwd.getdEpsVoigt\(\) \(\text{pos, alp, bet, sig, gam, xdata}\)
needs a doc string

GSASIIpwd.getdFCJVoigt3\(\) \(\text{pos, sig, gam, shl, xdata}\)
Compute analytic derivatives the Finger-Cox-Jepcoat modified Pseudo-Voigt function for a CW powder peak

GSASIIpwd.getdPsVoigt\(\) \(\text{pos, sig, gam, xdata}\)
needs a doc string

GSASIIpwd.getgamFW\(\) \(\text{g, s}\)
Compute total FWHM from Thompson, Cox & Hastings (1987), J. Appl. Cryst. 20, 79-83 lambda fxn needs FWHM for both Gaussian & Lorentzian components

Parameters

\- g \quad \text{float} \quad \text{Lorentzian gamma = FWHM(L)}
\- s \quad \text{float} \quad \text{Gaussian sig}

Returns float total FWHM of pseudoVoigt

GSASIIpwd.makeFFTsizeList\(\) \(\text{nmin=1, nmax=1023, thresh=15}\)
Provide list of optimal data sizes for FFT calculations

Parameters

\- \text{nmin (int)} \quad \text{minimum data size >= 1}
\- \text{nmax (int)} \quad \text{maximum data size > nmin}
\- \text{thresh (int)} \quad \text{maximum prime factor allowed}

Returns list of data sizes where the maximum prime factor is < thresh

GSASIIpwd.makeMEMfile\(\) \(\text{data, reflData, MEMtype, DYSNOMIA}\)
make Dysnomia .mem file of reflection data, etc.

Parameters

\- \text{data (dict)} \quad \text{GSAS-II phase data}
\- \text{reflData (list)} \quad \text{GSAS-II reflection data}
\- \text{MEMtype (int)} \quad 1 \text{ for neutron data with negative scattering lengths 0 otherwise}
\- \text{DYSNOMIA (str)} \quad \text{path to dysnomia.exe}
makePRFfile(data, MEMtype)

makes Dysnomia .prf control file from Dysnomia GUI controls

Parameters

- data (dict) – GSAS-II phase data
- MEMtype (int) – 1 for neutron data with negative scattering lengths 0 otherwise

Returns str
name of Dysnomia control file

class GSASIIpwd.norm_gen(*args, **kwargs)

needs a doc string

1.14 GSAS-II Small Angle Scattering Submodules

1.14.1 GSASII small angle calculation module

CylinderARFF(Q, R, args)
Computes form factor for cylinders - can use numpy arrays param float Q: Q value array (A-1) param float R: cylinder radius (A) param array args: [float AR]: cylinder aspect ratio = L/D = L/2R returns float: form factor

CylinderARVol(R, args)
Computes cylinder volume for radius & aspect ratio = L/D - numpy array friendly param float: R radius (A) param array args: [float AR]: =L/D=L/2R aspect ratio returns float: volume

CylinderDFF(Q, L, args)
Computes form factor for cylinders - can use numpy arrays param float Q: Q value array (A-1) param float L: cylinder half length (A) param array args: [float R]: cylinder radius (A) returns float: form factor

CylinderDVol(L, args)
Computes cylinder volume for length & diameter - numpy array friendly param float: L half length (A) param array args: [float D]: diameter (A) returns float: volume (A^3)

CylinderFF(Q, R, args)
Computes form factor for cylinders - can use numpy arrays param float Q: Q value array (A-1) param float R: cylinder radius (A) param array args: [float L]: cylinder length (A) returns float: form factor

CylinderVol(R, args)
Computes cylinder volume for radius & length - numpy array friendly param float R: diameter (A) param array args: [float L]: length (A) returns float: volume (A^3)

DiluteSF(Q, args=[])
Default: no structure factor correction for dilute system

G_matrix(q, r, contrast, FFfxn, Volfxn, args=[])
Calculates the response matrix G(Q,r)

Parameters

- q (float) – Q
- r (float) – r
- contrast (float) – |Δρ|^2, the scattering contrast
- FFfxn (function) – form factor function FF(q,r,args)
- Volfxn (function) – volume function Vol(r,args)

Returns float G(Q,r)
GSASIIssadsn.GaussCume (x, pos, args)
Standard Normal cumulative distribution - numpy friendly on x axis param float x: independent axis (can be numpy array) param float pos: location of distribution param float scale: width of distribution (sigma) param float shape: not used returns float: Normal cumulative distribution

GSASIIssadsn.GaussDist (x, pos, args)
Standard Normal distribution - numpy friendly on x axis param float x: independent axis (can be numpy array) param float pos: location of distribution param float scale: width of distribution (sigma) param float shape: not used returns float: Normal distribution

GSASIIssadsn.HardSpheresSF (Q, args)
param float Q: Q value array (A-1) param array args: [float R, float VolFrac]: interparticle distance & volume fraction returns numpy array S(Q)

GSASIIssadsn.IPG (datum, sigma, G, Bins, Dbins, IterMax, Qvec=[], approach=0.8, Power=-1, report=False)

GSASIIssadsn.InterPrecipitateSF (Q, args)

GSASIIssadsn.LSWCume (x, pos, args=[])
Lifshitz-Slyozov-Wagner Ostwald ripening cumulative distribution - numpy friendly on x axis param float x: independent axis (can be numpy array) param float pos: location of distribution param float scale: not used returns float: LSW cumulative distribution

GSASIIssadsn.LSWDist (x, pos, args=[])
Lifshitz-Slyozov-Wagner Ostwald ripening distribution - numpy friendly on x axis ref: param float x: independent axis (can be numpy array) param float pos: location of distribution param float scale: not used returns float: LSW distribution

GSASIIssadsn.LogNormalCume (x, pos, args)
Standard LogNormal cumulative distribution - numpy friendly on x axis ref: http://www.itl.nist.gov/div898/handbook/index.htm 1.3.6.6.9 param float x: independent axis (can be numpy array) param float pos: location of distribution param float scale: width of distribution (sigma) param float shape: shape parameter returns float: LogNormal cumulative distribution

GSASIIssadsn.LogNormalDist (x, pos, args)
Standard LogNormal distribution - numpy friendly on x axis ref: http://www.itl.nist.gov/div898/handbook/index.htm 1.3.6.6.9 param float x: independent axis (can be numpy array) param float pos: location of distribution param float scale: width of distribution (m) param float shape: shape - (sigma of log(LogNormal)) returns float: LogNormal distribution

exception GSASIIssadsn.MaxEntException
Any exception from this module

GSASIIssadsn.MaxEnt_SB (datum, sigma, G, base, IterMax, image_to_data=None, data_to_image=None, report=False)
do the complete Maximum Entropy algorithm of Skilling and Bryan
Parameters

- `datum[] (float)` –
- `sigma[] (float)` –
- `G(float[][])` – transformation matrix
- `base[] (float)` –
- `IterMax (int)` –
- `image_to_data (obj)` – opus function (defaults to opus)
- `data_to_image (obj)` – tropus function (defaults to tropus)

Returns `float[] f(r)dr`

GSASIIsasd.SchulzZimmCume(x, pos, args)
Schulz-Zimm cumulative distribution - numpy friendly on x axis param float x: independent axis (can be numpy array) param float pos: location of distribution param float scale: width of distribution (sigma) param float shape: not used returns float: Normal distribution

GSASIIsasd.SchulzZimmDist(x, pos, args)
Schulz-Zimm macromolecule distribution - numpy friendly on x axis ref: http://goldbook.iupac.org/S05502.html param float x: independent axis (can be numpy array) param float pos: location of distribution param float scale: width of distribution (sigma) param float shape: not used returns float: Schulz-Zimm distribution

GSASIIsasd.SphereFF(Q, R, args=())
Compute hard sphere form factor - can use numpy arrays :param float Q: Q value array (usually in A-1) :param float R: sphere radius (Usually in A - must match Q-1 units) :param array args: ignored :returns: form factors as array as needed (float)

GSASIIsasd.SphereVol(R, args=())
Compute volume of sphere - numpy array friendly param float R: sphere radius param array args: ignored returns float: volume

GSASIIsasd.SphericalShellFF(Q, R, args=())
Compute spherical shell form factor - can use numpy arrays :param float Q: Q value array (usually in A-1) :param float R: sphere radius (Usually in A - must match Q-1 units) :param array args: [float r]: controls the shell thickness: R_inner = min(r*R,R), R_outer = max(r*R,R) :returns float: form factors as array as needed

Contributed by: L.A. Avakyan, Southern Federal University, Russia

GSASIIsasd.SphericalShellVol(R, args)
Compute volume of spherical shell - numpy array friendly param float R: sphere radius param array args: [float r]: controls shell thickness, see SphericalShellFF description returns float: volume

GSASIIsasd.SpheroidFF(Q, R, args)
Compute form factor of cylindrically symmetric ellipsoid (spheroid) - can use numpy arrays for R & AR; will return corresponding numpy array param float Q: Q value array (usually in A-1) param float R: radius along 2 axes of spheroid param array args: [float AR]: aspect ratio so 3rd axis = R*AR returns float: form factors as array as needed

GSASIIsasd.SpheroidVol(R, args)
Compute volume of cylindrically symmetric ellipsoid (spheroid) - numpy array friendly param float R: radius along 2 axes of spheroid param array args: [float AR]: aspect ratio so radius of 3rd axis = R*AR returns float: volume

GSASIIsasd.SquareWellSF(Q, args)
Computes structure factor for not dilute monodisperse hard sphere with a square well potential interaction. Refs.: SHARMA,SHARMA, PHYSICA 89A,(1977),213-
Parameters

- \(Q\) (float) – \(Q\) value array (A\(^{-1}\))
- \(args\) (array) – [float \(R\), float VolFrac, float depth, float width]: interparticle distance, volume fraction (<0.08), well depth (e/kT<1.5kT), well width

Returns

numpy array \(S(Q)\) well depth > 0 attractive & values outside above limits nonphysical cf. Monte Carlo simulations

\[
\text{GSASIIsasd. StickyHardSpheresSF}(Q, args)
\]

param float Q: \(Q\) value array (A\(^{-1}\)) param array args: [float \(R\), float VolFrac]: sphere radius & volume fraction returns numpy array \(S(Q)\)

\[
\text{GSASIIsasd. UniDiskFF}(Q, R, args)
\]
Compute form factor for unified disk - can use numpy arrays param float \(Q\) value array (A\(^{-1}\)) param float \(R\): cylinder radius (A) param array args: [float \(T\): disk thickness (A) returns float: form factor

\[
\text{GSASIIsasd. UniDiskVol}(R, args)
\]
Compute disk volume for radius & thickness - numpy array friendly param float \(R\): diameter (A) param array args: [float \(T\): thickness returns float:volume (A\(^{3}\))

\[
\text{GSASIIsasd. UniRodARFF}(Q, R, args)
\]
Compute form factor for unified rod of fixed aspect ratio - can use numpy arrays param float \(Q\) value array (A\(^{-1}\)) param float \(R\): cylinder radius (A) param array args: [float \(AR\): cylinder aspect ratio = L/D = L/2R returns float: form factor

\[
\text{GSASIIsasd. UniRodARVol}(R, args)
\]
Compute rod volume for radius & aspect ratio - numpy array friendly param float \(R\): diameter (A) param array args: [float \(AR\): =L/D=L/2R aspect ratio returns float:volume (A\(^{3}\))

\[
\text{GSASIIsasd. UniRodFF}(Q, R, args)
\]
Compute form factor for unified rod - can use numpy arrays param float \(Q\) value array (A\(^{-1}\)) param float \(R\): cylinder radius (A) param array args: [float \(R\): cylinder radius (A) returns float: form factor

\[
\text{GSASIIsasd. UniRodVol}(R, args)
\]
Compute cylinder volume for radius & length - numpy array friendly param float \(R\): diameter (A) param array args: [float \(L\): length (A) returns float:volume (A\(^{3}\))

\[
\text{GSASIIsasd. UniSphereFF}(Q, R, args=0)
\]
Compute form factor for unified sphere - can use numpy arrays param float \(Q\) value array (A\(^{-1}\)) param float \(R\): cylinder radius (A) param array args: ignored returns float: form factor

\[
\text{GSASIIsasd. UniSphereVol}(R, args=())
\]
Compute volume of sphere - numpy array friendly param float \(R\): sphere radius param array args: ignored returns float: volume

\[
\text{GSASIIsasd. UniTubeFF}(Q, R, args)
\]
Compute form factor for unified tube - can use numpy arrays assumes that core of tube is same as the matrix/solvent so contrast is from tube wall vs matrix param float \(Q\) value array (A\(^{-1}\)) param float \(R\): cylinder radius (A) param array args: [float \(L,T\): tube length & wall thickness(A) returns float: form factor

\[
\text{GSASIIsasd. UniTubeVol}(R, args)
\]
Compute tube volume for radius, length & wall thickness - numpy array friendly param float \(R\): diameter (A) param array args: [float \(L,T\): tube length & wall thickness(A) returns float: volume (A\(^{3}\)) of tube wall

\[
\text{GSASIIsasd. print_arr}(text, a)
\]
print the contents of an array to the console
print the contents of a vector to the console

1.14.2 Substances: Define Materials

Defines materials commonly found in small angle & reflectometry experiments. GSASII substances as a dictionary “Substances.Substances” with these materials.

Each entry in “Substances” consists of:

```
'key': {'Elements': {element: {'Num': float number in formula}, ...}, 'Density': value, 'Volume': value}
```

Density & Volume are optional, if one missing it is calculated from the other; if both are missing then Volume is estimated from composition & assuming 10Å^3 for each atom, Density is calculated from that Volume. See examples below for what is needed.

1.15 GSASIIscriptable: Scripting Interface

Routines to use an increasing amount of GSAS-II’s capabilities from scripts, without use of the graphical user interface (GUI). GSASIIscriptable can create and access GSAS-II project (.gpx) files and can directly perform image handling and refinements. The module defines wrapper classes (inheriting from \texttt{G2ObjectWrapper}) for a growing number of data tree items.

GSASIIscriptable can be used in two ways. It offers a command-line mode (see \textit{GSASIIscriptable Command-line Interface}) that provides access a number of features without writing Python scripts via shell/batch commands. The more powerful mode of GSASIIscriptable is use is through Python scripts that call the module’s application interface (API), see API summary that follows or the API: Complete Documentation section.

1.15.1 Application Interface (API) Summary

This section of the documentation provides an overview to API, with full documentation in the API: Complete Documentation section. The typical API use will be with a Python script, such as what is found in Code Examples. Most functionality is provided via the objects and methods summarized below.

Overview of Classes

<table>
<thead>
<tr>
<th>class</th>
<th>Encapsulates</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{G2Project}</td>
<td>a GSAS-II project file, provides references to objects below, each corresponding to a tree item (excepting \texttt{G2AtomRecord})</td>
</tr>
<tr>
<td>\texttt{G2Phase}</td>
<td>a phase</td>
</tr>
<tr>
<td>\texttt{G2PwdrData}</td>
<td>a powder histogram</td>
</tr>
<tr>
<td>\texttt{G2Image}</td>
<td>an image</td>
</tr>
<tr>
<td>\texttt{G2PDF}</td>
<td>a PDF histogram</td>
</tr>
<tr>
<td>\texttt{G2SeqRefRes}</td>
<td>the sequential results table</td>
</tr>
<tr>
<td>\texttt{G2AtomRecord}</td>
<td>an atom within a phase</td>
</tr>
</tbody>
</table>
Functions

A small amount of the Scriptable code does not require use of objects.

<table>
<thead>
<tr>
<th>method</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>GenerateReflections()</td>
<td>Generates a list of unique powder reflections</td>
</tr>
<tr>
<td>SetPrintLevel()</td>
<td>Sets the amount of output generated when running a script</td>
</tr>
</tbody>
</table>

Class G2Project

All GSASII scriptable scripts will need to create a G2Project object either for a new GSAS-II project or to read in an existing project (.gpx) file. The most commonly used routines in this object are:

<table>
<thead>
<tr>
<th>method</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>G2Project.save()</td>
<td>Writes the current project to disk.</td>
</tr>
<tr>
<td>G2Project.add_powder_histogram()</td>
<td>Used to read in powder diffraction data into a project file.</td>
</tr>
<tr>
<td>G2Project.add_simulated_powder_histogram()</td>
<td>Defines a &quot;dummy&quot; powder diffraction data that will be simulated after a refinement step.</td>
</tr>
<tr>
<td>G2Project.add_image()</td>
<td>Reads in an image into a project.</td>
</tr>
<tr>
<td>G2Project.add_phase()</td>
<td>Adds a phase to a project.</td>
</tr>
<tr>
<td>G2Project.add_PDF()</td>
<td>Adds a PDF entry to a project (does not compute it)</td>
</tr>
<tr>
<td>G2Project.histograms()</td>
<td>Provides a list of histograms in the current project, as G2PwdrData objects</td>
</tr>
<tr>
<td>G2Project.phases()</td>
<td>Provides a list of phases defined in the current project, as G2Phase objects</td>
</tr>
<tr>
<td>G2Project.images()</td>
<td>Provides a list of images in the current project, as G2Image objects</td>
</tr>
<tr>
<td>G2Project.pdfs()</td>
<td>Provides a list of PDFs in the current project, as G2PDF objects</td>
</tr>
<tr>
<td>G2Project.seqref()</td>
<td>Returns a G2SeqRefRes object if there are Sequential Refinement results</td>
</tr>
<tr>
<td>G2Project.do_refinements()</td>
<td>This is passed a list of dictionaries, where each dict defines a refinement step. Passing a list with a single empty dict initiates a refinement with the current parameters and flags. A refinement dict sets up a single refinement step (as described in Project-level Parameter Dict). Also see Refinement recipe.</td>
</tr>
<tr>
<td>G2Project.set_refinement()</td>
<td>This is passed a single dict which is used to set parameters and flags. These actions can be performed also in G2Project.do_refinements().</td>
</tr>
<tr>
<td>G2Project.get_Variable()</td>
<td>Retrieves the value and esd for a parameter</td>
</tr>
<tr>
<td>G2Project.get_Covariance()</td>
<td>Retrieves values and covariance for a set of refined parameters</td>
</tr>
<tr>
<td>G2Project.set_Controls()</td>
<td>Set overall GSAS-II control settings such as number of cycles and to set up a sequential fit. (Also see G2Project.get_Controls() to read values.)</td>
</tr>
<tr>
<td>G2Project.imageMultiDistCalib()</td>
<td>Performs a global calibration fit with images at multiple distance settings.</td>
</tr>
</tbody>
</table>

Class G2Phase

Another common object in GSASII scriptable scripts is G2Phase, used to encapsulate each phase in a project, with commonly used methods:
method & Use
---
G2Phase.set_refinements() & Provides a mechanism to set values and refinement flags for the phase. See Phase parameters for more details. This information also can be supplied within a call to G2Project.do_refinements() or G2Project.set_refinement().
G2Phase.clear_refinements() & Unsets refinement flags for the phase.
G2Phase.set_HAP_refinements() & Provides a mechanism to set values and refinement flags for parameters specific to both this phase and one of its histograms. See Histogram-and-phase parameters. This information also can be supplied within a call to G2Project.do_refinements() or G2Project.set_refinement().
G2Phase.clear_HAP_refinements() & Clears refinement flags specific to both this phase and one of its histograms.
G2Phase.getHAPvalues() & Returns values of parameters specific to both this phase and one of its histograms.
G2Phase.copyHAPvalues() & Copies HAP settings between from one phase/histogram and to other histograms in same phase.
G2Phase.atoms() & Returns a list of atoms in the phase
G2Phase.atom() & Returns an atom from its label
G2Phase.histograms() & Returns a list of histograms linked to the phase
G2Phase.get_cell() & Returns unit cell parameters (also see G2Phase.get_cell_and_esd()).
G2Phase.export_CIF() & Writes a CIF for the phase

Class G2PwdrData

Another common object in GSASIIscriptable scripts is G2PwdrData, which encapsulate each powder diffraction histogram in a project, with commonly used methods:

method & Use
---
G2PwdrData.set_refinements() & Provides a mechanism to set values and refinement flags for the powder histogram. See Histogram parameters for details.
G2PwdrData.clear_refinements() & Unsets refinement flags for the the powder histogram.
G2PwdrData.residuals() & Reports R-factors etc. for the the powder histogram (also see G2PwdrData.get_wR()).
G2PwdrData.add_back_peak() & Adds a background peak to the histogram. Also see G2PwdrData.del_back_peak() and G2PwdrData.ref_back_peak().
G2PwdrData.fit_fixed_points() & Fits background to the specified fixed points.
G2PwdrData.getdata() & Provides access to the diffraction data associated with the histogram.
G2PwdrData.reflections() & Provides access to the reflection lists for the histogram.
G2PwdrData.Export() & Writes the diffraction data or reflection list into a file
G2PwdrData.add_peak() & Adds a peak to the peak list. Also see Peak Fitting.
G2PwdrData.set_peakFlags() & Sets refinement flags for peaks
G2PwdrData.refine_peaks() & Starts a peak/background fitting cycle
G2PwdrData.Peaks & Provides access to the peak list data structure
G2PwdrData.PeakList & Provides the peak list parameter values
G2PwdrData.Export_peaks() & Writes the peak parameters to a text file
Class G2Image

When working with images, there will be a G2Image object for each image (also see G2Project. add_image() and G2Project.images()).

<table>
<thead>
<tr>
<th>method</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>G2Image.Recalibrate()</td>
<td>Invokes a recalibration fit starting from the current Image Controls calibration coefficients.</td>
</tr>
<tr>
<td>G2Image.Integrate()</td>
<td>Invokes an image integration. All parameters Image Controls will have previously been set.</td>
</tr>
<tr>
<td>G2Image.setControl()</td>
<td>Set an Image Controls parameter in the current image.</td>
</tr>
<tr>
<td>G2Image.getControl()</td>
<td>Return an Image Controls parameter in the current image.</td>
</tr>
<tr>
<td>G2Image.findControl()</td>
<td>Get the names of Image Controls parameters.</td>
</tr>
<tr>
<td>G2Image.loadControls()</td>
<td>Load controls from a .imctrl file (also see G2Image. saveControls()).</td>
</tr>
<tr>
<td>G2Image.loadMasks()</td>
<td>Load masks from a .immask file.</td>
</tr>
<tr>
<td>G2Image.setVary()</td>
<td>Set a refinement flag for Image Controls parameter in the current image. (Also see G2Image.getVary()).</td>
</tr>
<tr>
<td>G2Image.setCalibrate()</td>
<td>Set a calibrant type (or show choices) for the current image.</td>
</tr>
<tr>
<td>G2Image.setControlFile()</td>
<td>Set a image to be used as a background/dark/gain map image.</td>
</tr>
</tbody>
</table>

Class G2PDF

To work with PDF entries, object G2PDF, encapsulates a PDF entry with methods:

<table>
<thead>
<tr>
<th>method</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>G2PDF.export()</td>
<td>Used to write G(r), etc. as a file</td>
</tr>
<tr>
<td>G2PDF.calculate()</td>
<td>Computes the PDF using parameters in the object</td>
</tr>
<tr>
<td>G2PDF.optimize()</td>
<td>Optimizes selected PDF parameters</td>
</tr>
<tr>
<td>G2PDF.set_background()</td>
<td>Sets the histograms used for sample background, container, etc.</td>
</tr>
<tr>
<td>G2PDF.set_formula()</td>
<td>Sets the chemical formula for the sample</td>
</tr>
</tbody>
</table>

Class G2SeqRefRes

To work with Sequential Refinement results, object G2SeqRefRes, encapsulates the sequential refinement table with methods:

<table>
<thead>
<tr>
<th>method</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>G2SeqRefRes.histograms()</td>
<td>Provides a list of histograms used in the Sequential Refinement</td>
</tr>
<tr>
<td>G2SeqRefRes.get_cell_and_esd()</td>
<td>Returns cell dimensions and standard uncertainties for a phase and histogram from the Sequential Refinement</td>
</tr>
<tr>
<td>G2SeqRefRes.get_Variable()</td>
<td>Retrieves the value and esd for a parameter from a particular histogram in the Sequential Refinement</td>
</tr>
<tr>
<td>G2SeqRefRes.get_Covariance()</td>
<td>Retrieves values and covariance for a set of refined parameters for a particular histogram</td>
</tr>
</tbody>
</table>

Class G2AtomRecord

When working with phases, G2AtomRecord objects provide access to the contents of each atom in a phase. This provides access to “properties” that can be used to get values of much of the atoms associated
settings: label, type, refinement_flags, coordinates, occupancy, ranId, adp_flag, and uiso. In addition, refinement_flags, occupancy and uiso can be used to set values. See the G2AtomRecord docs and source code.

1.15.2 Refinement parameters

While scripts can be written that setup refinements by changing individual parameters through calls to the methods associated with objects that wrap each data tree item, many of these actions can be combined into fairly complex dict structures to conduct refinement steps. Use of these dicts is required with the GSASIIscriptable Command-line Interface. This section of the documentation describes these dicts.

Project-level Parameter Dict

As noted below (Refinement parameter types), there are three types of refinement parameters, which can be accessed individually by the objects that encapsulate individual phases and histograms but it will often be simplest to create a composite dictionary that is used at the project-level. A dict is created with keys “set” and “clear” that can be supplied to G2Project.set_refinement() (or G2Project.do_refinements(), see Refinement recipe below) that will determine parameter values and will determine which parameters will be refined.

The specific keys and subkeys that can be used are defined in tables Histogram parameters, Phase parameters and Histogram-and-phase parameters.

Note that optionally a list of histograms and/or phases can be supplied in the call to G2Project.set_refinement(), but if not specified, the default is to use all defined phases and histograms.

As an example:

```python
pardict = {'set': { 'Limits': [0.8, 12.0],
       'Sample Parameters': ['Absorption', 'Contrast', 'DisplaceX'],
       'Background': {'type': 'chebyschev-1', 'refine': True,
                      'peaks':[[0,True],[1,1,1]] },
       'clear': {'Instrument Parameters': ['U', 'V', 'W']}}
my_project.set_refinement(pardict)
```

Refinement recipe

Building on the Project-level Parameter Dict, it is possible to specify a sequence of refinement actions as a list of these dicts and supplying this list as an argument to G2Project.do_refinements().

As an example, this code performs the same actions as in the example in the section above:

```python
pardict = {'set': { 'Limits': [0.8, 12.0],
       'Sample Parameters': ['Absorption', 'Contrast', 'DisplaceX'],
       'Background': {'type': 'chebyschev-1', 'refine': True}},
       'clear': {'Instrument Parameters': ['U', 'V', 'W']}}
my_project.do_refinements([pardict])
```

However, in addition to setting a number of parameters, this example will perform a refinement as well, after setting the parameters. More than one refinement can be performed by including more than one dict in the list.

In this example, two refinement steps will be performed:

```python
my_project.do_refinements([pardict,pardict1])
```
The keys defined in the following table may be used in a dict supplied to `G2Project.do_refinements()`. Note that keys `histograms` and `phases` are used to limit actions to specific sets of parameters within the project.

<table>
<thead>
<tr>
<th>key</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>set</td>
<td>Specifies a dict with keys and subkeys as described in the <em>Specifying Refinement Parameters</em> section. Items listed here will be set to be refined.</td>
</tr>
<tr>
<td>clear</td>
<td>Specifies a dict, as above for set, except that parameters are cleared and thus will not be refined.</td>
</tr>
<tr>
<td>once</td>
<td>Specifies a dict as above for set, except that parameters are set for the next cycle of refinement and are cleared once the refinement step is completed.</td>
</tr>
<tr>
<td>skip</td>
<td>Normally, once parameters are processed with a set/clear/once action(s), a refinement is started. If skip is defined as True (or any other value) the refinement step is not performed.</td>
</tr>
<tr>
<td>output</td>
<td>If a file name is specified for output is will be used to save the current refinement.</td>
</tr>
<tr>
<td>histograms</td>
<td>Should contain a list of histogram(s) to be used for the set/clear/once action(s) on <em>Histogram parameters</em> or <em>Histogram-and-phase parameters</em>. Note that this will be ignored for <em>Phase parameters</em>. Histograms may be specified as a list of strings [('PWDR ...'),...], indices [0,1,2] or as list of objects [hist1, hist2].</td>
</tr>
<tr>
<td>phases</td>
<td>Should contain a list of phase(s) to be used for the set/clear/once action(s) on <em>Phase parameters</em> or <em>Histogram-and-phase parameters</em>. Note that this will be ignored for <em>Histogram parameters</em>. Phases may be specified as a list of strings [('Phase name'),...], indices [0,1,2] or as list of objects [phase0, phase2].</td>
</tr>
<tr>
<td>call</td>
<td>Specifies a function to call after a refinement is completed. The value supplied can be the object (typically a function) that will be called or a string that will evaluate (in the namespace inside <code>G2Project.iter_refinements()</code> where <code>self</code> references the project.) Nothing is called if this is not specified.</td>
</tr>
<tr>
<td>callargs</td>
<td>Provides a list of arguments that will be passed to the function in call (if any). If call is defined and callargs is not, the current <code>&lt;tt&gt;G2Project&lt;/tt&gt;</code> is passed as a single argument.</td>
</tr>
</tbody>
</table>

An example that performs a series of refinement steps follows:

```python
reflist = [
    {"set": { "Limits": { "low": 0.7 },
      "Background": { "no. coeffs": 3,
        "refine": True }},
    {"set": { "LeBail": True,
      "Cell": True }},
    {"set": { "Sample Parameters": ["DisplaceX"]}},
    {"set": { "Instrument Parameters": ["U", "V", "W", "X", "Y"]}},
    {"set": { "Mustrain": { "type": "uniaxial",
        "refine": "equatorial",
        "direction": [0, 0, 1]}}},
    {"set": { "Mustrain": { "type": "uniaxial",
        "refine": "axial"}}},
    {"clear": { "LeBail": True},
    {"set": { "Atoms": { "Mn": "X" }}},
    {"set": { "Atoms": { "O1": "X", "O2": "X" }}}}]
my_project.do_refinements(reflist)
```

In this example, a separate refinement step will be performed for each dict in the list. The keyword “skip” can be used to specify a dict that should not include a refinement. Note that in the second from last refinement step, parameters are both set and cleared.

### Refinement parameter types

Note that parameters and refinement flags used in GSAS-II fall into three classes:

- **Histogram**: There will be a set of these for each dataset loaded into a project file. The parameters available depend on the type of histogram (Bragg-Brentano, Single-Crystal, TOF, ...). Typical Histogram parameters
include the overall scale factor, background, instrument and sample parameters; see the Histogram parameters table for a list of the histogram parameters where access has been provided.

- **Phase**: There will be a set of these for each phase loaded into a project file. While some parameters are found in all types of phases, others are only found in certain types (modulated, magnetic, protein...). Typical phase parameters include unit cell lengths and atomic positions; see the Phase parameters table for a list of the phase parameters where access has been provided.

- **Histogram-and-phase (HAP)**: There is a set of these for every histogram that is associated with each phase, so that if there are $N$ phases and $M$ histograms, there can be $N \times M$ total sets of “HAP” parameters sets (fewer if all histograms are not linked to all phases.) Typical HAP parameters include the phase fractions, sample microstrain and crystallite size broadening terms, hydrostatic strain perturbations of the unit cell and preferred orientation values. See the Histogram-and-phase parameters table for the HAP parameters where access has been provided.

### 1.15.3 Specifying Refinement Parameters

Refinement parameter values and flags to turn refinement on and off are specified within dictionaries, where the details of these dicts are organized depends on the type of parameter (see Refinement parameter types), with a different set of keys (as described below) for each of the three types of parameters.

**Histogram parameters**

This table describes the dictionaries supplied to `G2PwdrData.set_refinements()` and `G2PwdrData.clear_refinements()`. As an example,

```python
hist.set_refinements({'Background': {'no.coeffs': 3, 'refine': True},
                      'Sample Parameters': ['Scale'],
                      'Limits': [10000, 40000])
```

With `G2Project.do_refinements()`, these parameters should be placed inside a dict with a key `set`, `clear`, or `once`. Values will be set for all histograms, unless the `histograms` key is used to define specific histograms. As an example:

```python
gsas_proj.do_refinements(
    {'set': {
        'Background': {'no.coeffs': 3, 'refine': True},
        'Sample Parameters': ['Scale'],
        'Limits': [10000, 40000],
        'histograms': [1, 2]}
    )
```

Note that below in the Instrument Parameters section, related profile parameters (such as U and V) are grouped together but separated by commas to save space in the table.
<table>
<thead>
<tr>
<th>key</th>
<th>subkey</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limits</td>
<td></td>
<td>The range of 2-theta (degrees) or TOF (in microsec) range of values to use. Can be either a dictionary of ‘low’ and/or ‘high’, or a list of 2 items [low, high]</td>
</tr>
<tr>
<td></td>
<td>low</td>
<td>Sets the low limit</td>
</tr>
<tr>
<td></td>
<td>high</td>
<td>Sets the high limit</td>
</tr>
<tr>
<td>Sample Parameters</td>
<td></td>
<td>Should be provided as a list of subkeys to set or clear, e.g. ['DisplaceX', 'Scale']</td>
</tr>
<tr>
<td>Absorption</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contrast</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DisplaceX</td>
<td></td>
<td>Sample displacement along the X direction</td>
</tr>
<tr>
<td>DisplaceY</td>
<td></td>
<td>Sample displacement along the Y direction</td>
</tr>
<tr>
<td>Scale</td>
<td></td>
<td>Histogram Scale factor</td>
</tr>
<tr>
<td>Background</td>
<td></td>
<td>Sample background. Value will be a dict or a boolean. If True or False, the refine parameter for background is set to that. Note that background peaks are not handled via this; see G2PwdrData.ref_back_peak() instead. When value is a dict, supply any of the following keys:</td>
</tr>
<tr>
<td></td>
<td>type</td>
<td>The background model, e.g. ‘chebyschev-1’</td>
</tr>
<tr>
<td></td>
<td>refine</td>
<td>The value of the refine flag, boolean</td>
</tr>
<tr>
<td></td>
<td>‘no. coeffs’</td>
<td>Number of coefficients to use, integer</td>
</tr>
<tr>
<td></td>
<td>coeffs</td>
<td>List of floats, literal values for background</td>
</tr>
<tr>
<td>FixedPoints</td>
<td></td>
<td>List of (2-theta, intensity) values for fixed points</td>
</tr>
<tr>
<td></td>
<td>‘fit fixed points’</td>
<td>If True, triggers a fit to the fixed points to be calculated. It is calculated when this key is detected, regardless of calls to refine.</td>
</tr>
<tr>
<td></td>
<td>peaks</td>
<td>Specifies a set of flags for refining background peaks as a nested list. There may be an item for each defined background peak (or fewer) and each item is a list with the flag values for pos, int, sig &amp; gam (fewer than 4 values are allowed).</td>
</tr>
<tr>
<td>Instrument Parameters</td>
<td></td>
<td>As in Sample Parameters, provide as a list of subkeys to set or clear, e.g. ['X', 'Y', 'Zero', 'SH/L']</td>
</tr>
<tr>
<td></td>
<td>U, V, W</td>
<td>Gaussian peak profile terms</td>
</tr>
<tr>
<td></td>
<td>X, Y, Z</td>
<td>Lorentzian peak profile terms</td>
</tr>
<tr>
<td></td>
<td>alpha, beta-0, beta-1, beta-q,</td>
<td>TOF profile terms</td>
</tr>
<tr>
<td></td>
<td>sig-0, sig-1, sig-2, sig-q</td>
<td>TOF profile terms</td>
</tr>
<tr>
<td></td>
<td>difA, difB, difC</td>
<td>TOF Calibration constants</td>
</tr>
<tr>
<td></td>
<td>Zero</td>
<td>Zero shift</td>
</tr>
<tr>
<td></td>
<td>SH/L</td>
<td>Finger-Cox-Jephcoat low-angle peak asymmetry</td>
</tr>
<tr>
<td></td>
<td>Polariz.</td>
<td>Polarization parameter</td>
</tr>
<tr>
<td></td>
<td>Lam</td>
<td>Lambda, the incident wavelength</td>
</tr>
</tbody>
</table>

**Phase parameters**

This table describes the dictionaries supplied to G2Phase.set_refinements() and G2Phase.clear_refinements(). With G2Project.do_refinements(), these parameters should be placed inside a dict with a key set, clear, or once. Values will be set for all phases, unless the phases key is used to define specific phase(s).
<table>
<thead>
<tr>
<th>key</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell</td>
<td>Whether or not to refine the unit cell.</td>
</tr>
<tr>
<td>Atoms</td>
<td>Dictionary of atoms and refinement flags. Each key should be an atom label, e.g. ‘O3’, ‘Mn5’, and each value should be a string defining what values to refine. Values can be any combination of ‘F’ for fractional occupancy, ‘X’ for position, and ‘U’ for Debye-Waller factor</td>
</tr>
<tr>
<td>LeBail</td>
<td>Enables LeBail intensity extraction.</td>
</tr>
</tbody>
</table>

**Histogram-and-phase parameters**

This table describes the dictionaries supplied to `G2Phase.set_HAP_refinements()` and `G2Phase.clear_HAP_refinements()`. When supplied to `G2Project.do_refinements()`, these parameters should be placed inside a dict with a key `set`, `clear`, or `once`. Values will be set for all histograms used in each phase, unless the `histograms` and `phases` keys are used to define specific phases and histograms.

<table>
<thead>
<tr>
<th>key</th>
<th>subkey</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Babinet</td>
<td></td>
<td>Should be a list of the following subkeys. If not, assumes both BabA and BabU</td>
</tr>
<tr>
<td>BabA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BabU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extinction</td>
<td></td>
<td>Boolean, True to refine.</td>
</tr>
<tr>
<td>HStrain</td>
<td></td>
<td>Boolean or list/tuple, True to refine all appropriate D&lt;sub&gt;ij&lt;/sub&gt; terms or False to not refine any. If a list/tuple, will be a set of True &amp; False values for each D&lt;sub&gt;ij&lt;/sub&gt; term; number of items must match number of terms.</td>
</tr>
<tr>
<td>Mustrain</td>
<td>type</td>
<td>Mustrain model. One of ‘isotropic’, ‘uniaxial’, or ‘generalized’. Should always be included when Mustrain is used.</td>
</tr>
<tr>
<td></td>
<td>direction</td>
<td>For uniaxial only. A list of three integers, the [hkl] direction of the axis.</td>
</tr>
<tr>
<td></td>
<td>refine</td>
<td>Usually boolean, set to True to refine. or False to clear. For uniaxial model, can specify a value of ‘axial’ or ‘equatorial’ to set that flag to True or a single boolean sets both axial and equatorial.</td>
</tr>
<tr>
<td>Size</td>
<td>type</td>
<td>Size broadening model. One of ‘isotropic’, ‘uniaxial’, or ‘ellipsoid’. Should always be specified when Size is used.</td>
</tr>
<tr>
<td></td>
<td>direction</td>
<td>For uniaxial only. A list of three integers, the [hkl] direction of the axis.</td>
</tr>
<tr>
<td></td>
<td>refine</td>
<td>Boolean, True to refine.</td>
</tr>
<tr>
<td></td>
<td>value</td>
<td>float, size value in microns</td>
</tr>
<tr>
<td>Pref.Ori.</td>
<td></td>
<td>Boolean, True to refine</td>
</tr>
<tr>
<td>Show</td>
<td></td>
<td>Boolean, True to refine</td>
</tr>
<tr>
<td>Use</td>
<td></td>
<td>Boolean, True to refine</td>
</tr>
<tr>
<td>Scale</td>
<td></td>
<td>Phase fraction; Boolean, True to refine</td>
</tr>
</tbody>
</table>

**Histogram/Phase objects**

Each phase and powder histogram in a `G2Project` object has an associated object. Parameters within each individual object can be turned on and off by calling `G2PwdrData.set_refinements()` or `G2PwdrData.clear_refinements()`.
G2PwdrData.clear_refinements() for histogram parameters; G2Phase.set_refinements() or G2Phase.clear_refinements() for phase parameters; and G2Phase.set_HAP_refinements() or G2Phase.clear_HAP_refinements(). As an example, if some_histogram is a histogram object (of type G2PwdrData), use this to set parameters in that histogram:

```python
params = { 'Limits': [0.8, 12.0],
          'Sample Parameters': ['Absorption', 'Contrast', 'DisplaceX'],
          'Background': { 'type': 'chebyschev-1', 'refine': True}}
some_histogram.set_refinements(params)
```

Likewise to turn refinement flags on, use code such as this:

```python
params = { 'Instrument Parameters': ['U', 'V', 'W']}
some_histogram.set_refinements(params)
```

and to turn these refinement flags, off use this (Note that the .clear_refinements() methods will usually will turn off refinement even if a refinement parameter is set in the dict to True.):

```python
params = { 'Instrument Parameters': ['U', 'V', 'W']}
some_histogram.clear_refinements(params)
```

For phase parameters, use code such as this:

```python
params = { 'LeBail': True, 'Cell': True,
          'Atoms': { 'Mn1': 'X',
                     'O3': 'XU',
                     'V4': 'FXU'}}
some_histogram.set_refinements(params)
```

and here is an example for HAP parameters:

```python
params = { 'Babinet': 'BabA',
          'Extinction': True,
          'Mustrain': { 'type': 'uniaxial',
                        'direction': [0, 0, 1],
                        'refine': True}}
some_phase.set_HAP_refinements(params)
```

Note that the parameters must match the object type and method (phase vs. histogram vs. HAP).

### 1.15.4 Code Examples

#### Peak Fitting

Peak refinement is performed with routines `G2PwdrData.add_peak()`, `G2PwdrData.set_peakFlags()` and `G2PwdrData.refine_peaks()`. Method `G2PwdrData.Export_peaks()` and properties `G2PwdrData.Peaks` and `G2PwdrData.PeakList` provide ways to access the results. Note that when peak parameters are refined with `refine_peaks()`, the background may also be refined. Use `G2PwdrData.set_refinements()` to change background settings and the range of data used in the fit. See below for an example peak refinement script, where the data files are taken from the “Rietveld refinement with CuKa lab Bragg-Brentano powder data” tutorial (in https://subversion.xray.aps.anl.gov/pyGSAS/Tutorials/LabData/data/).

```python
from __future__ import division, print_function
import os, sys
sys.path.insert(0,'/Users/toby/software/G2/GSASII')  # needed to "find" GSAS-II modules
```

(continues on next page)
import GSASIIscriptable as G2sc

datadir = os.path.expanduser("~/Scratch/peakfit")
PathWrap = lambda fil: os.path.join(datadir,fil)
gpx = G2sc.G2Project(newgpx=PathWrap('pkfit.gpx'))

hist = gpx.add_powder_histogram(PathWrap('FAP.XRA'), PathWrap('INST_XRY.PRM'),
    fmthint='GSAS powder')

hist.set_refinements({'Limits': [16.,24.],
    'Background': {'no. coeffs': 2,'type': 'chebyschev-1', 'refine': True}})

peak1 = hist.add_peak(1, ttheta=16.8)
peak2 = hist.add_peak(1, ttheta=18.9)
peak3 = hist.add_peak(1, ttheta=21.8)
peak4 = hist.add_peak(1, ttheta=22.9)

hist.set_peakFlags(area=True)

hist.refine_peaks()

hist.set_peakFlags(area=True, pos=True)

hist.refine_peaks()

hist.set_peakFlags(area=True, pos=True, sig=True, gam=True)

hist.refine_peaks()

print ('peak positions: ',[i[0] for i in hist.PeakList])
for i in range(len(hist.Peaks['peaks'])):
    print ('peak',i,'pos=',hist.Peaks['peaks'][i][0],'sig=',hist.Peaks['sigDict'][pos -'+str(i)])

hist.Export_peaks('pkfit.txt')

#gpx.save() # gpx file is not written without this

Pattern Simulation

This shows an example where a structure is read from a CIF, a pattern is computed using a instrument parameter file to specify the probe type (neutrons here) and wavelength. The pattern and reflection list are computed. Data files are found in the Scripting Tutorial.

import os, sys
sys.path.insert(0,'/Users/toby/software/G2/GSASII')
import GSASIIscriptable as G2sc

datadir = "'/Users/toby/software/G2/Tutorials/PythonScript/data"
PathWrap = lambda fil: os.path.join(datadir,fil)
gpx = G2sc.G2Project(filename='PbSO4sim.gpx') # create a project

# add a phase to the project

phase0 = gpx.add_phase(PathWrap("PbSO4-Wyckoff.cif"),
    phasename="PbSO4",fmthint='CIF')

# add a simulated histogram and link it to the previous phase(s)

hist1 = gpx.add_simulated_powder_histogram("PbSO4 simulation",
    PathWrap("inst_d1a.prm"),5.,120.,0.01,
    phases=gpx.phases())

# Set the scale factor to adjust the y scale

hist1.SampleParameters['Scale'][0] = 1000000.

# parameter optimization and calculate pattern

gpx.data['Controls']['Scale'][0] = 1000000.

# refinement not needed

gpx.do_refinements({})

gpx.save()

# save results

gpx.histogram(0).Export("PbSO4data','.csv','hist') # data

gpx.histogram(0).Export('PbSO4refl','.csv','refl') # reflections
Simple Refinement

GSASIIscriptable can be used to setup and perform simple refinements. This example reads in an existing project (.gpx) file, adds a background peak, changes some refinement flags and performs a refinement.

```python
from __future__ import division, print_function
import os, sys
sys.path.insert(0, '/Users/toby/software/G2/GSASII')  # needed to "find" GSAS-II modules
import GSASIIscriptable as G2sc

datadir = '/Users/Scratch/

gpx = G2sc.G2Project(os.path.join(datadir,'test2.gpx'))
gpx.histogram(0).add_back_peak(4.5,30000,5000,0)
pardict = {'set': {'Sample Parameters': ['Absorption', 'Contrast', 'DisplaceX'],
                  'Background': {'type': 'chebyschev-1', 'refine': True,
                                 'peaks':[[0,True]]}}}

gpx.set_refinement(pardict)
```

Sequential Refinement

GSASIIscriptable can be used to setup and perform sequential refinements. This example script is used to take the single-dataset fit at the end of Step 1 of the Sequential Refinement tutorial and turn on and off refinement flags, add histograms and setup the sequential fit, which is then run:

```python
import os, sys, glob
sys.path.insert(0, '/Users/toby/software/G2/GSASII')
import GSASIIscriptable as G2sc

datadir = os.path.expanduser('~/Scratch/SeqTut2019Mar')
PathWrap = lambda fil: os.path.join(datadir,fil)

# load and rename project

gpx = G2sc.G2Project(PathWrap('7Konly.gpx'))
gpx.save(PathWrap('SeqRef.gpx'))

# turn off some variables; turn on Dijs
for p in gpx.phases():
    p.set_refinements({'Cell': False})
gpx.phase(0).set_HAP_refinements(
    {'Scale': False,
     'Size': {'type': 'isotropic', 'refine': False},
     'Mustrain': {'type': 'uniaxial', 'refine': False},
     'HStrain': True})
gpx.phase(1).set_HAP_refinements({'HStrain': (1,1,1,0)})
gpx.histogram(0).clear_refinements({'Background':False,
                                      'Sample Parameters': ['DisplaceX']})
gpx.histogram(0).ref_back_peak(0,[])

gpx.phase(1).set_HAP_refinements({'HStrain': (1,1,1,0)})

for fil in sorted(glob.glob(PathWrap('*.fxye'))):  # load in remaining fxye files
    if '00' in fil: continue
    gpx.add_powder_histogram(fil, PathWrap('OH_00.prm'), fmthint="GSAS powder", phases='all')

# copy HAP values, background, instrument params. & limits, not sample params.
gpx.copyHistParms(0,'all', ['b','i','l'])

for p in gpx.phases(): p.copyHAPvalues(0,'all')

# setup and launch sequential fit

gpx.set_Controls('sequential', gpx.histograms())
gpx.set_Controls('cycles', 10)
gpx.set_Controls('seqCopy', True)
gpx.refine()
```
Image Processing

A sample script where an image is read, assigned calibration values from a file and then integrated follows. The data files are found in the Scripting Tutorial.

```python
import os, sys
sys.path.insert(0, '/Users/toby/software/G2/GSASII')
import GSASIIscriptable as G2sc

datadir = '/tmp'
PathWrap = lambda fil: os.path.join(datadir, fil)

gpx = G2sc.G2Project(filename=PathWrap('inttest.gpx'))
imlst = gpx.add_image(PathWrap('Si_free_dc800_1-00000.tif'), fmthint='TIF')
imlst[0].loadControls(PathWrap('Si_free_dc800_1-00000.imctrl'))
pwdrList = imlst[0].Integrate()
gpx.save()

This example shows a computation similar to what is done in tutorial Area Detector Calibration with Multiple Distances.

```python
import os, sys, glob
sys.path.insert(0, '/Users/toby/software/G2/GSASII')
import GSASIIscriptable as G2sc
PathWrap = lambda fil: os.path.join('/Users/toby/wp/Active/MultidistanceCalibration/multimg', fil)

gpx = G2sc.G2Project(filename='/tmp/img.gpx')
for f in glob.glob(PathWrap('*.tif')):  
im = gpx.add_image(f, fmthint='TIF')
# image parameter settings
defImgVals = {'wavelength': 0.24152, 'center': [206., 205.],  
             'pixLimit': 2, 'cutoff': 5.0, 'DetDepth': 0.055, 'calibdmin': 1.}
# set controls and vary options, then fit
for img in gpx.images():  
im.setCalibrant('Si SRM640c')
im.setVary('*', False)  
im.setVary(['det-X', 'det-Y', 'phi', 'tilt', 'wave'], True)  
im.setControls(defImgVals)  
im.Recalibrate()  
im.Recalibrate()  
# 2nd run better insures convergence

gpx.save()  
# make dict of images for sorting
images = {img.getControl('setdist'): img for img in gpx.images()}
# show values
for key in sorted(images.keys()):  
im = images[key]  
c = im.getControls()  
print(c['distance'], c['wavelength'])
```

Image Calibration

This example performs a number of cycles of constrained fitting. A project is created with the images found in a directory, setting initial parameters as the images are read. The initial values for the calibration are not very good, so a G2Image.Recalibrate() is done to quickly improve the fit. Once that is done, a fit of all images is performed where the wavelength, an offset and detector orientation are constrained to be the same for all images. The detector
penetration correction is then added. Note that as the calibration values improve, the algorithm is able to find more points on diffraction rings to use for calibration and the number of “ring picks” increase. The calibration is repeated until that stops increasing significantly (<10%). Detector control files are then created. The files used for this exercise are found in the Area Detector Calibration Tutorial (see Area Detector Calibration with Multiple Distances).

```python
import os, sys, glob
sys.path.insert(0, '/Users/toby/software/G2/GSASII')
import GSASIIscriptable as G2sc
PathWrap = lambda fil: os.path.join(
    '/Users/toby/wp/Active/MultidistanceCalibration/multimg',
    fil)

gpx = G2sc.G2Project(filename='/tmp/calib.gpx')
for f in glob.glob(PathWrap('*.tif')):
    im = gpx.add_image(f, fmtHint='TIF')
# starting image parameter settings
defImgVals = {'wavelength': 0.240, 'center': [206., 205.],
              'pixLimit': 2, 'cutoff': 5.0, 'DetDepth': 0.03, 'calibdmin': 0.5,}
# set controls and vary options, then initial fit
for img in gpx.images():
    img.setCalibrant('Si SRM640c')
    img.setVary('X', False)
    img.setVary(['det-X', 'det-Y', 'phi', 'tilt', 'wave'], True)
    img.setControls(defImgVals)
    if img.getControl('setdist') > 900:
        img.setControls({'calibdmin': 1.,})
    img.Recalibrate()
G2sc.SetPrintLevel('warn') # cut down on output
result, covData = gpx.imageMultiDistCalib()
print('1st global fit: initial ring picks', covData['obs'])
print({i: result[i] for i in result if '-' not in i})
# add parameter to all images & refit multiple times
for img in gpx.images():
    img.setVary('dep', True)
ringpicks = covData['obs']
delta = ringpicks
while delta > ringpicks/10:
    result, covData = gpx.imageMultiDistCalib(verbosе=False)
    delta = covData['obs'] - ringpicks
    print('ring picks went from', ringpicks, 'to', covData['obs'])
    print({i: result[i] for i in result if '-' not in i})
    ringpicks = covData['obs']
# once more for good measure & printout
result, covData = gpx.imageMultiDistCalib(verbosе=True)
# create image control files
for img in gpx.images():
    img.saveControls(os.path.splitext(img.name)[0] + '.imctrl')
gpx.save()
```

1.15.5 GSASIIscriptable Command-line Interface

The routines described above are intended to be called from a Python script, but an alternate way to access some of the same functionality is to invoke the GSASIIscriptable.py script from the command line usually from within a shell script or batch file. This will usually be done with a command such as:

```bash
python <path/>GSASIIscriptable.py <subcommand> <file.gpx> <options>
```

The following subcommands are defined:
• create, see create()
• add, see add()
• dump, see dump()
• refine, see refine()
• export, export()
• browse, see IPyBrowse()

Run:

```
python GSASIIscriptable.py --help
```

to show the available subcommands, and inspect each subcommand with `python GSASIIscriptable.py <subcommand> --help` or see the documentation for each of the above routines.

### Parameters in JSON files

The refine command requires two inputs: an existing GSAS-II project (.gpx) file and a JSON format file (see Introducing JSON) that contains a single dict. This dict may have two keys:

- **refinements**: This defines the a set of refinement steps in a JSON representation of a Refinement recipe list.
- **code**: This optionally defines Python code that will be executed after the project is loaded, but before the refinement is started. This can be used to execute Python code to change parameters that are not accessible via a Refinement recipe dict (note that the project object is accessed with variable `proj`) or to define code that will be called later (see key `call` in the Refinement recipe section.)

JSON website: Introducing JSON.

### 1.15.6 API: Complete Documentation

The classes and modules in this module are described below. A script will create one or more `G2Project` objects by reading a GSAS-II project (.gpx) file or creating a new one and will then perform actions such as adding a histogram (method `G2Project.add_powder_histogram()`), adding a phase (method `G2Project.add_phase()`), or setting parameters and performing a refinement (method `G2Project.do_refinements()`).

To change settings within histograms, images and phases, one usually needs to use methods inside `G2PwdrData`, `G2Image` or `G2Phase`.

**class GSASIIscriptable.G2AtomRecord(data, indices, proj)**

Wrapper for an atom record. Has convenient accessors via @property: label, type, refinement_flags, coordinates, occupancy, ranId, id, adp_flag, uiso

Example:

```
>>> atom = some_phase.atom("O3")
>>> # We can access the underlying data format
>>> atom.data
['O3', 'O-2', '', ... ]
>>> # We can also use wrapper accessors
>>> atom.coordinates
(0.33, 0.15, 0.5)
>>> atom.refinement_flags
u'FX'
>>> atom.ranId
```

(continues on next page)
atom.occupancy
1.0

adp_flag
Get the associated atom’s iso/aniso setting, ‘I’ or ‘A’

coordinates
Get the associated atom’s coordinates

element
Get the associated atom’s element symbol

class GSASIIscriptable.G2Image (data, name, proj)

Wrapper for an IMG tree entry, containing an image and associated metadata.

Note that in a GSASIIscriptable script, instances of G2Image will be created by calls to G2Project.add_image() or G2Project.images(). Scripts will not use G2Image() to call G2Image.__init__() directly. The object contains these class variables:

• G2Image.proj: contains a reference to the G2Project object that contains this image
• G2Image.name: contains the name of the image
• G2Image.data: contains the image’s associated data in a dict, as documented for the Image Data Structure.

Example use of G2Image:

```python
gpx = G2sc.G2Project(filename='itest.gpx')
imlst = gpx.add_image(idata,fmthint="TIF")
imlst[0].loadControls('stdSettings.imctrl')
imlst[0].setCalibrant('Si SRM640c')
imlst[0].loadMasks('stdMasks.immask')
imlst[0].Recalibrate()
imlst[0].setControl('outAzimuths',3)
pwdrList = imlst[0].Integrate()
```

More detailed image processing examples are shown at Image Processing.

ControlList = {'bool': ['setRings', 'setDefault', 'centerAzm', 'fullIntegrate', 'DetDepthRef', 'showLines'], 'dict': ... 'pixelSize', 'range', 'ring', 'rings', 'size'], 'str': ['SampleShape', 'binType', 'formatName', 'color', 'type']}

Defines the items known to exist in the Image Controls tree section and the item’s data types. A few are
not included here (‘background image’, ‘dark image’, ‘Gain map’, and ‘calibrant’) because these items have special set routines, where references to entries are checked to make sure their values are correct.

**Integrate** *(name=None, MaskMap=None, ThetaAzimMap=None)*

Invokes an image integration (same as Image Controls/Integration/Integrate menu command). All parameters will have previously been set with Image Controls so no input is needed here. However, the optional parameters MaskMap and ThetaAzimMap may be supplied to save computing these items more than once, speeding integration of multiple images with the same image/mask parameters.

Note that if integration is performed on an image more than once, histogram entries may be overwritten. Use the name parameter to prevent this if desired.

**Parameters**

- **name** *(str)* – base name for created histogram(s). If None (default), the histogram name is taken from the image name.
- **MaskMap** *(list)* – from `calcMaskMap()`
- **ThetaAzimMap** *(list)* – from `calcThetaAzimMap()`

**Returns** a list of created histogram *(G2PwdrData)* objects.

**Recalibrate** *

Invokes a recalibration fit (same as Image Controls/Calibration/Recalibrate menu command). Note that for this to work properly, the calibration coefficients (center, wavelength, distance & tilts) must be fairly close. This may produce a better result if run more than once.

**findControl**(arg="")

Finds the Image Controls parameter(s) in the current image that match the string in arg. Default is ‘’ which returns all parameters.

Example:

```python
>>> findControl('calib')
[['calibskip', 'int'], ['calibdmin', 'float'], ['calibrant', 'str']]
```

**Parameters** arg *(str)* – a string containing part of the name of a parameter (dict entry) in the image’s Image Controls.

**Returns** a list of matching entries in form [['item', 'type'], ['item', 'type']... ] where each ‘item’ string contains the sting in arg.

**getControl**(arg)

Return an Image Controls parameter in the current image. If the parameter is not found an exception is raised.

**Parameters** arg *(str)* – the name of a parameter (dict entry) in the image.

**Returns** the value as a int, float, list,...

**getControls**(clean=False)

returns current Image Controls as a dict

**Parameters** clean *(bool)* – causes the calibration information to be deleted

**getMasks** *

load masks from an IMG tree entry

**getVary** *(args)*

Return the refinement flag(s) for Image Controls parameter(s) in the current image. If the parameter is not found, an exception is raised.
Parameters

- **arg (str)** – the name of a refinement parameter in the varyList for the image. The name should be one of ‘dep’, ‘det-X’, ‘det-Y’, ‘dist’, ‘phi’, ‘tilt’, or ‘wave’

- **arg1 (str)** – the name of a parameter (dict entry) as before, optional

Returns a list of bool value(s)

```python
class

initMasks()
```

Initialize Masks, including resetting the Thresholds values

```python
class loadControls(filename=None, imgDict=None)
```

load controls from a .imctrl file

Parameters

- **filename (str)** – specifies a file to be read, which should end with .imctrl (defaults to None, meaning parameters are input with imgDict.)

- **imgDict (dict)** – contains a set of image parameters (defaults to None, meaning parameters are input with filename.)

loadMasks (filename, ignoreThreshold=False)

load masks from a .immask file

Parameters

- **filename (str)** – specifies a file to be read, which should end with .immask

- **ignoreThreshold (bool)** – If True, masks are loaded with threshold masks. Default is False which means any Thresholds in the file are ignored.

```python
class saveControls(filename)
```

write current controls values to a .imctrl file

Parameters **filename (str)** – specifies a file to write, which should end with .imctrl

```python
class setCalibrant(calib)
```

Set a calibrant for the current image

Parameters **calib (str)** – specifies a calibrant name which must be one of the entries in file ImageCalibrants.py. This is validated and an error provides a list of valid choices.

```python
class setControl(arg, value)
```

Set an Image Controls parameter in the current image. If the parameter is not found an exception is raised.

Parameters

- **arg (str)** – the name of a parameter (dict entry) in the image. The parameter must be found in ControlList or an exception is raised.

- **value** – the value to set the parameter. The value is cast as the appropriate type from ControlList.

```python
class setControlFile(typ, imageRef, multi=None)
```

Set a image to be used as a background/dark/gain map image

Parameters

- **typ (str)** – specifies image type, which must be one of: ‘background image’, ‘dark image’, ‘gain map’; N.B. only the first four characters must be specified and case is ignored.

- **imageRef** – A reference to the desired image. Either the Image tree name (str), the image’s index (int) or a image object (G2Image)
• **mult** *(float)* – a multiplier to be applied to the image (not used for ‘Gain map’; required for ‘background image’, ‘dark image’)

**setControls** *(controlsDict)*
uses dict from `getControls()` to set Image Controls for current image

**setMasks** *(maskDict, resetThresholds=False)*
load masks dict (from `getMasks()`) into current IMG record

**Parameters**

- **maskDict** *(dict)* – specifies a dict with image parameters, from `getMasks()`
- **resetThresholds** *(bool)* – If True, Threshold Masks in the dict are ignored. The default is False which means Threshold Masks are retained.

**setVary** *(arg, value)*
Set a refinement flag for Image Controls parameter in the current image. If the parameter is not ‘*’ or found, an exception is raised.

**Parameters**

- **arg** *(str)* – the name of a refinement parameter in the varyList for the image. The name should be one of ‘dep’, ‘det-X’, ‘det-Y’, ‘dist’, ‘phi’, ‘tilt’, or ‘wave’, or it may be a list or tuple of names, or it may be ‘*’ in which all parameters are set accordingly.
- **value** – the value to set the parameter. The value is cast as the appropriate type from `ControlList`.

**exception** `GSASIIscriptable.G2ImportException`

**class** `GSASIIscriptable.G2ObjectWrapper` *(datadict)*
Base class for all GSAS-II object wrappers.

The underlying GSAS-II format can be accessed as `wrapper.data`. A number of overrides are implemented so that the wrapper behaves like a dictionary.

Author: Jackson O’Donnell (jacksonhodonnell .at. gmail.com)

**class** `GSASIIscriptable.G2PDF` *(data, name, proj)*
Wrapper for a PDF tree entry, containing the information needed to compute a PDF and the S(Q), G(r) etc. after the computation is done. Note that in a GSASIIscriptable script, instances of G2PDF will be created by calls to `G2Project.add_PDF()` or `G2Project.pdf()`, not via calls to `G2PDF.__init__()`.

Example use of `G2PDF`:

```python
import gpx
import pdf

# Create a PDF entry
pdf = gpx.add_PDF('250umSiO2.pdfprm',0)
pdf.set_formula(['Si',1],['O',2])
pdf.set_background('Container',1,-0.21)
for i in range(5):
    if pdf.optimize():
        break
pdf.calculate()
pdf.export(gpx.filename,'S(Q), pdfGUI')
gpx.save('pdfcalc.gpx')
```

See also:

`G2Project.pdf()` `G2Project.pdfs()`

**calculate** *(xydata=None, limits=None, inst=None)*
Compute the PDF using the current parameters. Results are set in the PDF object arrays (self.data['PDF Controls'][‘G(R)’] etc.). Note that if `xydata` is specified, the background histograms(s) will not be accessed from the project file associated with the current PDF entry. If `limits` and `inst` are both
specified, no histograms need be in the current project. However, the self.data['PDF Controls'] sections ('Sample', 'Sample Bkg.', 'Container Bkg.') must be non-blank for the corresponding items to be used from 'xydata'.

Parameters

- **xydata (dict)** – an array containing the Sample’s I vs Q, and any or none of the Sample Background, the Container scattering and the Container Background. If xydata is None (default), the values are taken from histograms, as named in the PDF’s self.data['PDF Controls'] entries with keys ‘Sample’, ‘Sample Bkg.’, ‘Container Bkg.’ & ‘Container’.

- **limits (list)** – upper and lower Q values to be used for PDF computation. If None (default), the values are taken from the Sample histogram’s .data['Limits'][1] values.

- **inst (dict)** – The Sample histogram’s instrument parameters to be used for PDF computation. If None (default), the values are taken from the Sample histogram’s .data['Instrument Parameters'][0] values.

**export (fileroot, formats)**

Write out the PDF-related data \((G(r), S(Q), \ldots)\) into files

Parameters

- **fileroot (str)** – name of file(s) to be written. The extension will be ignored and set to .iq, .sq, .fq or .gr depending on the formats selected.

- **formats (str)** – string specifying the file format(s) to be written, should contain at least one of the following keywords: I(Q), S(Q), F(Q), G(r) and/or PDFgui (capitalization and punctuation is ignored). Note that G(r) and PDFgui should not be specified together.

**optimize (showFit=True, maxCycles=5, xydata=None, limits=None, inst=None)**

Optimize the low R portion of G(R) to minimize selected parameters. Note that this updates the parameters in the settings (self.data['PDF Controls']) but does not update the PDF object arrays (self.data['PDF Controls']['G(R)'] etc.) with the computed values, use **calculate()** after a fit to do that.

Parameters

- **showFit (bool)** – if True (default) the optimized parameters are shown before and after the fit, as well as the RMS value in the minimized region.

- **maxCycles (int)** – the maximum number of least-squares cycles; defaults to 5.

- **xydata (dict)** – an array containing the Sample’s I vs Q, and any or none of the Sample Background, the Container scattering and the Container Background. If xydata is None (default), the values are taken from histograms, as named in the PDF’s self.data['PDF Controls'] entries with keys ‘Sample’, ‘Sample Bkg.’, ‘Container Bkg.’ & ‘Container’.

- **limits (list)** – upper and lower Q values to be used for PDF computation. If None (default), the values are taken from the Sample histogram’s .data['Limits'][1] values.

- **inst (dict)** – The Sample histogram’s instrument parameters to be used for PDF computation. If None (default), the values are taken from the Sample histogram’s .data['Instrument Parameters'][0] values.

Returns the result from the optimizer as True or False, depending on if the refinement converged.

**set_background (btype, histogram, mult=-1.0, refine=False)**

Sets a histogram to be used as the ‘Sample Background’, the ‘Container’ or the ‘Container Background.’

Parameters

- **btype (str)** – Type of background to set, must contain the string ‘samp’ for Sample Background’, ‘cont’ and ‘back’ for the ‘Container Background’ or only ‘cont’ for the
'Container'. Note that capitalization and extra characters are ignored, so the full strings (such as 'Sample Background' & 'Container Background') can be used.

- **histogram** – A reference to a histogram, which can be reference by object, name, or number.

- **mult** (*float*) – a multiplier for the histogram; defaults to -1.0

- **refine** (*bool*) – a flag to enable refinement (only implemented for ‘Sample Background’); defaults to False

```python
def set_formula(*args)
    Set the chemical formula for the PDF computation. Use pdf.set_formula(['Si',1],['O',2]) for SiO2.
```

**Parameters**

- **item1** (*list*) – The element symbol and number of atoms in formula for first element

- **item2** (*list*) – The element symbol and number of atoms in formula for second element,...

  repeat parameters as needed for all elements in the formula.

```python
class GSASIIscriptable.G2Phase(data, name, proj)
    A wrapper object around a given phase. The object contains these class variables:

    - G2Phase.proj: contains a reference to the G2Project object that contains this phase
    - G2Phase.name: contains the name of the phase
    - G2Phase.data: contains the phases’s associated data in a dict, as documented for the Phase Tree items.
```

**Author**: Jackson O’Donnell (jacksonhodonnell .at. gmail.com)

```python
def atom(atomlabel)
    Returns the atom specified by atomlabel, or None if it does not exist.
```

**Parameters**

- **atomlabel** (*str*) – The name of the atom (e.g. “O2”)

  Returns  A G2AtomRecord object representing the atom.

```python
def atoms()
    Returns a list of atoms present in the current phase.
```

  Returns  A list of G2AtomRecord objects.

**See also:**

```
atom() G2AtomRecord
```

```python
def clear_HAP_refinements(refs, histograms='all')
    Clears the given HAP refinement parameters between this phase and the given histograms.
```

**Parameters**

- **refs** (*dict*) – A dictionary of the parameters to be cleared. See the the Histogram-and-phase parameters table for what can be specified.

- **histograms** – Either ‘all’ (default) or a list of the histograms by index, name or object. The index number is relative to all histograms in the tree, not to those in the phase. Histograms not associated with the current phase will be ignored. whose HAP parameters will be set with this phase. Histogram and phase must already be associated

  Returns  None

```python
def clear_refinements(refs)
    Clears a given set of parameters.
```
Parameters **refs** (*dict*) – The parameters to clear. See the *Phase parameters* table for what can be specified.

**composition**

Provides a dict where keys are atom types and values are the number of atoms of that type in cell (such as `{‘H’: 2.0, ‘O’: 1.0}`)

**copyHAPvalues** (*sourcehist, targethistlist= ‘all’, skip=[], use=None*)

Copies HAP parameters for one histogram to a list of other histograms. Use skip or use to select specific entries to be copied or not used.

**Parameters**

- **sourcehist** – is a histogram object (*G2PwdrData*) or a histogram name or the index number of the histogram to copy parameters from. The index number is relative to all histograms in the tree, not to those in the phase.

- **targethistlist** (*list*) – a list of histograms where each item in the list can be a histogram object (*G2PwdrData*), a histogram name or the index number of the histogram. If the string ‘all’ (default), then all histograms in the phase are used.

- **skip** (*list*) – items in the HAP dict that should not be copied. The default is an empty list, which causes all items to be copied. To see a list of items in the dict, use *getHAPvalues()* or use an invalid item, such as ‘?’.

- **use** (*list*) – specifies the items in the HAP dict should be copied. The default is None, which causes all items to be copied.

**examples:**

```python
ph0.copyHAPvalues(0,[1,2,3])
ph0.copyHAPvalues(0,use=['HStrain','Size'])
```

The first example copies all HAP parameters from the first histogram to the second, third and fourth histograms (as listed in the project tree). The second example copies only the ‘HStrain’ (Dij parameters and refinement flags) and the ‘Size’ (crystallite size settings, parameters and refinement flags) from the first histogram to all histograms.

**density**

Provides a scalar with the density of the phase. In case of a powder this assumes a 100% packing fraction.

**export_CIF** (*outputname, quickmode=True*)

Write this phase to a .cif file named outputname.

**Parameters**

- **outputname** (*str*) – The name of the .cif file to write to

- **quickmode** (*bool*) – Currently ignored. Carryover from exports.G2export_CIF

**getHAPvalues** (*histname*)

Returns a dict with HAP values for the selected histogram.

**Parameters**

- **histogram** – is a histogram object (*G2PwdrData*) or a histogram name or the index number of the histogram. The index number is relative to all histograms in the tree, not to those in the phase.

**Returns** HAP value dict

**get_cell** ()

Returns a dictionary of the cell parameters, with keys: `length_a`, `length_b`, `length_c`, `angle_alpha`, `angle_beta`, `angle_gamma`, `volume`
Returns a dict

See also:

get_cell_and_esd()

get_cell_and_esd()

Returns a pair of dictionaries, the first representing the unit cell, the second representing the estimated standard deviations of the unit cell.

Returns a tuple of two dictionaries

See also:

get_cell()

histograms()

Returns a list of histogram names associated with the current phase ordered as they appear in the tree (see G2Project.histograms()).

mu (wave)

Provides mu values for a phase at the supplied wavelength in A. Uses GSASIImath.XScattDen which seems to be off by an order of magnitude, which has been corrected here.

setHAPvalues (HAPdict, targethistlist='all', skip=[], use=None)

Copies HAP parameters for one histogram to a list of other histograms. Use skip or use to select specific entries to be copied or not used. Note that HStrain and sometimes Mustrain values can be specific to a Laue class and should be copied with care between phases of different symmetry. A “sanity check” on the number of Dij terms is made if HStrain values are copied.

Parameters

- HAPdict (dict) – is a dict returned by getHAPvalues() containing HAP parameters.
- targethistlist (list) – a list of histograms where each item in the list can be a histogram object (G2PwdrData), a histogram name or the index number of the histogram. The index number is relative to all histograms in the tree, not to those in the phase. If the string ‘all’ (default), then all histograms in the phase are used.
- skip (list) – items in the HAP dict that should not be copied. The default is an empty list, which causes all items to be copied. To see a list of items in the dict, use getHAPvalues() or use an invalid item, such as ‘?’.
- use (list) – specifies the items in the HAP dict should be copied. The default is None, which causes all items to be copied.

example:

HAPdict = ph0.getHAPvalues(0)
ph1.setHAPvalues(HAPdict, use=['HStrain', 'Size'])

This copies the Dij (hydrostatic strain) HAP parameters and the crystallite size broadening terms from the first histogram in phase ph0 to all histograms in phase ph1.

set_HAP_refinements (refs, histograms='all')

Sets the given HAP refinement parameters between the current phase and the specified histograms.

Parameters

- refs (dict) – A dictionary of the parameters to be set. See the Histogram-and-phase parameters table for a description of this dictionary.
• **histograms** – Either ‘all’ (default) or a list of the histograms by index, name or object. The index number is relative to all histograms in the tree, not to those in the phase. Histograms not associated with the current phase will be ignored. whose HAP parameters will be set with this phase. Histogram and phase must already be associated.

**Returns**  None

```python
set_refinements(refs)
```

Sets the phase refinement parameter ‘key’ to the specification ‘value’

**Parameters**  
ref (dict) – A dictionary of the parameters to be set. See the Phase parameters table for a description of this dictionary.

**Returns**  None

```python
class GSASIIscriptable.G2Project(gpxfile=None, author=None, filename=None, newgpx=None)
```

Represents an entire GSAS-II project. The object contains these class variables:

• `G2Project.filename`: contains the .gpx filename

• `G2Project.names`: contains the contents of the project “tree” as a list of lists. Each top-level entry in the tree is an item in the list. The name of the top-level item is the first item in the inner list. Children of that item, if any, are subsequent entries in that list.

• `G2Project.data`: contains the entire project as a dict. The keys for the dict are the top-level names in the project tree (initial items in the `G2Project.names` inner lists) and each top-level item is stored as a dict.

  – The contents of Top-level entries will be found in the item named ‘data’, as an example, `G2Project.data[‘Notebook’][‘data’]`

  – The contents of child entries will be found in the item using the names of the parent and child, for example `G2Project.data[‘Phases’][‘NaCl’]`

**Parameters**

• `gpxfile (str)` – Existing .gpx file to be loaded. If nonexistent, creates an empty project.

• `author (str)` – Author’s name (not yet implemented)

• `newgpx (str)` – The filename the project should be saved to in the future. If both newgpx and gpxfile are present, the project is loaded from the gpxfile, then when saved will be written to newgpx.

• `filename (str)` – Name to be used to save the project. Has same function as parameter newgpx (do not use both gpxfile and filename). Use of newgpx is preferred over filename.

There are two ways to initialize this object:

```python
>>> # Load an existing project file
>>> proj = G2Project('filename.gpx')

>>> # Create a new project
>>> proj = G2Project(newgpx='new_file.gpx')
```

Histograms can be accessed easily.

```python
>>> # By name
>>> hist = proj.histogram('PWDR my-histogram-name')
```
Phases can be accessed the same way.

```python
>>> phase = proj.phase('name of phase')
```

New data can also be loaded via `add_phase()` and `add_powder_histogram()`.

```python
>>> hist = proj.add_powder_histogram('some_data_file.chi',
                                   'instrument_parameters.prm')
>>> phase = proj.add_phase('my_phase.cif', histograms=[hist])
```

Parameters for Rietveld refinement can be turned on and off as well. See `set_refinement()`, `clear_refinements()`, `iter_refinements()`, `do_refinements()`.

### add_PDF (prmfile, histogram)

Creates a PDF entry that can be used to compute a PDF. Note that this command places an entry in the project, but `G2PDF.calculate()` must be used to actually perform the computation.

**Parameters**

- `datafile (str)` – The powder data file to read, a filename.
- `histogram` – A reference to a histogram, which can be reference by object, name, or number.

**Returns** A `G2PDF` object for the PDF entry

### add_constraint_raw (cons_scope, constr)

Adds a constraint of type `consType` to the project. `cons_scope` should be one of “Hist”, “Phase”, “HAP”, or “Global”.

**WARNING** it does not check the constraint is well-constructed

### add_image (imagefile, fmt_hint=None, defaultImage=None)

Load an image into a project

**Parameters**

- `imagefile (str)` – The image file to read, a filename.
- `fmt_hint (str)` – If specified, only importers where the format name (reader.formatName, as shown in Import menu) contains the supplied string will be tried as importers. If not specified, all importers consistent with the file extension will be tried (equivalent to “guess format” in menu).
- `defaultImage (str)` – The name of an image to use as a default for setting parameters for the image file to read.

**Returns** a list of `G2Image` object for the added image(s) [this routine has not yet been tested with files with multiple images...]

### add_phase (phasefile, phasename=None, histograms=[], fmt_hint=None)

Loads a phase into the project from a .cif file

**Parameters**
• **phasefile** *(str)* – The CIF file from which to import the phase.

• **phasename** *(str)* – The name of the new phase, or None for the default.

• **histograms** *(list)* – The names of the histograms to associate with this phase. Use proj.histograms() to add to all histograms.

• **fmthint** *(str)* – If specified, only importers where the format name (reader.formatName, as shown in Import menu) contains the supplied string will be tried as importers. If not specified, all importers consistent with the file extension will be tried (equivalent to “guess format” in menu).

**Returns** A `G2Phase` object representing the new phase.

```python
add_powder_histogram(datafile, iparams, phases=[], fmthint=None, databank=None, instbank=None)
```

Loads a powder data histogram into the project.

Automatically checks for an instrument parameter file, or one can be provided. Note that in unix fashion, “~” can be used to indicate the home directory (e.g. ~/G2data/data.fxye).

**Parameters**

• **datafile** *(str)* – The powder data file to read, a filename.

• **iparams** *(str)* – The instrument parameters file, a filename.

• **phases** *(list)* – A list of phases to link to the new histogram, phases can be references by object, name, rld or number. Alternately, use ‘all’ to link to all phases in the project.

• **fmthint** *(str)* – If specified, only importers where the format name (reader.formatName, as shown in Import menu) contains the supplied string will be tried as importers. If not specified, all importers consistent with the file extension will be tried (equivalent to “guess format” in menu).

• **databank** *(int)* – Specifies a dataset number to read, if file contains more than set of data. This should be 1 to read the first bank in the file (etc.) regardless of the number on the Bank line, etc. Default is None which means there should only be one dataset in the file.

• **instbank** *(int)* – Specifies an instrument parameter set to read, if the instrument parameter file contains more than set of parameters. This will match the INS # in an GSAS type file so it will typically be 1 to read the first parameter set in the file (etc.) Default is None which means there should only be one parameter set in the file.

**Returns** A `G2PwdrData` object representing the histogram.

```python
add_simulated_powder_histogram(histname, iparams, Tmin, Tmax, Tstep, wavelength=None, scale=None, phases=[])```

Loads a powder data histogram into the project.

Requires an instrument parameter file. Note that in unix fashion, “~” can be used to indicate the home directory (e.g. ~/G2data/data.prm). The instrument parameter file will determine if the histogram is x-ray, CW neutron, TOF, etc. as well as the instrument type.

**Parameters**

• **histname** *(str)* – A name for the histogram to be created.

• **iparams** *(str)* – The instrument parameters file, a filename.

• **Tmin** *(float)* – Minimum 2theta or TOF (ms) for dataset to be simulated

• **Tmax** *(float)* – Maximum 2theta or TOF (ms) for dataset to be simulated
• **Tstep**(float) – Step size in 2theta or TOF (ms) for dataset to be simulated

• **wavelength**(float) – Wavelength for CW instruments, overriding the value in the instrument parameters file if specified.

• **scale**(float) – Histogram scale factor which multiplies the pattern. Note that simulated noise is added to the pattern, so that if the maximum intensity is small, the noise will mask the computed pattern. The scale needs to be a large number for CW neutrons. The default, None, provides a scale of 1 for x-rays and TOF; 10,000 for CW neutrons.

• **phases**(list) – Phases to link to the new histogram. Use proj.phases() to link to all defined phases.

**Returns** A *G2PwdrData* object representing the histogram

**clone_powder_histogram**(histref, newname, Y, Yerr=None)

Creates a copy of a powder diffraction histogram with new Y values. The X values are not changed. The number of Y values must match the number of X values.

**Parameters**

• **histref** – The histogram object, the name of the histogram (str), or ranId or histogram index.

• **newname**(str) – The name to be assigned to the new histogram

• **Y**(list) – A set of intensity values

• **Yerr**(list) – A set of uncertainties for the intensity values (may be None, sets all weights to unity)

**Returns** the new histogram object (type G2PwdrData)

**copyHistParms**(sourcehist, targethistlist=‘all’, modelist=‘all’)

Copy histogram information from one histogram to others

**Parameters**

• **sourcehist** – is a histogram object (*G2PwdrData*) or a histogram name or the index number of the histogram

• **targethistlist**(list) – a list of histograms where each item in the list can be a histogram object (*G2PwdrData*), a histogram name or the index number of the histogram. if the string ‘all’ (default value), then all histograms in the project are used.

• **modelist**(list) – May be a list of sections to copy, which may include ‘Background’, ‘Instrument Parameters’, ‘Limits’ and ‘Sample Parameters’ (items may be shortened to uniqueness and capitalization is ignored, so [‘b’, ‘i’, ‘L’, ‘s’] will work.) The default value, ‘all’ causes the listed sections to

**copy_PDF**(PDFobj, histogram)

Creates a PDF entry that can be used to compute a PDF as a copy of settings in an existing PDF (*G2PDF*) object. This places an entry in the project but *G2PDF.calculate()* must be used to actually perform the PDF computation.

**Parameters**

• **PDFobj** – A *G2PDF* object which may be in a separate project or the dict associated with the PDF object (G2PDF.data).

• **histogram** – A reference to a histogram, which can be reference by object, name, or number.

**Returns** A *G2PDF* object for the PDF entry
do_refinements (refinements, histogram='all', phase='all', outputnames=None, makeBack=False)

Conducts one or a series of refinements according to the input provided in parameter refinements. This is a wrapper around iter_refinements()

Parameters

- **refinements (list)** – A list of dictionaries specifying changes to be made to parameters before refinements are conducted. See the Refinement recipe section for how this is defined.

- **histogram (str)** – Name of histogram for refinements to be applied to, or ‘all’; note that this can be overridden for each refinement step via a “histograms” entry in the dict.

- **phase (str)** – Name of phase for refinements to be applied to, or ‘all’; note that this can be overridden for each refinement step via a “phases” entry in the dict.

- **outputnames (list)** – Provides a list of project (.gpx) file names to use for each refinement step (specifying None skips the save step). See save(). Note that this can be overridden using an “output” entry in the dict.

- **makeBack (bool)** – determines if a backup .bckX.gpx file is made before a refinement is performed. The default is False.

To perform a single refinement without changing any parameters, use this call:

```python
my_project.do_refinements([])
```

classmethod from_dict_and_names (gpxdict, names, filename=None)

Creates a G2Project directly from a dictionary and a list of names. If in doubt, do not use this.

Returns a G2Project

get_Controls (control)

Return project controls settings

- **Parameters control (str)** – the item to be returned. See below for allowed values.

Returns The value for the control.

Allowed values for parameter control:

- cycles: the maximum number of cycles (returns int)

- sequential: the histograms used for a sequential refinement as a list of histogram names or an empty list when in non-sequential mode.

- Reverse Seq: returns True or False. True indicates that fitting of the sequence of histograms proceeds in reversed order.

- seqCopy: returns True or False. True indicates that results from each sequential fit are used as the starting point for the next histogram.

- Anything else returns the value in the Controls dict, if present. An exception is raised if the control value is not present.

See also:

set_Controls()

get_Covariance (varList)

Returns the values and covariance matrix for a series of variable parameters. as defined in the last refinement cycle
Parameters \texttt{varList} (\texttt{tuple}) – a list of variable names of form ‘<p>:<h>:<name>’

Returns \texttt{(valueList,CovMatrix)} where \texttt{valueList} contains the \texttt{(n)} values in the same order as \texttt{varList} (also length \texttt{n}) and \texttt{CovMatrix} is a \texttt{(n x n)} matrix. If any variable name is not found in the \texttt{varList} then \texttt{None} is returned.

Use this code, where \texttt{sig} provides standard uncertainties for parameters and where \texttt{covArray} provides the correlation between off-diagonal terms:

\begin{verbatim}
sig = np.sqrt(np.diag(covMatrix))
xvar = np.outer(sig,np.ones_like(sig))
covArray = np.divide(np.divide(covMatrix,xvar),xvar.T)
\end{verbatim}

\texttt{get_ParmList}()
Returns a list of all the parameters defined in the last refinement cycle

\texttt{get_Variable} (\texttt{var})
Returns the value and standard uncertainty (\texttt{esd}) for a variable parameters, as defined in the last refinement cycle

Parameters \texttt{var} (\texttt{str}) – a variable name of form ‘<p>:<h>:<name>’, such as ‘:0:Scale’

Returns \texttt{(value,esd)} if the parameter is refined or \texttt{(value, None)} if the variable is in a constraint or is not refined or \texttt{None} if the parameter is not found.

\texttt{get_VaryList}()
Returns a list of the refined variables in the last refinement cycle

\texttt{histogram} (\texttt{histname})
Returns the histogram named \texttt{histname}, or \texttt{None} if it does not exist.

Parameters \texttt{histname} – The name of the histogram (\texttt{str}), or ranId or index.

Returns \texttt{A G2PwdrData object}, or \texttt{None} if the histogram does not exist

See also:
\texttt{histograms()} \texttt{phase()} \texttt{phases()}

\texttt{histograms} (\texttt{typ=None})
Return a list of all histograms, as \texttt{G2PwdrData objects}

For now this only finds Powder/Single Xtal histograms, since that is all that is currently implemented in this module.

Parameters \texttt{typ} (\texttt{ste}) – The prefix (\texttt{type}) the histogram such as ‘PWDR’. If None (the default) all known histograms types are found.

Returns \texttt{a list of objects}

See also:
\texttt{histogram()} \texttt{phase()} \texttt{phases()}

\texttt{hold_many} (\texttt{vars, type})
Apply holds for all the variables in \texttt{vars}, for constraint of a given type.

type is passed directly to \texttt{add_constraint_raw} as \texttt{consType}

Parameters
• **vars** *(list)* – A list of variables to hold. Either `GSASIIobj.G2VarObj` objects, string variable specifiers, or arguments for `make_var_obj()`

• **type** *(str)* – A string constraint type specifier. See `G2Project.add_constraint_raw`

`image(imageRef)`

Gives an object representing the specified image in this project.

**Parameters**

- **imageRef** *(str)* – A reference to the desired image. Either the Image tree name (str), the image’s index (int) or a image object (`G2Image`)

**Returns**

A `G2Image` object

**Raises**

`KeyError`

See also:

`images()`

`imageMultiDistCalib(imageList=None, verbose=False)`

Invokes a global calibration fit (same as Image Controls/Calibration/Multi-distance Recalibrate menu command) with images as multiple distance settings. Note that for this to work properly, the initial calibration parameters (center, wavelength, distance & tilts) must be close enough to converge. This may produce a better result if run more than once.

See `Image Calibration` for example code.

**Parameters**

- **imageList** *(str)* – the images to include in the fit, if not specified all images in the project will be included.

**Returns**

`parmDict,covData` where `parmDict` has the refined parameters and their values and `covData` is a dict containing the covariance matrix (‘covMatrix’), the number of ring picks (‘obs’) the reduced Chi-squared (‘chisq’), the names of the variables (‘varyList’) and their values (‘variables’)

`images()`

Returns a list of all the images in the project.

**Returns**

A list of `G2Image` objects

`iter_refinements(refinements, histogram='all', phase='all', outputnames=None, makeBack=False)`

Conducts a series of refinements, iteratively. Stops after every refinement and yields this project, to allow error checking or logging of intermediate results. Parameter use is the same as for `do_refinements()` (which calls this method).

```python
>>> def checked_refinements(proj):
...     for p in proj.iter_refinements(refs):
...         # Track intermediate results
...         log(p.histogram('0').residuals)
...         log(p.phase('0').get_cell())
...         # Check if parameter diverged, nonsense answer, or whatever
...         if is_something_wrong(p):
...             raise Exception("I need a human!")
```

`link_histogram_phase(histogram, phase)`

Associates a given histogram and phase.

See also:

`histogram()` `phase()`
make_var_obj (phase=None, hist=None, varname=None, atomId=None, reloadIdx=True)
Wrapper to create a G2VarObj. Takes either a string representation (“p:h:name:a”) or individual names of
phase, histogram, varname, and atomId.

Automatically converts string phase, hist, or atom names into the ID required by G2VarObj.

df (pdfRef)
Gives an object representing the specified PDF entry in this project.

Parameters pdfRef – A reference to the desired image. Either the PDF tree name (str), the
pdf’s index (int) or a PDF object (G2PDF)

Returns A G2PDF object

Raises KeyError

See also:

dfs () G2PDF

pdfs ()
Returns a list of all the PDFs in the project.

Returns A list of G2PDF objects

phase (phasename)
Gives an object representing the specified phase in this project.

Parameters phasename (str) – A reference to the desired phase. Either the phase name (str),
the phase’s ranId, the phase’s index (both int) or a phase object (G2Phase)

Returns A G2Phase object

Raises KeyError

See also:

histograms () phase () phases ()

phases ()
Returns a list of all the phases in the project.

Returns A list of G2Phase objects

See also:

histogram () histograms () phase ()

refine (newfile=None, printFile=None, makeBack=False)
Invoke a refinement for the project. The project is written to the currently selected gpx file and then either
a single or sequential refinement is performed depending on the setting of ‘Seq Data’ in Controls (set in
get.Controls())

reload ()
Reload self from self.filename

save (filename=None)
Saves the project, either to the current filename, or to a new file.

Updates self.filename if a new filename provided

seqref ()
Returns a sequential refinement results object, if present

Returns A G2SeqRefRes object or None if not present
**set_Controls**(control, value)
Set project controls

**Parameters**

- **control**(str) – the item to be set. See below for allowed values.
- **value** – the value to be set.

Allowed values for parameter control:

- **cycles**: sets the maximum number of cycles (value must be int)
- **sequential**: sets the histograms to be used for a sequential refinement. Use an empty list to turn off sequential fitting. The values in the list may be the name of the histogram (a str), or a ranld or index (int values), see `histogram()`.
- **seqCopy**: when True, the results from each sequential fit are used as the starting point for the next. After each fit is is set to False. Ignored for non-sequential fits.
- **Reverse Seq**: when True, sequential refinement is performed on the reversed list of histograms.

**See also:**

`get_Controls()`

**set_refinement**(refinement, histogram='all', phase='all')
Apply specified refinements to a given histogram(s) or phase(s).

**Parameters**

- **refinement**(dict) – The refinements to be conducted
- **histogram** – Specifies either ‘all’ (default), a single histogram or a list of histograms. Histograms may be specified as histogram objects (see `G2PwdrData`), the histogram name (str) or the index number (int) of the histogram in the project, numbered starting from 0. Omitting the parameter or the string ‘all’ indicates that parameters in all histograms should be set.
- **phase** – Specifies either ‘all’ (default), a single phase or a list of phases. Phases may be specified as phase objects (see `G2Phase`), the phase name (str) or the index number (int) of the phase in the project, numbered starting from 0. Omitting the parameter or the string ‘all’ indicates that parameters in all phases should be set.

Note that refinement parameters are categorized as one of three types:

1. Histogram parameters
2. Phase parameters
3. Histogram-and-Phase (HAP) parameters

**See also:**


**update_ids**()
Makes sure all phases and histograms have proper hId and pId

**class** **GSASIIscriptable.G2PwdrData**(data, proj, name)
Wraps a Powder Data Histogram. The object contains these class variables:

- **G2PwdrData.proj**: contains a reference to the `G2Project` object that contains this histogram
• G2PwdrData.name: contains the name of the histogram

• G2PwdrData.data: contains the histogram’s associated data in a dict, as documented for the Powder Diffraction Tree. The actual histogram values are contained in the ‘data’ dict item, as documented for Data.

**Background**

Provides a list with with the Background parameters for this histogram.

**Returns** list containing a list and dict with background values

**Export** *(fileroot, extension, fmthint=None)*

Write the histogram into a file. The path is specified by fileroot and extension.

**Parameters**

- **fileroot** *(str)* – name of the file, optionally with a path (extension is ignored)

- **extension** *(str)* – includes ‘.’, must match an extension in global exportersByExtension[‘powder’] or a Exception is raised.

- **fmthint** *(str)* – If specified, the first exporter where the format name (obj.formatName, as shown in Export menu) contains the supplied string will be used. If not specified, an error will be generated showing the possible choices.

**Returns** name of file that was written

**Export_peaks** *(filename)*

Write the peaks file. The path is specified by filename extension.

**Parameters**

- **filename** *(str)* – name of the file, optionally with a path, includes an extension

**Returns** name of file that was written

**InstrumentParameters**

Provides a dictionary with with the Instrument Parameters for this histogram.

**LoadProfile** *(filename, bank=0)*

Reads a GSAS-II (new style).instprm file and overwrites the current parameters

**Parameters**

- **filename** *(str)* – instrument parameter file name, extension ignored if not .instprm

- **bank** *(int)* – bank number to read, defaults to zero

**PeakList**

Provides a list of peaks parameters for this histogram.

**Returns** a list of peaks, where each peak is a list containing [pos,area,sig,gam] (position, peak area, Gaussian width, Lorentzian width)

**Peaks**

Provides a dict with the Peak List parameters for this histogram.

**Returns** dict with two elements where item ‘peaks’ is a list of peaks where each element is [pos,pos-ref,area,area-ref,sig,sig-ref,gam,gam-ref], where the -ref items are refinement flags and item ‘sigDict’ is a dict with possible items ‘Back;#’, ‘pos;#’, ‘int;#’, ‘sig;#’, ‘gam;#’

**SampleParameters**

Provides a dictionary with with the Sample Parameters for this histogram.

**SaveProfile** *(filename)*

Writes a GSAS-II (new style).instprm file
add_back_peak (pos, int, sig, gam, refflags=[])

Adds a background peak to the Background parameters

Parameters

- **pos** (float) – position of peak, a 2theta or TOF value
- **int** (float) – integrated intensity of background peak, usually large
- **sig** (float) – Gaussian width of background peak, usually large
- **gam** (float) – Lorentzian width of background peak, usually unused (small)
- **refflags** (list) – a list of 1 to 4 boolean refinement flags for pos,int,sig & gam, respectively (use [0,1] to refine int only). Defaults to [] which means nothing is refined.

add_peak (area, dspace=None, Q=None, ttheta=None)

Adds a single peak to the peak list

- **area** (float) – peak area
- **dspace** (float) – peak position as d-space (Å)
- **Q** (float) – peak position as Q (Å⁻¹)
- **ttheta** (float) – peak position as 2Theta (deg)

Note: only one of the parameters: dspace, Q or ttheta may be specified. See Peak Fitting for an example.

clear_refinements (refs)

Clears the refinement parameter ‘key’ and its associated value.

Parameters

- **refs** (dict) – A dictionary of parameters to clear. See the Histogram parameters table for what can be specified.

del_back_peak (peaknum)

Removes a background peak from the Background parameters

Parameters

- **peaknum** (int) – the number of the peak (starting from 0)

fit_fixed_points ()

Attempts to apply a background fit to the fixed points currently specified.

get_wR ()

returns the overall weighted profile R factor for a histogram

Returns

a wR value as a percentage or None if not defined

geta (datatype)

Provides access to the histogram data of the selected data type

Parameters

- **datatype** (str) – must be one of the following values (case is ignored)
  - ’X’: the 2theta or TOF values for the pattern
  - ’Yobs’: the observed intensity values
  - ’Yweight’: the weights for each data point (1/sigma**2)
  - ’Ycalc’: the computed intensity values
  - ’Background’: the computed background values
  - ’Residual’: the difference between Yobs and Ycalc (obs-calc)

Returns

an numpy MaskedArray with data values of the requested type

ref_back_peak (peaknum, refflags=[])

Sets refinement flag for a background peak

Parameters

- **peaknum** (int) – the number of the peak (starting from 0)
• **refflags** *(list)* – a list of 1 to 4 boolean refinement flags for pos, int, sig & gam, respectively. If a flag is not specified it defaults to False (use [0,1] to refine int only). Defaults to [] which means nothing is refined.

**refine_peaks()**
Causes a refinement of peak position, background and instrument parameters

**reflections()**
Returns a dict with an entry for every phase in the current histogram. Within each entry is a dict with keys ‘RefList’ (reflection list, see Powder Reflections), ‘Type’ (histogram type), ‘FF’ (form factor information), ‘Super’ (True if this is superspace group).

**residuals**
Provides a dictionary with with the R-factors for this histogram. Includes the weighted and unweighted profile terms (R, Rb, wR, wRb, wRmin) as well as the Bragg R-values for each phase (ph:H:Rf and ph:H:Rf^2).

**set_background**(key, value)
Set background parameters (this serves a similar function as in set_refinements(), but with a simplified interface).

**Parameters**

- **key** *(str)* – a string that defines the background parameter that will be changed. Must appear in the table below.

<table>
<thead>
<tr>
<th>key name</th>
<th>type of value</th>
<th>meaning of value</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed-Hist</td>
<td>int, str, None or G2PwdrData</td>
<td>reference to a histogram in the current project or None to remove the reference.</td>
</tr>
<tr>
<td>fixed-File-Mult</td>
<td>float</td>
<td>multiplier applied to intensities in the background histogram where a value of -1.0 means full subtraction of the background histogram.</td>
</tr>
</tbody>
</table>

- **value** – a value to set the selected background parameter. The meaning and type for this parameter is listed in the table above.

**set_peakFlags**(peaklist=None, area=None, pos=None, sig=None, gam=None)
Set refinement flags for peaks

**Parameters**

- **peaklist** *(list)* – a list of peaks to change flags. If None (default), changes are made to all peaks.
- **area** *(bool)* – Sets or clears the refinement flag for the peak area value. If None (the default), no change is made.
- **pos** *(bool)* – Sets or clears the refinement flag for the peak position value. If None (the default), no change is made.
- **sig** *(bool)* – Sets or clears the refinement flag for the peak sig (Gaussian width) value. If None (the default), no change is made.
- **gam** *(bool)* – Sets or clears the refinement flag for the peak sig (Lorentzian width) value. If None (the default), no change is made.

Note that when peaks are first created the area flag is on and the other flags are initially off.

Example:
set_peakFlags(sig=False, gam=True)

causes the sig refinement flag to be cleared and the gam flag to be set, in both cases for all peaks. The position and area flags are not changed from their previous values.

set_refinements(refs)
Sets the histogram refinement parameter ‘key’ to the specification ‘value’.

Parameters:

- refs (dict) – A dictionary of the parameters to be set. See the Histogram parameters table for a description of what these dictionaries should be.

Returns:

None

y_calc()
Returns the calculated intensity values; better to use getdata()

exception GSASIIscriptable.G2ScriptException
class GSASIIscriptable.G2SeqRefRes(data, proj)
Wrapper for a Sequential Refinement Results tree entry, containing the results for a refinement

As an example:

```python
from __future__ import division, print_function
import os, sys
sys.path.insert(0, '/Users/toby/software/G2/GSASII')
PathWrap = lambda fil: os.path.join('/Users/toby/Scratch/SeqTut2019Mar', fil)
import GSASIIscriptable as G2sc
gpx = G2sc.G2Project(PathWrap('scr4.gpx'))
seq = gpx.seqref()
lbl = ('a', 'b', 'c', 'alpha', 'beta', 'gamma', 'Volume')
for j, h in enumerate(seq.histograms()):
    cell, cellU, uniq = seq.get_cell_and_esd(1, h)
    print(h)
    print([cell[i] for i in list(uniq)+[6]])
    print([cellU[i] for i in list(uniq)+[6]])
    print('')
    print('printed', [lbl[i] for i in list(uniq)+[6]])
```

See also:

G2Project.seqref()

RefData(hist)
Provides access to the output from a particular histogram

Parameters:

- hist – Specify a histogram or using the histogram name (str) or the index number (int) of the histogram in the sequential refinement (not the project), numbered as in the project tree starting from 0.

Returns:

A list of dicts where the first element has sequential refinement results and the second element has the contents of the histogram tree items.

get_Covariance(hist, varList)
Returns the values and covariance matrix for a series of variable parameters, as defined for the selected histogram in the last sequential refinement cycle

Parameters:

- hist – Specify a histogram or using the histogram name (str) or the index number (int) of the histogram in the sequential refinement (not the project), numbered as in the project tree starting from 0.
• **varList (tuple)** – a list of variable names of form ‘<p>:<h>:<name>’

**Returns** (valueList,CovMatrix) where valueList contains the (n) values in the same order as varList (also length n) and CovMatrix is a (n x n) matrix. If any variable name is not found in the varyList then None is returned.

Use this code, where sig provides standard uncertainties for parameters and where covArray provides the correlation between off-diagonal terms:

```python
sig = np.sqrt(np.diag(covMatrix))
xvar = np.outer(sig, np.ones_like(sig))
covArray = np.divide(np.divide(covMatrix, xvar), xvar.T)
```

**get_ParmList (hist)**

Returns a list of all the parameters defined in the last refinement cycle for the selected histogram

**Parameters**

• **hist** – Specify a histogram or using the histogram name (str) or the index number (int) of the histogram in the sequential refinement (not the project), numbered as in the project tree starting from 0.

**Returns** a list of parameters or None if no refinement has been performed.

**get_Variable (hist, var)**

Returns the value and standard uncertainty (esd) for a variable parameters, as defined for the selected histogram in the last sequential refinement cycle

**Parameters**

• **hist** – Specify a histogram or using the histogram name (str) or the index number (int) of the histogram in the sequential refinement (not the project), numbered as in the project tree starting from 0.

• **var (str)** – a variable name of form ‘<p>:<h>:<name>’, such as ‘:0:Scale’

**Returns** (value,esd) if the parameter is refined or (value, None) if the variable is in a constraint or is not refined or None if the parameter is not found.

**get_VaryList (hist)**

Returns a list of the refined variables in the last refinement cycle for the selected histogram

**Parameters**

• **hist** – Specify a histogram or using the histogram name (str) or the index number (int) of the histogram in the sequential refinement (not the project), numbered starting from 0.

**Returns** a list of variables or None if no refinement has been performed.

**get_cell_and_esd (phase, hist)**

Returns a vector of cell lengths and esd values

**Parameters**

• **phase** – A phase, which may be specified as a phase object (see G2Phase), the phase name (str) or the index number (int) of the phase in the project, numbered starting from 0.

• **hist** – Specify a histogram or using the histogram name (str) or the index number (int) of the histogram in the sequential refinement (not the project), numbered as in the project tree starting from 0.

**Returns** cell,cellESD,uniqCellIndx where cell (list) with the unit cell parameters (a,b,c,alpha,beta,gamma,Volume); cellESD are the standard uncertainties on the 7 unit cell parameters; and uniqCellIndx is a tuple with indicies for the unique (non-symmetry determined) unit parameters (e.g. [0,2] for a,c in a tetragonal cell)
histograms()
returns a list of histograms in the sequential fit

GSASIIscriptable.GenerateReflections(spcGrp, cell, Qmax=None, dmin=None, TTmax=None, wave=None)
Generates the crystallographically unique powder diffraction reflections for a lattice and space group (see GSASIIlattice.GenHLauue()).

Parameters
- **spcGrp** *(str)* — A GSAS-II formatted space group (with spaces between axial fields, e.g. ‘P 21 21 21’ or ‘P 42/m m c’). Note that non-standard space groups, such as ‘P 21/n’ or ‘F -1’ are allowed (see GSASIIspc.SpcGroup()).
- **cell** *(list)* — A list/tuple with six unit cell constants, (a, b, c, alpha, beta, gamma) with values in Angstroms/degrees. Note that the cell constants are not checked for consistency with the space group.
- **Qmax** *(float)* — Reflections up to this Q value are computed (do not use with dmin or TTmax)
- **dmin** *(float)* — Reflections with d-space above this value are computed (do not use with Qmax or TTmax)
- **TTmax** *(float)* — Reflections up to this 2-theta value are computed (do not use with dmin or Qmax, use of wave is required.)
- **wave** *(float)* — wavelength in Angstroms for use with TTmax (ignored otherwise.)

Returns a list of reflections, where each reflection contains four items: h, k, l, d, where d is the d-space (Angstroms)

Example:

```python
>>> import os, sys
>>> sys.path.insert(0, '/Users/toby/software/G2/GSASII')
>>> import GSASIIscriptable as G2sc
GSAS-II binary directory: /Users/toby/software/G2/GSASII/bin
17 values read from config file /Users/toby/software/G2/GSASII/config.py
>>> refs = G2sc.GenerateReflections('P 1',
... (5., 6., 7., 90., 90., 90),
... TTmax=20, wave=1)
>>> for r in refs: print(r)
... [0, 0, 1, 7.0]
[0, 1, 0, 6.0]
[1, 0, 0, 5.0]
[0, 1, 1, 4.55553961419178]
[0, 1, -1, 4.55553961419178]
[1, 0, 1, 4.068667356033675]
[1, 0, -1, 4.068667356033674]
[1, 1, 0, 3.8411063979868794]
[1, -1, 0, 3.8411063979868794]
```

GSASIIscriptable.IPyBrowse(args)
Load a .gpx file and then open a IPython shell to browse it:

```python
usage: GSASIIscriptable.py browse [-h] files [files ...]
```

positional arguments:
**files**

list of files to browse

optional arguments:

```
-h, --help   show this help message and exit
```

**GSASIIscriptable.LoadDictFromProjFile** *(ProjFile)*

Read a GSAS-II project file and load items to dictionary

**Parameters**

`ProjFile (str)` – GSAS-II project (name.gpx) full file name

**Returns**

Project, nameList, where

- Project (dict) is a representation of gpx file following the GSAS-II tree structure for each item: key = tree name (e.g. `Controls`, `Restraints`, etc.), data is dict data dict = `{data: item data which may be list, dict or None, subitems: subdata (if any)}`
- nameList (list) has names of main tree entries & subentries used to reconstruct project file

Example for fap.gpx:

```
Project = {
    #NB: dict order is not tree order
    'Phases': {'data': None, 'fap': {phase dict}},
    'PWDR FAP.XRA Bank 1': {'data': [histogram data list], 'Comments': comments, 'Limits': limits, etc},
    'Rigid bodies': {'data': {rigid body dict}},
    'Covariance': {'data': {covariance data dict}},
    'Controls': {'data': {controls data dict}},
    'Notebook': {'data': [notebook list]},
    'Restraints': {'data': {restraint data dict}},
    'Constraints': {'data': {constraint data dict}}
}
nameList = [
    #NB: reproduces tree order
    ['Notebook'],
    ['Controls'],
    ['Covariance'],
    ['Constraints'],
    ['Restraints'],
    ['Rigid bodies'],
    ['PWDR FAP.XRA Bank 1',
        'Comments',
        'Limits',
        'Background',
        'Instrument Parameters',
        'Sample Parameters',
        'Peak List',
        'Index Peak List',
        'Unit Cells List',
        'Reflection Lists'],
    ['Phases', 'fap']
]
```

**GSASIIscriptable.LoadG2fil** ()

Setup GSAS-II importers. Delay importing this module when possible, it is slow. Multiple calls are not. Only the first does anything.

**GSASIIscriptable.PreSetup** *(data)*

Create part of an initial (empty) phase dictionary
from GSASIIphsGUI.py, near end of UpdatePhaseData

Author: Jackson O’Donnell (jacksonhodonnell .at. gmail.com)

GSASIIscriptable.Readers = {'Image': [], 'Phase': [], 'Pwdr': []}
Readers by reader type

GSASIIscriptable.SaveDictToProjFile(Project, nameList, ProjFile)
Save a GSAS-II project file from dictionary/nameList created by LoadDictFromProjFile
Parameters
• Project (dict) – representation of gpx file following the GSAS-II tree structure as de-
dscribed for LoadDictFromProjFile
• nameList (list) – names of main tree entries & subentries used to reconstruct project
file
• ProjFile (str) – full file name for output project.gpx file (including extension)

GSASIIscriptable.SetDefaultDData(dType, histoName, NShkl=0, NDij=0)
Create an initial Histogram dictionary

Author: Jackson O’Donnell (jacksonhodonnell .at. gmail.com)

GSASIIscriptable.SetPrintLevel(level)
Set the level of output from calls to GSASIIfiles.G2Print(), which should be used in place of print()
where possible. This is a wrapper for GSASIIfiles.G2SetPrintLevel() so that this routine is docu-
mented here.
Parameters level (str) – a string used to set the print level, which may be ‘all’, ‘warn’, ‘error’
or ‘none’. Note that capitalization and extra letters in level are ignored, so ‘Warn’, ‘warnings’,
etc. will all set the mode to ‘warn’

GSASIIscriptable.SetupGeneral(data, dirname)
Helps initialize phase data.
From GSASIIphsGui.py, function of the same name. Minor changes for imports etc.

Author: Jackson O’Donnell (jacksonhodonnell .at. gmail.com)

GSASIIscriptable.add(args)
Implements the add command-line subcommand. This adds histograms and/or phases to GSAS-II project:

usage: GSASIIscriptable.py add [-h] [-d HISTOGRAMS [HISTOGRAMS ...]]
[-i IPARAMS [IPARAMS ...]]
[-hf HISTOGRAMFORMAT] [-p PHASES [PHASES ...]]
[-pf PHASEFORMAT] [-l HISTLIST [HISTLIST ...]]
filename

positional arguments:
filename the project file to open. Should end in .gpx

optional arguments:
-h, --help show this help message and exit
-d HISTOGRAMS [HISTOGRAMS ...], --histograms HISTOGRAMS [HISTOGRAMS ...]
list of datafiles to add as histograms
-i IPARAMS [IPARAMS ...], --iparams IPARAMS [IPARAMS ...]
instrument parameter file, must be one for every
histogram
(continues on next page)
-hf HISTOGRAMFORMAT, --histogramformat HISTOGRAMFORMAT
    format hint for histogram import. Applies to all histograms
-p PHASES [PHASES ...], --phases PHASES [PHASES ...]
    list of phases to add. phases are automatically associated with all histograms given.
-pf PHASEFORMAT, --phaseformat PHASEFORMAT
    format hint for phase import. Applies to all phases. Example: -pf CIF
-l HISTLIST [HISTLIST ...], --histlist HISTLIST [HISTLIST ...]
    list of histogram indices to associate with added phases. If not specified, phases are associated with all previously loaded histograms. Example: -l 2 3 4

GSASIIscriptable.blkSize = 256
Integration block size; 256 seems to be optimal, must be <=1024 (for polymask)

GSASIIscriptable.calcMaskMap(imgprms, mskprms)
    Computes the mask array for a set of image controls and mask parameters

GSASIIscriptable.calcThetaAzimMap(imgprms)
    Computes the array for theta-azimuth mapping for a set of image controls

GSASIIscriptable.create(args)
    Implements the create command-line subcommand. This creates a GSAS-II project, optionally adding histograms and/or phases:

    usage: GSASIIscriptable.py create [-h] [-d HISTOGRAMS [HISTOGRAMS ...]]
        [-i IPARAMS [IPARAMS ...]]
        [-p PHASES [PHASES ...]]
        filename

positional arguments:

    filename                the project file to create. should end in .gpx

optional arguments:

    -h, --help           show this help message and exit
    -d HISTOGRAMS [HISTOGRAMS ...], --histograms HISTOGRAMS [HISTOGRAMS ...]
        list of datafiles to add as histograms
    -i IPARAMS [IPARAMS ...], --iparams IPARAMS [IPARAMS ...]
        instrument parameter file, must be one for every histogram
    -p PHASES [PHASES ...], --phases PHASES [PHASES ...]
        list of phases to add. phases are automatically associated with all histograms given.

GSASIIscriptable.dump(args)
    Implements the dump command-line subcommand, which shows the contents of a GSAS-II project:


positional arguments:

    files

optional arguments:
-h, --help  show this help message and exit
-d, --histograms  list histograms in files, overrides --raw
-p, --phases  list phases in files, overrides --raw
-r, --raw  dump raw file contents, default

GSASIIscriptable.export(args)
Implements the export command-line subcommand: Exports phase as CIF:

usage: GSASIIscriptable.py export [-h] gpxfile phase exportfile

positional arguments:
  gpxfile  the project file from which to export
  phase    identifier of phase to export
  exportfile  the .cif file to export to

optional arguments:
  -h, --help  show this help message and exit

GSASIIscriptable.exportersByExtension = {}
Specifies the list of extensions that are supported for Powder data export

GSASIIscriptable.import_generic(filename, readerlist, fmthint=None, bank=None)
Attempt to import a filename, using a list of reader objects.

  Returns the first reader object which worked.

GSASIIscriptable.load_iprms(instfile, reader, bank=None)
Loads instrument parameters from a file, and edits the given reader.

  Returns a 2-tuple of (Iparm1, Iparm2) parameters

GSASIIscriptable.load_pwd_from_reader(reader, instprm, existingnames=[], bank=None)
Loads powder data from a reader object, and assembles it into a GSASII data tree.

  Returns (name, tree) - 2-tuple of the histogram name (str), and data

  Author: Jackson O’Donnell (jacksonhodonnell .at. gmail.com)

GSASIIscriptable.main()
The command-line interface for calling GSASIIscriptable as a shell command, where it is expected to be called as:

  python GSASIIscriptable.py <subcommand> <file.gpx> <options>

The following subcommands are defined:
  • create, see create()
  • add, see add()
  • dump, see dump()
  • refine, see refine()
  • export, export()
  • browse, see IPyBrowse()

  See also:

  create() add() dump() refine() export() IPyBrowse()
GSASIIscriptable_make_empty_project (author=None, filename=None)
Creates an dictionary in the style of GSASIIscriptable, for an empty project.

If no author name or filename is supplied, ‘no name’ and <current dir>/test_output.gpx are used , respectively.

Returns: project dictionary, name list

Author: Jackson O’Donnell (jacksonhodonnell .at. gmail.com)

GSASIIscriptable_refine (args)

**Implements the refine command-line subcommand:** conducts refinements on GSAS-II projects according to a JSON refinement dict:

```
usage: GSASIIscriptable.py refine [-h] gpxfile [refinements]

positional arguments:
  gpxfile the project file to refine
  refinements json file of refinements to apply. if not present refines file as-is
```

optional arguments:

```
-h, --help show this help message and exit
```

1.16 GSAS-II Misc Scripts

1.16.1 testDeriv: Check derivative computation

Use this to check derivatives used in structure least squares refinement against numerical values computed in this script.

To use set DEBUG=True in GSASIIstrMain.py (line 40, as of version 2546); run the least squares - zero cycles is sufficient. Do the “Save Results”; this will write the file testDeriv.dat in the local directory.

Then run this program to see plots of derivatives for all parameters refined in the last least squares. Shown will be numerical derivatives generated over all observations (including penalty terms) and the corresponding analytical ones produced in the least squares. They should match. Profiling is also done for function calculation & for the 1st selected derivative (rest should be the same).

```
testDeriv.main()
  Starts main application to compute and plot derivatives

class testDeriv.testDeriv (parent)

class testDeriv.testDerivmain (*args, **kwargs)
```

1.16.2 GSASIItestplot: Plotting for testDeriv

Plotting module used for script testDeriv.

```
class GSASIItestplot.Plot (parent, id=-1, dpi=None, **kwars)
  Creates a plotting window

class GSASIItestplot.PlotNotebook (id=1)
  creates a Wx application and a plotting notebook
```
1.16.3 scanCCD: reduce data from scanning CCD


```python
class scanCCD.main(Class)
    starts main application to merge data from scanning CCD

class scanCCD.scanCCD(parent)

    PlotXY (XY, newPlot=False, type='')
    simple plot of xy data, used for diagnostic purposes

class scanCCD.scanCCDmain(*args, **kwargs)
```

1.16.4 makeMacApp: Create Mac Applet

This script creates an AppleScript app bundle to launch GSAS-II. The app is created in the directory where the GSAS-II script (GSASII.py) is located. A softlink to Python is created, but is named GSAS-II, and placed inside the bundle so that GSAS-II shows up as the name of the app rather than Python in the menu bar, etc. A soft link named GSAS-II.py, referencing GSASII.py, is created so that appropriate menu items also are labeled with GSAS-II (but not the right capitalization, alas).

This has been tested with several versions of Python interpreters from Anaconda and does not require pythonw (Python.app). It tests to make sure that a wx python script will run inside Python and if not, it searches for a pythonw image and tries that.

Run this script with no arguments or with one optional argument, a reference to the GSASII.py, with a relative or absolute path. Either way the path will be converted via an absolute path. If no arguments are supplied, the GSASII.py script must be located in the same directory as this file.

```python
makeMacApp.AppleScript = ''
    Contains an AppleScript to start GSAS-II, launching Python and the GSAS-II python script

makeMacApp.RunPython(image, cmd)
    Run a command in a python image
```

1.16.5 makeBat: Create GSAS-II Batch File

This script creates a file named RunGSASII.bat and a desktop shortcut to that file. It registers the filetype .gpx so that the GSAS-II project files exhibit the GSAS-II icon and so that double-clicking on them opens them in GSAS-II.

Run this script with no arguments; the path to the GSASII.py file is assumed to be the the same as the path to the makeBat.py file and the path to Python is determined from the version of Python used to run this script.

1.16.6 makeLinux: Create Linux Shortcuts

This script creates a menu entry for Gnome & KDE desktop managers and puts it in a place where it can be “indexed.”

This is a work in progress as I learn more about shortcuts in Linux.

Run this script with one optional argument, the path to the GSASII.py The script path may be specified relative to the current path or given an absolute path, but will be accessed via an absolute path. If no arguments are supplied, the GSASII.py script is assumed to be in the same directory as this file.
1.16.7 makeTutorial: Make Tutorial Web Page

Creates an HTML page (GSASII/help/Tutorials.html) listing all the tutorials in GSASIIctrlGUI. tutorialIndex. Run this after adding a new tutorial to that index.


A list of videos that are in box, since I don’t know how to check if they are present anymore

1.16.8 unit_tests: Self-test Module

A script that can be run to test a series of self-tests in GSAS-II. At present, only modules GSASIIspc and GSASIIlattice have self-tests.

unit_tests.test_GSASIIlattice()
Test registered self-tests in GSASIIlattice. Takes no input and returns nothing. Throws an Exception if a test fails.

unit_tests.test_GSASIIspc()
Test registered self-tests in GSASIIspc. Takes no input and returns nothing. Throws an Exception if a test fails.

1.17 GSAS-II Import Modules

Imports are implemented by deriving a class from GSASIIobj.ImportPhase, GSASIIobj.ImportStructFactor, GSASIIobj.ImportPowderData, GSASIIobj.ImportSmallAngleData, GSASIIobj.ImportReflectometryData, GSASIIobj.ImportPDFData, or GSASIIobj.ImportImage (which are in turn derived from GSASIIobj.ImportBaseclass) to implement import of a phase, a single crystal or a powder dataset, respectively. Module file names (G2phase_, G2pwd_, and G2sfact_, etc.) are used to determine which menu an import routine should be placed into. (N.B. this naming was an unnecessary choice; importer types could be determined from the base class.)

Most importers are listed below by type (provided this documentation is up to date), but note that since modules may be loaded from anywhere in the path, your installation could have locally-defined importers as well.

1.17.1 Writing an Import Routine

When writing a import routine, one should create a new class derived from GSASIIobj.ImportPhase, GSASIIobj.ImportStructFactor, GSASIIobj.ImportPowderData, GSASIIobj.ImportSmallAngleData, GSASIIobj.ImportReflectometryData, GSASIIobj.ImportPDFData, or GSASIIobj.ImportImage. As described below, all these classes will implement an __init__() and a Reader() method, and most will supply a ContentsValidator() method, too. See the appropriate class documentation for details on what values each type of Reader() should set.

__init__()

The __init__ method will follow standard boilerplate:

```python
def __init__(self):
    super(self.__class__,self).__init__( # fancy way to self-reference
        extensionlist=('ext1','ext2'),
        strictExtension=True,
        formatName = 'example image',
    )
```

(continues on next page)
The first line in the __init__ method calls the parent class __init__ method with the following parameters:

- **extensionlist**: a list of extensions that may be used for this type of file.
- **strictExtension**: Should be True if only files with extensions in extensionlist are allowed; False if all file types should be offered in the file browser. Also if False, the import class will be used on all files when “guess from format” is tried, though readers with matching extensions will be tried first. It is a very good idea to supply a ContentsValidator method when strictExtension is False.
- **formatName**: a string to be used in the menu. Should be short.
- **longFormatName**: a longer string to be used to describe the format in help.

Note that if an importer detects a condition which prevents its use, for example because a required Python package is not present, it can set the value of self.UseReader to False. Another possible use for this would be an importer that requires a network connection to a remote site. Setting self.UseReader to False must be done in the __init__ method and will prevent the importer from being used or included in the expected menu.

### Reader()

The class must supply a Reader method that actually performs the reading. All readers must have at a minimum these arguments:

```python
def Reader(self, filename, filepointer, ParentFrame, **unused):
```

where the arguments have the following uses:

- **filename**: a string with the name of the file being read
- **filepointer**: a file object (created by `open()`) that accesses the file and is points to the beginning of the file when Reader is called.
- **ParentFrame**: a reference to the main GSAS-II (tree) windows, for the unusual Reader routines that will create GUI windows to ask questions. The Reader should do something reasonable such as take a reasonable default if ParentFrame is None, which indicates that GUI should not be accessed.

In addition, the following keyword parameters are defined that Reader routines may optionally use:

- **buffer**: a dict that can be used to retain information between repeated calls of the routine
- **blocknum**: counts the number of times that a reader is called, to be used with files that contain more than one set of data (e.g. GSAS .gsa/.fxye files with multiple banks or image files with multiple images.)
- **usedRanIdList**: a list of previously used random Id values that can be checked to determine that a value is unique.

As an example, the buffer dict is used in CIF reading to hold the parsed CIF file so that it can be reused without having to reread the file from scratch.

### Reader return values

The Reader routine should return the value of True if the file has been read successfully. Optionally, use self.warnings to indicate any problems.
If the file cannot be read, the Reader routine should return False or raise a `GSASIIobj.ImportBaseclass ImportError` exception. (Why either? Sometimes an exception is the easiest way to bail out of a called routine.) Place text in `self.errors` and/or use:

```
ImportException('Error message')
```

to give the user information on what went wrong during the reading.

**self.warnings**

Use `self.warnings` to indicate any information that should be displayed to the user if the file is read successfully, but perhaps not completely or additional settings will need to be made.

**self.errors**

Use `self.errors` to give the user information on where and why a read error occurs in the file. Note that text supplied with the `raise` statement will be appended to `self.errors`.

**self.repeat**

Set `self.repeat` to True (the default is False) if a Reader should be called again to after reading to indicate that more data may exist in the file to be read. This is used for reading multiple powder histograms or multiple images from a single file. Variable `self.repeatcount` is used to keep track of the block numbers.

**support routines**

Note that GSASIIIO supplies three routines, `BlockSelector()` `MultipleBlockSelector()` and `MultipleChoiceSelector()` that are useful for selecting amongst one or more datasets (and perhaps phases) or data items for Reader() routines that may encounter more than one set of information in a file.

**ContentsValidator()**

Defining a `ContentsValidator` method is optional, but is usually a good idea, particularly if the file extension is not a reliable identifier for the file type. The intent of this routine is to take a superficial look at the file to see if it has the expected characteristics of the expected file type. For example, are there numbers in the expected places?

This routine is passed a single argument:

- `filepointer`: a file object (created by `open()`) that accesses the file and is points to the beginning of the file when ContentsValidator is called.

Note that `GSASIIobj.ImportBaseclass.CIFValidator()` is a ContentsValidator for validating CIF files.

**Reinitialize()**

Import classes are substantiated only once and are used as needed. This means that if something needs to be initialized before the Reader() will be called to read a new file, it must be coded. The ReInitialize() method is provided for this and it is always called before the ContentsValidator method is called. Use care to call the parent class ReInitialize() method, if this is overridden.
ContentsValidator return values

The ContentsValidator routine should return the value of True if the file appears to match the type expected for the class.

If the file cannot be read by this class, the routine should return False. Preferably one will also place text in self.errors to give the user information on what went wrong during the reading.

1.17.2 Phase Import Routines

Phase import routines are classes derived from GSASIIobj.ImportPhase. They must be found in files named G2phase*.py that are in the Python path and the class must override the __init__ method and add a Reader method. The distributed routines are:

**Module G2phase: PDB, .EXP & JANA m40,m50**

A set of short routines to read in phases using routines that were previously implemented in GSAS-II: PDB, GSAS .EXP and JANA m40-m50 file formats

**class G2phase.EXP_ReaderClass**

Routine to import Phase information from GSAS .EXP files

ContentsValidator (filename)

Look for a VERSION tag in 1st line

ReadEXPPhase (G2frame, filepointer)

Read a phase from a GSAS .EXP file.

Reader (filename, ParentFrame=None, usedRanIdList=[], **unused)

Read a phase from a GSAS .EXP file using ReadEXPPhase()

**class G2phase.JANA_ReaderClass**

Routine to import Phase information from a JANA2006 file

ContentsValidator (filename)

Taking a stab a validating a .m50 file (look for cell & at least one atom)

ReadJANAPhase (filename, parent=None)

Read a phase from a JANA2006 m50 & m40 files.

Reader (filename, ParentFrame=None, **unused)

Read a m50 file using ReadJANAPhase()

**class G2phase.PDB_ReaderClass**

Routine to import Phase information from a PDB file

ContentsValidator (filename)

Taking a stab a validating a PDB file (look for cell & at least one atom)

ReadPDBPhase (filename, parent=None)

Read a phase from a PDB file.

Reader (filename, ParentFrame=None, **unused)

Read a PDF file using ReadPDBPhase()

**class G2phase.PDF_ReaderClass**

Routine to import Phase information from ICDD PDF Card files

ContentsValidator (filename)

Look for a str tag in 1st line
ReadPDFPhase \((G2\text{frame}, fp)\)
Read a phase from a ICDD .str file.

Reader \((filename, ParentFrame=None, **unused)\)
Read phase from a ICDD .str file using ReadPDFPhase().

**Module G2phase\_GPX: Import phase from GSAS-II project**

Copies a phase from another GSAS-II project file into the current project.

class G2phase\_GPX.PhaseReaderClass
Opens a .GPX file and pulls out a selected phase

ContentsValidator \((filename)\)
Test if the 1st section can be read as a cPickle block, if not it can’t be .GPX!

Reader \((filename, ParentFrame=None, **unused)\)
Read a phase from a .GPX file. Does not (yet?) support selecting and reading more than one phase at a time.

**Module G2phase\_CIF: Coordinates from CIF**

Parses a CIF using PyCifRW from James Hester and pulls out the structural information.

If a CIF generated by ISODISTORT is encountered, extra information is added to the phase entry and constraints are generated.

class G2phase\_CIF.CIFPhaseReader
Implements a phase importer from a possibly multi-block CIF file

ContentsValidator \((filename)\)
This routine will attempt to determine if the file can be read with the current format. This will typically be overridden with a method that takes a quick scan of [some of] the file contents to do a “sanity” check if the file appears to match the selected format. the file must be opened here with the correct format (binary/text)

ISODISTORT\_proc \((blk, atomlbllist, ranIdlookup)\)
Process ISODISTORT items to create constraints etc.

**Module G2phase\_INS: Import phase from SHELX INS file**

Copies a phase from SHELX ins file into the current project.

class G2phase\_INS.PhaseReaderClass
Opens a .INS file and pulls out a selected phase

ContentsValidator \((filename)\)
Test if the ins file has a CELL record

ReadINSPhase \((filename, parent=None)\)
Read a phase from a INS file.

Reader \((filename, filepointer, ParentFrame=None, **unused)\)
Read a ins file using ReadINSPhase().
1.17.3 Powder Data Import Routines

Powder data import routines are classes derived from `GSASIIobj.ImportPowderData`. They must be found in files named `G2pwd*.py` that are in the Python path and the class must override the `__init__` method and add a `Reader` method.

The distributed routines are:

**Module G2pwd_GPX: GSAS-II projects**

Routine to import powder data from GSAS-II .gpx files

```python
class G2pwd_GPX.GSAS2_ReaderClass:
    Routines to import powder data from a GSAS-II file. This should work to pull data out from an out of date .GPX file as long as the details of the histogram data itself don’t change.

    ContentsValidator(filename)
    Test if the 1st section can be read as a cPickle block, if not it can’t be .GPX!

    Reader(filename, ParentFrame=None, **kwargs)
    Read a dataset from a .GPX file. If multiple datasets are requested, use self.repeat and buffer caching.
```

**Module G2pwd_fxye: GSAS data files**

Routine to read in powder data in a variety of formats that are defined for GSAS.

```python
class G2pwd_fxye.GSAS_ReaderClass:
    Routines to import powder data from a GSAS file.

    ContentsValidator(filename)
    Validate by checking to see if the file has BANK lines & count them.

    Reader(filename, ParentFrame=None, **kwargs)
    Read a GSAS (old formats) file of type FXY, FXYE, ESD or STD types. If multiple datasets are requested, use self.repeat and buffer caching.

G2pwd_fxye.sfloat(S)
    convert a string to a float, treating an all-blank string as zero

G2pwd_fxye.sint(S)
    convert a string to an integer, treating an all-blank string as zero
```

**Module G2pwd_xye: Topas .xye data**

Routine to read in powder data from a Topas-compatible .xye file

```python
class G2pwd_xye.xye_ReaderClass:
    Routines to import powder data from a .xye/.chi file.

    ContentsValidator(filename)
    Look through the file for expected types of lines in a valid Topas file.

    Reader(filename, ParentFrame=None, **unused)
    Read a Topas file
```
Module G2pwd_CIF: CIF powder data

Routine to read in powder data from a CIF.

class G2pwd_CIF.CIFpwdReader
    Routines to import powder data from a CIF file

    ContentsValidator (filename)
        Use standard CIF validator

    Reader (filename, ParentFrame=None, **kwarg)
        Read powder data from a CIF. If multiple datasets are requested, use self.repeat and buffer caching.

Module G2pwd_BrukerRAW: Bruker v.1-v.3 .raw data

Routine to read in powder data from a Bruker versions 1-3 .raw file

class G2pwd_BrukerRAW.raw_ReaderClass
    Routines to import powder data from a binary Bruker .RAW file

    ContentsValidator (filename)
        Look through the file for expected types of lines in a valid Bruker RAW file

    Reader (filename, ParentFrame=None, **kwarg)
        Read a Bruker RAW file

Module G2pwd_FP: FullProf .dat data

Routine to read in powder data from a FullProf .dat file

class G2pwd_FP.fp_ReaderClass
    Routines to import powder data from a FullProf 1-10 column .dat file

    ContentsValidator (filename)
        Look through the file for expected types of lines in a valid FullProf file

    Reader (filename, ParentFrame=None, **unused)
        Read a FullProf file

class G2pwd_Panalytical.Panalytical_ReaderClass
    Routines to import powder data from a Pananalytical.xml (xml) file.

    ContentsValidator (filename)
        This routine will attempt to determine if the file can be read with the current format. This will typically be
        overridden with a method that takes a quick scan of [some of] the file contents to do a “sanity” check if the
        file appears to match the selected format. the file must be opened here with the correct format (binary/text)

    Reader (filename, ParentFrame=None, **kwarg)
        Read a Panalytical .xrdml (.xml) file; already in self.root

Module G2pwd_csv: Read Excel .csv data

Routine to read in powder data from Excel type comma separated variable column-oriented variable

class G2pwd_csv.csv_ReaderClass
    Routines to import powder data from a .xye file
### ContentsValidator (filename)

This routine will attempt to determine if the file can be read with the current format. This will typically be overridden with a method that takes a quick scan of [some of] the file contents to do a “sanity” check if the file appears to match the selected format. The file must be opened here with the correct format (binary/text).

### Reader (filename, ParentFrame=None, **unused)

Read a csv file

```python
class G2pwd_rigaku.Rigaku_rasReaderClass
Routines to import powder data from a Rigaku .ras file with multiple scans. All scans will be imported as individual PWDR entries.

### ContentsValidator (filename)

This routine will attempt to determine if the file can be read with the current format. This will typically be overridden with a method that takes a quick scan of [some of] the file contents to do a “sanity” check if the file appears to match the selected format. The file must be opened here with the correct format (binary/text).

### Reader (filename, ParentFrame=None, **kwarg)

Read a Rigaku .ras file

```python
class G2pwd_rigaku.Rigaku_txtReaderClass
Routines to import powder data from a Rigaku .txt file with an angle and then 1 or 11(!) intensity values on the line. The example file is proceeded with 10 of blank lines, but I have assumed they could be any sort of text. This code should work with an angle and any number of intensity values/line as long as the number is the same on each line. The step size may not change. The number of comment lines can also change, but should not appear to be intensity values (numbers only).

### ContentsValidator (filename)

This routine will attempt to determine if the file can be read with the current format. This will typically be overridden with a method that takes a quick scan of [some of] the file contents to do a “sanity” check if the file appears to match the selected format. The file must be opened here with the correct format (binary/text).

### Reader (filename, ParentFrame=None, **kwarg)

Read a Rigaku .txt file

#### 1.17.4 Single Crystal Data Import Routines

Single crystal data import routines are classes derived from `GSASIIobj.ImportStructFactor`. They must be found in files named `G2sfact*.py` that are in the Python path and the class must override the `__init__` method and add a `Reader` method. The distributed routines are:

**Module G2sfact: simple HKL import**

Read structure factors from a simple hkl file. Two routines are provided to read from files containing F or F² values.

```python
G2sfact.ColumnValidator (parent, filepointer, nCol=5)
Validate a file to check that it contains columns of numbers

class G2sfact.HKLF_ReaderClass
Routines to import F, sig(F) reflections from a HKLF file

### ContentsValidator (filename)

Make sure file contains the expected columns on numbers

### Reader (filename, ParentFrame=None, **unused)

Read the file
class G2sfact.HKLMF_ReaderClass
   Routines to import F, reflections from a REMOS HKLMF file
   ContentsValidator (filename)
      Make sure file contains the expected columns on numbers
   Reader (filename, ParentFrame=None, **unused)
      Read the file

class G2sfact.M90_ReaderClass
   Routines to import F**2, sig(F**2) reflections from a JANA M90 file
   ContentsValidator (filename)
      Discover how many columns are in the m90 file - could be 9-12 depending on satellites
   Reader (filename, filepointer, ParentFrame=None, **unused)
      Read the file

class G2sfact.NIST hb3a_INT_ReaderClass
   Routines to import neutron CW F**2, sig(F**2) reflections from a NIST hb3a int file
   ContentsValidator (filename)
      Make sure file contains the expected columns on numbers & count number of data blocks - “Banks”
   Reader (filename, filepointer, ParentFrame=None, **unused)
      Read the file

class G2sfact.NT_HKLF2_ReaderClass
   Routines to import neutron TOF F**2, sig(F**2) reflections from a HKLF file
   ContentsValidator (filename)
      Make sure file contains the expected columns on numbers & count number of data blocks - “Banks”
   Reader (filename, ParentFrame=None, **unused)
      Read the file

class G2sfact.NT_JANA2K_ReaderClass
   Routines to import neutron TOF F**2, sig(F**2) reflections from a JANA2000 file
   ContentsValidator (filename)
      Make sure file contains the expected columns on numbers & count number of data blocks - “Banks”
   Reader (filename, filepointer, ParentFrame=None, **unused)
      Read the file

class G2sfact.SHELX4_ReaderClass
   Routines to import F**2, sig(F**2) reflections from a Shelx HKLF 4 file
   ContentsValidator (filename)
      Make sure file contains the expected columns on numbers
   Reader (filename, ParentFrame=None, **unused)
      Read the file

class G2sfact.SHELX5_ReaderClass
   Routines to import F**2, sig(F**2) twin/incommensurate reflections from a fixed format SHELX HKLF5 file
   ContentsValidator (filename)
      Discover how many columns before F^2 are in the SHELX HKLF5 file - could be 3-6 depending on satellites
   Reader (filename, ParentFrame=None, **unused)
      Read the file
**Module G2sfact.CIF: CIF import**

Read structure factors from a CIF reflection table.

**class** G2sfact.CIF.CIFhklReader
   Routines to import Phase information from a CIF file

**ContentsValidator (filename)**
   Use standard CIF validator

**Reader (filename, ParentFrame=None, **kwargs)**
   Read single crystal data from a CIF. If multiple datasets are requested, use self.repeat and buffer caching.

### 1.17.5 Small Angle Scattering Data Import Routines

Small angle scattering data import routines are classes derived from .GSASIIobj.ImportSmallAngle. They must be found in files named G2sad*.py that are in the Python path and the class must override the __init__ method and add a Reader method. The distributed routines are:

**Module G2sad_xy: read small angle data**

Routines to read in small angle data from an .xye type file, with two-theta or Q steps.

**class** G2sad_xy.txt_CWNeutronReaderClass
   Routines to import neutron CW q SAXD data from a .nsad or .ndat file

**ContentsValidator (filename)**
   Look through the file for expected types of lines in a valid q-step file

**class** G2sad_xy.txt_XRayReaderClass
   Routines to import X-ray q SAXD data from a .xsad or .xdat file

**ContentsValidator (filename)**
   Look through the file for expected types of lines in a valid q-step file

**class** G2sad_xy.txt_nmCWNeutronReaderClass
   Routines to import neutron CW q in nm-1 SAXD data from a .nsad or .ndat file

**ContentsValidator (filename)**
   Look through the file for expected types of lines in a valid q-step file

**class** G2sad_xy.txt_nmXRayReaderClass
   Routines to import X-ray q SAXD data from a .xsad or .xdat file, q in nm-1

**ContentsValidator (filename)**
   Look through the file for expected types of lines in a valid q-step file
1.17.6 Image Import Routines

Image import routines are classes derived from `GSASIobj.ImportImage`. See `Writing a Import Routine` for general information on importers and the `GSASIobj.ImportImage` for information on what class variables a reader should set. Image importers must be found in files named `G2img*.py` that are in the Python path and the class must override the `__init__` method and add a `Reader` method.

The distributed routines are:

**Module G2img_ADSC: .img image file**

```python
class G2img_ADSC.ADSC_ReaderClass
    Reads an ADSC .img file
    ContentsValidator (filename)
        no test at this time
G2img_ADSC.GetData (filename, imageOnly=False)
    Read an ADSC image file
```

**Module G2img_EDF: .edf image file**

```python
class G2img_EDF.EDF_ReaderClass
    Routine to read a Read European detector data .edf file. This is a particularly nice standard.
    ContentsValidator (filename)
        no test used at this time
G2img_EDF.GetData (filename, imageOnly=False)
    Read European detector data edf file
```

**Module G2img_SumG2: Python pickled image**

```python
class G2img_SumG2.G2_ReaderClass
    Routine to read an image that has been pickled in Python. Images in this format are created by the “Sum image data” command. At least for now, only one image is permitted per file.
    ContentsValidator (filename)
        test by trying to unpickle (should be quick)
    Reader (filename, ParentFrame=None, **unused)
        Read using cPickle
```

**Module G2img_GE: summed GE image file**

Read data from General Electric angiography x-ray detectors, primarily as used at APS 1-ID. This shows an example of an importer that will handle files with more than a single image.

```python
class G2img_GE.GE_ReaderClass
    Routine to read a GE image, typically from APS Sector 1.
    The image files may be of form .geX (where X is ‘ ‘, 1, 2, 3, 4 or 5), which is a raw image from the detector. These files may contain more than one image and have a rudimentary header. Files with extension .sum or .cor are 4 byte integers/pixel, one image/file. Files with extension .avg are 2 byte integers/pixel, one image/file.
    ContentsValidator (filename)
        no test used at this time
    Reader (filename, ParentFrame=None, **unused)
        Read using cPickle
```

1.17. GSAS-II Import Modules 223
ContentsValidator (filename)
  just a test on file size

Reader (filename, ParentFrame=None, **kwarg)
  Read using GE file reader, GetGEsumData()

class G2img_GE.GEsum_ReaderClass
  Routine to read multiple GE images & sum them, typically from APS Sector 1.
  The image files may be of form .geX (where X is ‘ ’, 1, 2, 3, 4 or 5), which is a raw image from the detector.
  These files may contain more than one image and have a rudimentary header. Files with extension .sum or .cor
  are 4 byte integers/pixel, one image/file. Files with extension .avg are 2 byte integers/pixel, one image/file.

ContentsValidator (filename)
  just a test on file size

Reader (filename, ParentFrame=None, **kwarg)
  Read using GE file reader, GetGEsumData()

G2img_GE.GetGEsumData (self, filename, imagenum=1, sum=False)
  Read G.E. detector images from various files as produced at 1-ID and with Detector Pool detector. Also sums
  multiple image files if desired

Module G2img_MAR: MAR image files

G2img_MAR.GetMAR345Data (filename, imageOnly=False)
  Read a MAR-345 image plate image

class G2img_MAR.MAR_ReaderClass
  Routine to read several MAR formats, .mar3450,.mar2300,.mar2560

ContentsValidator (filename)
  no test at this time

Module G2img_Rigaku: .stl image file

G2img_Rigaku.GetRigaku (filename, imageOnly=False)
  Read Rigaku R-Axis IV image file

class G2img_Rigaku.Rigaku_ReaderClass
  Routine to read a Rigaku R-Axis IV image file.

ContentsValidator (filename)
  Test by checking if the file size makes sense.

Module G2img_1TIF: Tagged-image File images

Routine to read an image in Tagged-image file (TIF) format as well as a variety of slightly incorrect pseudo-TIF
formats used at instruments around the world. Note that the name G2img_1TIF is used so that this file will sort to
the top of the image formats and thus show up first in the menu. (It is the most common, alas).

G2img_1TIF.GetTifData (filename)
  Read an image in a pseudo-tif format, as produced by a wide variety of software, almost always incorrectly in
  some way.
class G2img_1TIF.TIF_ReaderClass
Reads TIF files using a routine (`GetTifData()`) that looks for files that can be identified from known instruments and will correct for slightly incorrect TIF usage. If that routine fails, it will be read with a standard TIF reader, which can handle compression and other things less commonly used at beamlines.

```
ContentsValidator(filename)
   Does the header match the required TIF header?

Reader(filename, ParentFrame=None, **unused)
   Read the TIF file using `GetTifData()`. If that fails, use `scipy.misc.imread()` and give the user a chance to edit the likely wrong default image parameters.
```

Module G2img_png: png image file
Routine to read an image in .png (Portable Network Graphics) format. For now, the only known use of this is with converted Mars Rover (CheMin) tif files, so default parameters are for that.

class G2img_CheMin.png_ReaderClass
Reads standard PNG images; parameters are set to those of the Mars Rover (CheMin) diffractometer.

```
ContentsValidator(filename)
   no test at this time

Reader(filename, ParentFrame=None, **unused)
   Reads using standard scipy PNG reader
```

Module G2img_CBF: .cbf cif image file

class G2img_CBF.CBF_ReaderClass
Routine to read a Read cif image data .cbf file. This is used by Pilatus.

```
ContentsValidator(filename)
   no test used at this time

Reader(filename, ParentFrame=None, **unused)
   Read using Bob’s routine `GetCbfData()`
```

G2img_CBF.GetCbfData(self, filename)
Read cif binarydetector data cbf file

Module G2img_HDF5: summed HDF5 image file
Reads all images found in a HDF5 file.

class G2img_HDF5.HDF5_Reader
Routine to read a HD5 image, typically from APS Sector 6. B. Frosik/SDM.

```
ContentsValidator(filename)
   Test if valid by seeing if the HDF5 library recognizes the file.

Reader(filename, ParentFrame=None, **kwarg)
   Scan file structure using `visit()` and map out locations of image(s) then read one image using `readDataset()`. Save map of file structure in buffer arg, if used.

readDataset(fp, imagenum=1)
   Read a specified image number from a file
```
visit \((fp)\)
Recursively visit each node in an HDF5 file. For nodes ending in ‘data’ look at dimensions of contents. If the shape is length 2 or 4 assume an image and index in self.buffer[‘imagemap’]

**Module G2img_SFRM: .sfrm image file**

G2img_SFRM.GetSFRMData \((self, filename)\)
Read cbf compressed binarydetector data .sfrm file

class G2img_SFRM.SFRM_ReaderClass
Routine to read a Read Bruker Advance image data .sfrm file.

ContentsValidator \((filename)\)
no test used at this time

Reader \((filename, ParentFrame=None, **unused)\)
Read using Bob’s routine GetSFRMData()

### 1.17.7 PDF Import Routines

PDF import routines are classes derived from GSASIIobj.ImportPDFData. See Writing a Import Routine for general information on importers.

The distributed routines are:

**Module G2pdf_gr: read PDF G(R) data**

Routines to read in G(R) data from an .gr type file, with Angstrom steps.

class G2pdf_gr.txt_PDFReaderClass
Routines to import PDF G(R) data from a .gr file

ContentsValidator \((filename)\)
Look through the file for expected types of lines in a valid r-step file

class G2pdf_gr.txt_PDFReaderClassG
Routines to import PDF G(R) data from a .dat file

ContentsValidator \((filename)\)
Look through the file for expected types of lines in a valid r-step file

### 1.17.8 Reflectometry Import Routines

Reflectometry import routines are classes derived from GSASIIobj.ImportReflectometryData. See Writing a Import Routine for general information on importers.

The distributed routines are:

**Module G2rfd_xye: read reflectometry data**

Routines to read in reflectometry data from an .xye type file, with two-theta or Q steps.

class G2rfd_xye.txt_NeutronReaderClass
Routines to import neutron q REFID data from a .nrfd or .ndat file
ContentsValidator(filename)
Look through the file for expected types of lines in a valid q-step file

class G2rfd_xye.txt.XRayReaderClass
Routines to import X-ray q REFD data from a .xrfd or .xdat file

ContentsValidator(filename)
Look through the file for expected types of lines in a valid q-step file

class G2rfd_xye.txt.XRayThetaReaderClass
Routines to import X-ray theta REFD data from a .xtrfd or .xtdat file

ContentsValidator(filename)
Look through the file for expected types of lines in a valid q-step file

1.18 GSAS-II Export Modules

Exports are implemented by deriving a class from GSASIIIO.ExportBaseClass. Initialization of self.exporttype determines the type of export that will be performed ('project', 'phase', 'single', 'powder', 'image', 'map' or (someday) 'pdf') and of self.multiple determines if only a single phase, data set, etc. can be exported at a time (when False) or more than one can be selected.

Powder export routines may optionally define a Writer() method that accepts the histogram tree name as well as a file name to be written. This allows ExportPowder() to use the exporter independent of the GUI.

1.18.1 Module G2export_examples: Examples

Code to demonstrate how GSAS-II data export routines are created. The classes defined here, ExportPhaseText, ExportSingleText, ExportPowderReflText, and ExportPowderText each demonstrate a different type of export. Also see G2export_map.ExportMapASCII for an example of a map export.

class G2export_examples.ExportPhaseText(G2frame)
Used to create a text file for a phase

Parameters G2frame(wx.Frame) – reference to main GSAS-II frame

Exporter(event=None)
Export a phase as a text file

class G2export_examples.ExportPowderReflText(G2frame)
Used to create a text file of reflections from a powder data set

Parameters G2frame(wx.Frame) – reference to main GSAS-II frame

Exporter(event=None)
Export a set of powder reflections as a text file

class G2export_examples.ExportPowderText(G2frame)
Used to create a text file for a powder data set

Parameters G2frame(wx.Frame) – reference to main GSAS-II frame

Exporter(event=None)
Export a set of powder data as a text file

class G2export_examples.ExportSingleText(G2frame)
Used to create a text file with single crystal reflection data skips user rejected & space group extinct reflections

Parameters G2frame(wx.Frame) – reference to main GSAS-II frame
Exporter(event=None)
Export a set of single crystal data as a text file

1.18.2 Module G2export_csv: Spreadsheet export

Code to create .csv (comma-separated variable) files for GSAS-II data export to a spreadsheet program, etc.

class G2export_csv.ExportMultiPowderCSV(G2frame)
Used to create a csv file for a stack of powder data sets suitable for display purposes only; no y-calc or weights are exported only x & y-obs :param wx.Frame G2frame: reference to main GSAS-II frame

Exporter(event=None)
Export a set of powder data as a single csv file

class G2export_csv.ExportPhaseCSV(G2frame)
Used to create a csv file for a phase

Parameters G2frame(wx.Frame) – reference to main GSAS-II frame

Exporter(event=None)
Export a phase as a csv file

class G2export_csv.ExportPowderCSV(G2frame)
Used to create a csv file for a powder data set

Parameters G2frame(wx.Frame) – reference to main GSAS-II frame

Exporter(event=None)
Export a set of powder data as a csv file

class G2export_csv.ExportPowderRef1CSV(G2frame)
Used to create a csv file of reflections from a powder data set

Parameters G2frame(wx.Frame) – reference to main GSAS-II frame

Exporter(event=None)
Export a set of powder reflections as a csv file

class G2export_csv.ExportSASDCSV(G2frame)
Used to create a csv file for a small angle data set

Parameters G2frame(wx.Frame) – reference to main GSAS-II frame

Exporter(event=None)
Export a set of small angle data as a csv file

class G2export_csv.ExportSingleCSV(G2frame)
Used to create a csv file with single crystal reflection data

Parameters G2frame(wx.Frame) – reference to main GSAS-II frame

Exporter(event=None)
Export a set of single crystal data as a csv file

class G2export_csv.ExportStrainCSV(G2frame)
Used to create a csv file with single crystal reflection data

Parameters G2frame(wx.Frame) – reference to main GSAS-II frame

Exporter(event=None)
Export a set of single crystal data as a csv file

G2export_csv.WriteList(obj, headerItems)
Write a CSV header
Parameters

- **obj (object)** – Exporter object
- **headerItems (list)** – items to write as a header

### 1.18.3 Module G2export_PDB: Macromolecular export

Code to export a phase into the venerated/obsolete (pick one) ASCII PDB format. Also defines exporter `ExportPhaseCartXYZ` which writes atom positions in orthogonal coordinates for a phase.

```python
class G2export_PDB.ExportPhaseCartXYZ(G2frame):
    Used to create a Cartesian XYZ file for a phase
    Parameters G2frame (wx.Frame) – reference to main GSAS-II frame
    Exporter (event=None)
    Export as a XYZ file

class G2export_PDB.ExportPhasePDB(G2frame):
    Used to create a PDB file for a phase
    Parameters G2frame (wx.Frame) – reference to main GSAS-II frame
    Exporter (event=None)
    Export as a PDB file
```

### 1.18.4 Module G2export_image: 2D Image data export

Demonstrates how an image is retrieved and written. Uses a SciPy routine to write a PNG format file.

```python
class G2export_image.ExportImagePNG(G2frame):
    Used to create a PNG file for a GSAS-II image
    Parameters G2frame (wx.Frame) – reference to main GSAS-II frame
    Exporter (event=None)
    Export an image
```

### 1.18.5 Module G2export_map: Map export

Code to write Fourier/Charge-Flip atomic density maps out in formats that can be read by external programs. At present a GSAS format that is supported by FOX and DrawXTL (ExportMapASCII) and the CCP4 format that is used by COOT (ExportMapCCP4) are implemented.

```python
class G2export_map.ExportMapASCII(G2frame):
    Used to create a text file for a phase
    Parameters G2frame (wx.Frame) – reference to main GSAS-II frame
    Exporter (event=None)
    Export a map as a text file

class G2export_map.ExportMapCCP4(G2frame):
    Used to create a text file for a phase
    Parameters G2frame (wx.Frame) – reference to main GSAS-II frame
    Exporter (event=None)
    Export a map as a text file
```
Write(\texttt{data, dtype})
write a line of output, attaching a line-end character

Parameters \texttt{line} (\texttt{str}) – the text to be written.

1.18.6 Module \texttt{G2export_shelx}: Examples

Code to export coordinates in the SHELX .ins format (as best as I can makes sense of it).

\texttt{class G2export_shelx.ExportPhaseShelx(G2frame)}

Used to create a SHELX .ins file for a phase

Parameters \texttt{G2frame} (\texttt{wx.Frame}) – reference to main GSAS-II frame

\texttt{Exporter(event=None)}

Export as a SHELX .ins file

1.18.7 Module \texttt{G2export_CIF}: CIF Exports

This implements a complex exporter \texttt{ExportCIF} that can implement an entire project in a complete CIF intended for submission as a publication. In addition, there are three subclasses of \texttt{ExportCIF: ExportProjectCIF, ExportPhaseCIF} and \texttt{ExportDataCIF} where extra parameters for the \texttt{_Exporter()} determine if a project, single phase or data set are written.

\texttt{G2export_CIF.CIF2dict(cf)}
copy the contents of a CIF out from a PyCifRW block object into a dict

Returns \texttt{cifblk, loopstructure} where \texttt{cifblk} is a dict with CIF items and \texttt{loopstructure} is a list of lists that defines which items are in which loops.

\texttt{class G2export_CIF.CIFdefHelp(parent, msg, helpwin, helptxt)}

Create a help button that displays help information on the current data item

Parameters

• \texttt{parent} – the panel which will be the parent of the button

• \texttt{msg} (\texttt{str}) – the help text to be displayed

• \texttt{helpwin} (\texttt{wx.Dialog}) – Frame for CIF editing dialog

• \texttt{helptxt} (\texttt{wx.TextCtrl}) – TextCtrl where help text is placed

\texttt{class G2export_CIF.CIFtemplateSelect(frame, panel, tmplate, G2dict, repaint, title, default-name="")}

Create a set of buttons to show, select and edit a CIF template

Parameters

• \texttt{frame} – \texttt{wx.Frame} object of parent

• \texttt{panel} – \texttt{wx.Panel} object where widgets should be placed

• \texttt{tmplate} (\texttt{str}) – one of ‘publ’, ‘phase’, or ‘instrument’ to determine the type of template

• \texttt{G2dict} (\texttt{dict}) – GSAS-II dict where CIF should be placed. The key “CIF_template” will be used to store either a list or a string. If a list, it will contain a dict and a list defining loops. If an str, it will contain a file name.

• \texttt{repaint} (\texttt{function}) – reference to a routine to be called to repaint the frame after a change has been made
• **title**(str) – A line of text to show at the top of the window

• **defaultname**(str) – specifies the default file name to be used for saving the CIF.

**class** G2export_CIF>EditCIFpanel(parent, cifblk, loopstructure, cifdic={}, OKbuttons=[], **kw)

Creates a scrolled panel for editing CIF template items

**Parameters**

• **parent**(wx.Frame) – parent frame where panel will be placed

• **cifblk** – dict or PyCifRW block containing values for each CIF item

• **loopstructure**(list) – a list of lists containing the contents of each loop, as an example:

```
[ ["_a","_b"], ["_c"], ["_d_1","_d_2","_d_3"]]
```

this describes a CIF with this type of structure:

```
loop_ _a _b <a1> <b1> <a2> ...
loop_ _c <c1> <c2>...
loop _d_1 _d_2 _d_3 ...
```

Note that the values for each looped CIF item, such as _a, are contained in a list, for example as cifblk["_a"]

• **cifdic**(dict) – optional CIF dictionary definitions

• **OKbuttons**(list) – A list of wx.Button objects that should be disabled when information in the CIF is invalid

• **(other)** – optional keyword parameters for wx.ScrolledPanel

**CIFEntryWidget**(dct, item, dataname)

Create an entry widget for a CIF item. Use a validated entry for numb values where int is required when limits are integers and floats otherwise. At present this does not allow entry of the special CIF values of “.” and “?” for numerical values and highlights them as invalid. Use a selection widget when there are specific enumerated values for a string.

**ControlOKButton**(setvalue)

Enable or Disable the OK button(s) for the dialog. Note that this is passed into the ValidatedTxtCtrl for use by validators.

**Parameters** setvalue**(bool)** – if True, all entries in the dialog are checked for validity. The first invalid control triggers disabling of buttons. If False then the OK button(s) are disabled with no checking of the invalid flag for each control.

**DoLayout**()

Update the Layout and scroll bars for the Panel. Clears self.LayoutCalled so that next change to panel can request a new update

**OnAddRow**(event)

add a row to a loop

**OnLayoutNeeded**(event)

Called when an update of the panel layout is needed. Calls self.DoLayout after the current operations are complete using CallAfter. This is called only once, according to flag self.LayoutCalled, which is cleared in self.DoLayout.

**class** G2export_CIF>EditCIFtemplate(parent, cifblk, loopstructure, defaultname)

Create a dialog for editing a CIF template. The edited information is placed in cifblk. If the CIF is saved as a file, the name of that file is saved as self.newfile.

1.18. GSAS-II Export Modules
Parameters

- **parent** (*wx.Frame*) – parent frame or None
- **cifblk** – dict or PyCifRW block containing values for each CIF item
- **loopstructure** (*list*) – a list of lists containing the contents of each loop, as an example:

  ```python
  [["_a","_b"], ["_c"], ["_d_1","_d_2","_d_3"]]
  ```

  this describes a CIF with this type of structure:

  ```text
  loop_ _a _b <a1> <b1> <a2> ...
  loop_ _c <c1> <c2>...
  loop _d_1 _d_2 _d_3 ...
  ```

  Note that the values for each looped CIF item, such as _a, are contained in a list, for example as cifblk["_a"]

- **defaultname** (*str*) – specifies the default file name to be used for saving the CIF.

*Post* ()

Display the dialog

*Returns* True unless Cancel has been pressed.

class G2export_CIF.ExportCIF (G2frame, formatName, extension, longFormatName=None)

Base class for CIF exports

ValidateAscii (checklist)

Validate items as ASCII

class G2export_CIF.ExportHKLCIF (G2frame)

Used to create a simple CIF containing diffraction data only. Uses exact same code as ExportCIF except that histOnly is set for the Exporter Shows up in menu as Quick CIF.

Parameters

- **G2frame** (*wx.Frame*) – reference to main GSAS-II frame

class G2export_CIF.ExportPhaseCIF (G2frame)

Used to create a simple CIF with one phase. Uses exact same code as ExportCIF except that phaseOnly is set for the Exporter Shows up in menu as Quick CIF.

Parameters

- **G2frame** (*wx.Frame*) – reference to main GSAS-II frame

class G2export_CIF.ExportProjectCIF (G2frame)

Used to create a CIF of an entire project

Parameters

- **G2frame** (*wx.Frame*) – reference to main GSAS-II frame

class G2export_CIF.ExportPwdrCIF (G2frame)

Used to create a simple CIF containing diffraction data only. Uses exact same code as ExportCIF except that histOnly is set for the Exporter Shows up in menu as Quick CIF.

Parameters

- **G2frame** (*wx.Frame*) – reference to main GSAS-II frame

Writer (*hist*, mode='w')

Used for histogram CIF export of a sequential fit.

c232 Chapter 1. Contents

G2export_CIF.FmtAtomType (sym)

Reformat a GSAS-II atom type symbol to match CIF rules

G2export_CIF.HillSortElements (elmlist)

Sort elements in “Hill” order: C, H, others, (where others are alphabetical).
**Params list elmlist**  a list of element strings

**Returns**  a sorted list of element strings

**G2export_CIF.LoadCIFdic()**
Create a composite core+powder CIF lookup dict containing information about all items in the CIF dictionaries, loading pickled files if possible. The routine looks for files named cif_core.cpickle and cif_pd.cpickle in every directory in the path and if they are not found, files cif_core.dic and/or cif_PD.dic are read.

**Returns**  the dict with the definitions

**G2export_CIF.PickleCIFdict (fil)**
Loads a CIF dictionary, cherry picks out the items needed by local code and sticks them into a python dict and writes that dict out as a pickle file for later reuse. If the write fails a warning message is printed, but no exception occurs.

**Parameters**  **fil**  (*str*) – file name of CIF dictionary, will usually end in .dic

**Returns**  the dict with the definitions

**G2export_CIF.WriteAtomsMagnetic (fp, phasedict, phasenam, parmDict, sigDict, labellist)**
Write atom positions to CIF

**G2export_CIF.WriteAtomsNuclear (fp, phasedict, phasenam, parmDict, sigDict, labellist)**
Write atom positions to CIF

**G2export_CIF.WriteCIFitem (fp, name, value=")**
Helper function for writing CIF output. Translated from exports/G2export_CIF.py

**G2export_CIF.WriteComposition (fp, phasedict, phasenam, parmDict)**
determine the composition for the unit cell, crudely determine Z and then compute the composition in formula units

**G2export_CIF.dict2CIF (dblk, loopstructure, blockname='Template')**
Create a PyCifRW CIF object containing a single CIF block object from a dict and loop structure list.

**Parameters**
- **dblk**  – a dict containing values for each CIF item
- **loopstructure**  (*list*)  – a list of lists containing the contents of each loop, as an example:

  ```
  [ ["_a","_b"], ["_c"], ["_d_1","_d_2","_d_3"]]
  ```

  this describes a CIF with this type of structure:

  ```
  loop_ _a _b <a1> <b1> <a2> ...
  loop_ _c <c1> <c2>...
  loop_ _d_1 _d_2 _d_3 ...
  ```

  Note that the values for each looped CIF item, such as _a, are contained in a list, for example as cifblk["_a"]

- **blockname**  (*str*)  – an optional name for the CIF block. Defaults to ‘Template’

**Returns**  the newly created PyCifRW CIF object

## 1.18.8 Module G2export_pwdr: Export powder input files

Creates files used by GSAS (FXYE) & TOPAS (XYE) as input
class G2export_pwdr.ExportPowderFXYE(G2frame)
    Used to create a FXYE file for a powder data set

    Parameters
    G2frame (wx.Frame) – reference to main GSAS-II frame

    Exporter (event=None)
        Export one or more sets of powder data as FXYE file(s)

    WriteInstFile (hist, Inst)
        Write an instrument parameter file

    Writer (TreeName, filename=None, prmname="")
        Write a single PWDR entry to a FXYE file

class G2export_pwdr.ExportPowderXYE(G2frame)
    Used to create a Topas XYE file for a powder data set

    Parameters
    G2frame (wx.Frame) – reference to main GSAS-II frame

    Exporter (event=None)
        Export one or more sets of powder data as XYE file(s)

1.18.9 Module G2export_FIT2D: Fit2D “Chi” export

Code to create .chi (Fit2D like) files for GSAS-II powder data export

class G2export_FIT2D.ExportPowderCHI(G2frame)
    Used to create a CHI file for a powder data set

    Parameters
    G2frame (wx.Frame) – reference to main GSAS-II frame

    Exporter (event=None)
        Export a set of powder data as a Fit2D .qchi file

class G2export_FIT2D.ExportPowderQCHI(G2frame)
    Used to create a q-binned CHI file for a powder data set

    Parameters
    G2frame (wx.Frame) – reference to main GSAS-II frame

    Exporter (event=None)
        Export a set of powder data as a q-bin Fit2D .qchi file
Python Module Index

a
atmdata, 62

b

c
config_example, 33
d
defaultIparms, 63
e
ElementTable, 61
f
FormFactors, 61
g
G2export_CIF, 230
G2export_csv, 228
G2export_examples, 227
G2export_FIT2D, 234
G2export_image, 229
G2export_map, 229
G2export_PDB, 229
G2export_pwd, 233
G2export_shelx, 230
G2img_1ITF, 224
G2img_ADSC, 223
G2img_CBF, 225
G2img_CheMin, 225
G2img_EDF, 223
G2img_GE, 223
G2img_HDF5, 225
G2img_MAR, 224
G2img_Rigaku, 224
G2img_SFRM, 226
G2img_SumG2, 223
G2pdf_gr, 226
G2phase, 216
G2phase_CIF, 217
G2phase_GPX, 217
G2phase_INS, 217
G2pwd_BrukerRAW, 219
G2pwd_CIF, 218
G2pwd_csv, 219
G2pwd_FP, 219
G2pwd_fxye, 218
G2pwd_GPX, 218
G2pwd_Panalytical, 219
G2pwd_rigaku, 220
G2pwd_xye, 218
G2rfd_xye, 226
G2sad_xye, 222
G2sfact, 220
G2sfact_CIF, 222
gltext, 59
GSASII, 4
GSASIIconstrGUI, 101
GSASIIctrlGUI, 64
GSASIIdata, 54
GSASIIdataGUI, 89
GSASIIddataGUI, 100
GSASIElem, 36
GSASIElemGUI, 100
GSASIIexprGUI, 103
GSASIIfiles, 54
GSASIIfpGUI, 108
GSASIIimage, 131
GSASIIimgGUI, 101
GSASIIIndex, 150
GSASIIIO, 82
GSASIII lattice, 38
GSASIIlog, 31
GSASIImapvars, 121
GSASIImath, 134
GSASIImpsub, 58
GSASIIobj, 4
GSASIIpath, 28
GSASIIphsGUI, 98
GSASIIplot, 151
GSASIIpwd, 159
GSASIIpwdGUI, 103
GSASIIpy3, 88
GSASIIrestrGUI, 105
GSASIIsasd, 165
GSASIIspc, 47
GSASIIsIO, 117
GSASIstrMain, 110
GSASIstrMath, 111
GSASIItestplot, 211

ImageCalibrants, 61

makeBat, 212
makeLinux, 212
makeMacApp, 212
makeTutorial, 212

ReadMarCCDFrame, 88

scanCCD, 211
Substances, 169

testDeriv, 211

unit_tests, 213
A
A2cell() (in module GSASIIlattice), 39
A2Gmat() (in module GSASIIlattice), 38
A2invcell() (in module GSASIIlattice), 39
A2values() (in module GSASIIindex), 150
abeles() (in module GSASIIpwd), 162
Absorb() (in module GSASIIpwd), 159
AbsorbDerv() (in module GSASIIpwd), 159
add() (in module GSASIIscriptable), 208
add3D() (GSASIIplot.G2PlotNoteBook method), 154
add_back_peak() (GSASIIscriptable.G2PwdrData method), 201
add_constraint_raw() (GSASIIscriptable.G2Project method), 193
add_image() (GSASIIscriptable.G2Project method), 193
add_PDF() (GSASIIscriptable.G2Project method), 193
add_peak() (GSASIIscriptable.G2PwdrData method), 202
add_phase() (GSASIIscriptable.G2Project method), 193
add_powder_histogram() (GSASIIscriptable.G2Project method), 194
add_simulated_powder_histogram() (GSASIIscriptable.G2Project method), 194
AddHatomDialog (class in GSASIIphsGUI), 99
addMpl() (GSASIIplot.G2PlotNoteBook method), 154
addOgl() (GSASIIplot.G2PlotNoteBook method), 154
addPrevGPX() (in module GSASIIpath), 29
AddSimulatedPowder() (GSASIIdataGUI.GSASII method), 90
adjHKLmax() (in module GSASIImath), 141
adp_flag (GSASIIscriptable.G2AtomRecord attribute), 184
ADSC_ReaderClass (class in G2img_ADSC), 223
AllOps() (in module GSASIIscpe), 47
altSettingOrtho (in module GSASIIspc), 53
AngleDialog (class in GSASIIexprGUI), 105
anneal() (in module GSASIImath), 142
AppleScript (in module makeMacApp), 212
ApplyEdit() (GSASIIctrlGUI.GridFractionEditor method), 72
ApplyModulation() (in module GSASIImath), 134
ApplyRBModelDervs() (in module GSASIIstrMath), 111
ApplyRBModels() (in module GSASIIstrMath), 111
ApplySeqData() (in module GSASIImath), 134
ApplyStringOps() (in module GSASIIspc), 48
ApplyStringOpsMom() (in module GSASIIspc), 48
ApplyXYZshifts() (in module GSASIIstrMath), 111
Arc_mask_azimuth (in module config_example), 34
arrayList (in module GSASIImapvars), 131
ASCIIValidator (class in GSASIIctrlGUI), 65
askSaveDirectory() (GSASIIIO.ExportBaseclass method), 84
askSaveDirectory() (in module GSASIIctrlGUI), 82
askSaveFile() (GSASIIIO.ExportBaseclass method), 84
askSaveFile() (in module GSASIIctrlGUI), 82
assignVars (GSASIIobj.ExpressionObj attribute), 20
atmdata (module), 62
atom() (GSASIIscriptable.G2Phase method), 189
AtomIdLookup (in module GSASIIobj), 17
AtomRandIdLookup (in module GSASIIobj), 17
Atoms record description, 10
atoms() (GSASIIscriptable.G2Phase method), 189
AtomTLS2UIJ() (in module GSASIImath), 134
AutoInt_PollTime (in module config_example), 34
AutoIntFrame (class in GSASIIimgGUI), 101
AV2Q() (in module GSASIImath), 134
AVdeg2Q() (in module GSASIImath), 134

B
Background (GSASIIscriptable.G2PwdrData attribute), 201
Banks (GSASIIobj.ImportStructFactor attribute), 24
BBPointDetector (in module GSASIIfpapGUI), 108
BBPSDDetector (in module GSASIIfpapGUI), 108
BessIn() (in module GSASIImath), 134
BessJn() (in module GSASIImath), 134
BestPlane() (in module GSASI1strMain), 110
betaij2Uij() (in module GSASI1lattice), 44
bind() (gltext.TextElement method), 60
blkSize (in module GSASI1scriptable), 209
BlockSelector() (in module GSASI1HO), 83
BondDialog (class in GSASI1expGUI), 105
BraggBrentanoParms (in module GSASI1pfaGUI), 108
ButtonBindingLookup (in module GSASI1log), 32
ButtonLogEntry (class in GSASI1log), 32

calc_M20() (in module GSASI1index), 150
calc_M20SS() (in module GSASI1index), 150
calc_rDsq2() (in module GSASI1lattice), 44
calc_rDsq2SS() (in module GSASI1lattice), 44
calc_rDsqT() (in module GSASI1lattice), 44
calc_rDsqTSS() (in module GSASI1lattice), 44
calc_rDsqZ() (in module GSASI1lattice), 44
calc_rV() (in module GSASI1lattice), 44
calc_rVsq() (in module GSASI1lattice), 44
calcFij() (in module GSASI1lattice), 44
calcIncident() (in module GSASI1lattice), 44
calcMaskMap() (in module GSASI1scriptable), 209
CalcPDF() (in module GSASI1pwd), 159
calcRamaEnergy() (in module GSASI1math), 143
calcThetaAzimMap() (in module GSASI1scriptable), 209
calcTorsionAzim() (in module GSASI1scriptable), 143
calculate() (GSASI1scriptable.G2PDF method), 187
CallScrolledMultiEditor() (in module GSASI1ctrlGUI), 65
cauchy_gen (class in GSASI1pwd), 162
CBF_ReaderClass (class in G2img_CBF), 225
cell2A() (in module GSASI1lattice), 45
cell2AB() (in module GSASI1lattice), 45
cell2Gmat() (in module GSASI1lattice), 45
cell2GS() (in module GSASI1lattice), 45
CellAbsorption() (in module GSASI1lattice), 39
CellBlock() (in module GSASI1lattice), 39
CellDijFill() (in module GSASI1lattice), 45
cellFill() (in module GSASI1strIO), 121
cellVary() (in module GSASI1strIO), 121
CentCheck() (in module GSASI1lattice), 39
centered (gltext.Text attribute), 59
centered (gltext.TextElement attribute), 60
changePlotSettings() (in module GSASI1plot), 159
ChargeFlip() (in module GSASI1math), 134
CheckAllScalePhaseFractions() (in module GSASI1constrGUI), 101
CheckConstraints() (in module GSASI1mapvars), 127
CheckElement() (in module GSASI1Elem), 36
checkEllipse() (in module GSASI1image), 133
CheckEquivalences() (in module GSASI1mapvars), 128
checkHKLextc() (in module GSASI1spc), 53
CheckInput() (GSASI1ctrlGUI.NumberValidator method), 75
checkMagextc() (in module GSASI1spc), 53
CheckNotebook() (GSASI1dataGUI.GSASI1 method), 90
checkPDFprm() (GSASI1imgGUI.AutoIntFrame method), 102
CheckScalePhaseFractions() (in module GSASI1constrGUI), 101
CheckSpin() (in module GSASI1spc), 48
CheckVars() (GSASI1exprGUI.ExpressionDialog method), 106
CheckVars() (GSASI1obj.ExpressionObj method), 19
ChooseTutorial() (GSASI1ctrlGUI.OpenTutorial method), 75
ChooseTutorial2() (GSASI1ctrlGUI.OpenTutorial method), 75
CIF2dict() (in module G2export_CIF), 230
CIFdefHelp (class in G2export_CIF), 230
CIFEntryWidget() (G2export_CIF.EditCIFpanel method), 231
CIFhklReader (class in G2sfact_CIF), 222
CIFFrameReader (class in G2phase_CIF), 217
CIFpwdReader (class in G2pwd_CIF), 219
CIFtemplateSelect (class in G2export_CIF), 230
CIFValidator() (GSASI1obj.ImportBaseclass method), 22
CleanupMasks() (in module GSASI1imgGUI), 102
clear() (GSASI1plot.G2PlotNoteBook method), 154
clear_HAP_refinements() (GSASI1scriptable.G2Phase method), 189
clear_refinements() (GSASI1scriptable.G2Phase method), 189
clear_refinements() (GSASI1scriptable.G2PwdrData method), 202
ClearData() (GSASI1dataGUI.G2DataWindow method), 90
ClearStartup() (GSASI1.G2App method), 4
Clip_on (in module config_example), 34
Clone() (GSASI1ctrlGUI.LASCIFValidator method), 65
Clone() (GSASI1ctrlGUI.NumberValidator method), 75
close_powder_histogram() (GSASI1scriptable.G2Project method), 195
CloseFile() (GSASIIIO.ExportBaseclass method), 83
Column_Metadata_directory (in module config_example), 34
ColumnValidator() (in module GSASIIfact), 220
combinations() (in module GSASIIlattice), 45
compiledExpr (GSASIIOb.ExpressionCalcObj attribute), 18
CompileVarDesc() (in module GSASIIobj), 17
composition (GSASIIScriptable.G2Phase attribute), 190
ComptonFac() (in module GSASIElem), 36
ComputeArc() (in module GSASIIplot), 153
ComputeDepESD() (in module GSASIImapvars), 128
computePDF() (in module GSASIIpwdGUI), 105
ComputePwdrProfCW() (in module GSASIImpsubs), 58
ComputePwdrProfTOF() (in module GSASIImpsubs), 58
cfg_example (module), 33
cnsNum (in module GSASIImapvars), 131
Constraint definition object description, 5
ConstraintDialog (class in GSASIIconstrGUI), 101
ConstraintException, 128
Constraints object description, 4
ContentsValidator()
    (G2img_ITIF.TIF_ReaderClass method), 225
ContentsValidator()
    (G2img_ADSC.ADSC_ReaderClass method), 223
ContentsValidator()
    (G2img_CBF.CBF_ReaderClass method), 225
ContentsValidator()
    (G2img_CheMin.png_ReaderClass method), 225
ContentsValidator()
    (G2img_EDF.EDF_ReaderClass method), 223
ContentsValidator()
    (G2img_GE.GE_ReaderClass method), 223
ContentsValidator()
    (G2img_GE.GEsum_ReaderClass method), 224
ContentsValidator()
    (G2img_HDF5.HDF5_Reader method), 225
ContentsValidator()
    (G2img_MAR.MAR_ReaderClass method), 224
ContentsValidator()
    (G2img_Rigaku.Rigaku_ReaderClass method), 220
ContentsValidator()
    (G2pdf_gr.txt_PDFReaderClass method), 226
ContentsValidator()
    (G2pdf_gr.txt_PDFReaderClassG method), 226
ContentsValidator()
    (G2phase.EXP_ReaderClass method), 216
ContentsValidator()
    (G2phase_JANA_ReaderClass method), 216
ContentsValidator()
    (G2phase.PDB_ReaderClass method), 216
ContentsValidator()
    (G2phase.PDF_ReaderClass method), 216
ContentsValidator()
    (G2phase_CIF.CIFPhaseReader method), 217
ContentsValidator()
    (G2phase_GPX.PhaseReaderClass method), 217
ContentsValidator()
    (G2phase_INS.PhaseReaderClass method), 217
ContentsValidator()
    (G2pwd_BrukerRAW.raw_ReaderClass method), 219
ContentsValidator()
    (G2pwd_CIF.CIFpwdReader method), 219
ContentsValidator()
    (G2pwd_csv.csv_ReaderClass method), 219
ContentsValidator()
    (G2pwd_FP.fp_ReaderClass method), 219
ContentsValidator()
    (G2pwd_GPX.GSAS2_ReaderClass method), 218
ContentsValidator()
    (G2pwd_fxye.GSAS_ReaderClass method), 218
ContentsValidator()
    (G2pwd_GPX.GSAS2_ReaderClass method), 218
ContentsValidator()
    (G2pwd_Panalytical.Panalytical_ReaderClass method), 219
ContentsValidator()
    (G2pwd_rigaku.Rigaku_rasReaderClass method), 220
ContentsValidator()
    (G2pwd_rigaku.Rigaku_txtReaderClass method), 220
ContentsValidator()
  (G2pwd_xye.xye_ReaderClass method), 218
ContentsValidator()
  (G2rfd_xye.txt_NeutronReaderClass method), 226
ContentsValidator()
  (G2rfd_xye.txt_XRayReaderClass method), 227
ContentsValidator()
  (G2rfd_xye.txt_XRayThetaReaderClass method), 227
ContentsValidator()
  (G2sad_xye.txt_CWNeutronReaderClass method), 222
ContentsValidator()
  (G2sad_xye.txt_nmCWNeutronReaderClass method), 222
ContentsValidator()
  (G2sad_xye.txt_XRayReaderClass method), 222
ContentsValidator()
  (G2sfact.HKLF_ReaderClass method), 220
ContentsValidator()
  (G2sfact.HKLMF_ReaderClass method), 221
ContentsValidator()
  (G2sfact.M90_ReaderClass method), 221
ContentsValidator()
  (G2sfact.NIST_hb3a_INT_ReaderClass method), 221
ContentsValidator()
  (G2sfact.NT_HKLF2_ReaderClass method), 221
ContentsValidator()
  (G2sfact.NT_JANA2K_ReaderClass method), 221
ContentsValidator()
  (G2sfact.SHELX4_ReaderClass method), 221
ContentsValidator()
  (G2sfact.SHELX5_ReaderClass method), 221
ContentsValidator()
  (G2sfact.SHELX6_ReaderClass method), 222
ContentsValidator()
  (G2sfact.CIF.CIFhklReader method), 222
ContentsValidator()
  (GSASIIobj.ImportBaseclass method), 22
Contour_color (in module config_example), 34

D
Data object descriptions
  Atoms record, 10
  Constraint Definition, 5
  Constraints, 4
  Covariance, 5
  Drawing atoms record, 10
  Phase, 6
  Powder Data, 11
  Powder Reflections, 13
  Rigid Body Data, 8
  Single crystal data, 13
  Single Crystal Reflections, 14
  Space Group Data, 9
  Superspace Group Data, 10
  debug (in module config_example), 35
DefaultAutoScale (in module config_example), 34
DefaultControls (in module GSASIIobj), 18
defaultIparms (in module defaultIparms), 64
defaultIparms (module), 63
Define_wxId() (in module GSASIIctrlGUI), 65
DefineEvaluator() (in module GSASIIctrlGUI), 102
del_back_peak() (GSASIIscriptable.G2PwdrData method), 202
Delete() (GSASIIplot.G2PlotNoteBook method), 153
DeleteElement (class in GSASIIElemGUI), 100
deleteTexture() (gltext.TextElement method), 60
Den2Vol() (in module GSASIImath), 134
density (GSASIIscriptable.G2Phase attribute), 190
dependentParmList (in module GSASIImapvars), 131
dependentVar (GSASIIexprGUI.ExpressionDialog attribute), 107
dependentVars (in module GSASIImapvars), 131
depVarDict (GSASIIexprGUI.ExpressionDialog attribute), 107
dervHKL() (in module GSASIIstrMath), 116
dervRefine() (in module GSASIIstrMath), 116
dict2CIF() (in module G2export_CIF), 233
Dict2Deriv() (in module GSASIImapvars), 128
Dict2Map() (in module GSASIImapvars), 128
Dict2Values() (in module GSASIImapvars), 159
Dict2Values() (in module GSASIImapvars), 111
dictLogged (class in GSASIIlog), 33
DIFFaXcontrols (class in GSASIIphsGUI), 99
DiluteSF() (in module GSASIIstrsas), 165
DisAglDialog (class in GSASIIctrlGUI), 66
DisAglTor() (in module GSASIIdstrMain), 110
do_refinements() (GSASIIscriptable.G2Project method), 195
doFPACalc() (in module GSASIIphsGUI), 109
DoIndexPeaks() (in module GSASIIindex), 150
DoLayout() (G2export_CIF.EditCIFpanel method), 150
DoNothing() (in module GSASIIpath), 28
DoPeakFit() (in module GSASIIpwd), 159
downdate (class in GSASIIctrlGUI), 82
DownloadAll() (GSASIIctrlGUI.OpenTutorial method), 75
DownloadG2Binaries() (in module GSASIIpath), 28
Draw() (GSASIIctrlGUI.DisAglDialog method), 66
Draw() (GSASIIphsGUI.AddHatomDialog method), 99
draw_text() (gltext.Text method), 59
draw_text() (gltext.TextElement method), 60
Drawing atoms record description, 10
DrawingPanel() (GSASIIctrlGUI.ShowLSParms method), 79
Dsp2pos() (in module GSASIIlattice), 39
dump() (in module GSASIIscriptable), 209
dumpTree() (GSASIIIO.ExportBaseclass method), 85

e
EDF_ReaderClass (class in G2img_EDF), 223
EdgeFinder() (in module GSASIIimage), 131
EditCIFpanel (class in G2export_CIF), 231
EditCIFtemplate (class in G2export_CIF), 231
EditExpression() (GSASIIobj.ExpressionObj method), 19
EditProxyInfo() (GSASIIdataGUI.GASASII method), 90
El2EstVol() (in module GSASIImath), 135
El2Mass() (in module GSASIImath), 135
ElButton() (GSASIIElemGUI.DeleteElement method), 101
ElButton() (GSASIIElemGUI.PickElement method), 101
element (GSASIIscriptable.G2AtomRecord attribute), 184
ElementTable (module), 61
ElemPosition() (in module GSASIIspc), 48
ellipseSize() (in module GSASIIpwd), 162
ellipseSizeDerv() (in module GSASIIpwd), 162
Enable_logging (in module config_example), 34
EnableButtons() (GSASIIimgGUI.AutoIntFrame method), 102
EnumSelector (class in GSASIIctrlGUI), 66
eObj (GSASIIobj.ExpressionCalcObj attribute), 18
ErrorDialog() (GSASIIdataGUI.GSASII method), 90
eErrRefine() (in module GSASIIstrMath), 116
evalColMetadataDicts() (in module GSASIIfiles), 56
EvalExpression() (GSASIIobj.ExpressionCalcObj method), 18
EvaluateMultipliers() (in module GSASIImapvars), 128
exceptHook() (in module GSASIIpath), 29
ExitMain() (GSASIIdataGUI.GSASII method), 91
EXP_ReaderClass (class in G2phase), 216
ExpandAll() (GSASIIdataGUI.GSASII method), 91
export() (GSASIIscriptable.G2PDF method), 188
export() (GSASIIscriptable.G2PwdrData method), 201
export() (in module GSASIIscriptable), 210
export_CIF() (GSASIIscriptable.G2Phase method), 190
Export_peaks() (GSASIIscriptable.G2PwdrData method), 201
ExportBaseclass (class in GSASIIIO), 83
ExportCIF (class in G2export_CIF), 232
Exporter() (G2export_csv.ExportMultiPowderCSV method), 228
Exporter () (G2export_csv.ExportPhaseCSV method), 228
Exporter () (G2export_csv.ExportPowderCSV method), 228
Exporter () (G2export_csv.ExportPowderRefCSV method), 228
Exporter () (G2export_csv.ExportSASDCSV method), 228
Exporter () (G2export_csv.ExportSingleCSV method), 228
Exporter () (G2export_csv.ExportStrainCSV method), 228
Exporter () (G2export_examples.ExportPhaseText method), 227
Exporter () (G2export_examples.ExportPowderText method), 227
Exporter () (G2export_examples.ExportPowderRefText method), 227
Exporter () (G2export_examples.ExportSingleText method), 227
Exporter () (G2export_fit2d.ExportPowderCHI method), 234
Exporter () (G2export_fit2d.ExportPowderQCHI method), 234
Exporter () (G2export_image.ExportImagePNG method), 229
Exporter () (G2export_map.ExportMapASCII method), 229
Exporter () (G2export_map.ExportMapCCP4 method), 229
Exporter () (G2export_pdb.ExportPhaseCartXYZ method), 229
Exporter () (G2export_pdb.ExportPhasePDB method), 229
Exporter () (G2export_pwdr.ExportPowderXYE method), 234
Exporter () (G2export_pwdr.ExportPowderXYE method), 234
Exporter () (G2export_shelx.ExportPhaseShelx method), 230
exportersByExtension (in module GSASIIExportable), 210
ExportHKLCIF (class in G2export_CIF), 232
ExportImagePNG (class in G2export_image), 228
ExportMapASCII (class in G2export_map), 229
ExportMapCCP4 (class in G2export_map), 229
ExportMultiPowderCSV (class in G2export_csv), 228
ExportPhaseCartXYZ (class in G2export_PDB), 229
ExportPhaseCIF (class in G2export_CIF), 232
ExportPhaseCSV (class in G2export_csv), 228
ExportPhasePDB (class in G2export_PDB), 229
ExportPhaseShelx (class in G2export_shelx), 230
ExportPhaseText (class in G2export_examples), 227
ExportPowder () (in module GSASIIIO), 85
ExportPowderCHI (class in G2export_fit2d), 234
ExportPowderCSV (class in G2export_csv), 228
ExportPowderFXYE (class in G2export_pwdr), 233
ExportPowderList () (in module GSASIIIO), 85
ExportPowderQCHI (class in G2export_fit2d), 234
ExportPowderRefCSV (class in G2export_csv), 228
ExportPowderRefText (class in G2export_examples), 227
ExportPowderText (class in G2export_examples), 227
ExportPowderXYE (class in G2export_pwdr), 234
ExportProjectCIF (class in G2export_CIF), 232
ExportPwdrCIF (class in G2export_CIF), 232
ExportSASDCSV (class in G2export_csv), 228
ExportSequential () (GSASIIIO.ExportBaseclass method), 83
ExportSequential () (in module GSASIIIO), 85
ExportSingleCSV (class in G2export_csv), 228
ExportSingleText (class in G2export_csv), 227
ExportStrainCSV (class in G2export_csv), 228
expr (GSASIIexprGUI.ExpressionDialog attribute), 107
exprDict (GSASIIobj.ExpressionCalcObj attribute), 18
expression (GSASIIobj.ExpressionObj attribute), 20
ExpressionCalcObj (class in GSASIIobj), 18
ExpressionDialog (class in GSASIIexprGUI), 105
ExpressionObj (class in GSASIIobj), 19
exprVarList (GSASIIexprGUI.ExpressionDialog attribute), 107
ExtensionValidator () (GSASIIObj.ImportBaseclass method), 22
ExtractFileFromZip () (in module GSASIIIO), 85
factorize () (in module GSASIIpwrd), 162
fcjde_gen (class in GSASIIpwrd), 162
FileDlgFixExt () (in module GSASIIIO), 86
Fill2ThetaAzimuthMap () (in module GSASIIimage), 131
FillAtomLookUp () (in module GSASIImath), 135
fillgmat () (in module GSASIIlattice), 45
FillList () (GSASIIImgGUI.ImgIntLstCtrl method), 102
FillMainMenu () (GSASIIDataGUI.GSASII method), 91
Filter () (GSASIIctrlGUI.G2MultiChoiceDialog method), 69
find () (in module GSASIIfiles), 56
FindAtomIndexByIds () (in module GSASIImath), 135
findBestCell () (in module GSASIIindex), 150
FindBondsDraw () (in module GSASIIphsGUI), 99
FindBondsDrawCell (in module GSASIIphasGUI), 99
findControl (in module GSASIIscriptable.G2Image method), 185
FindNonstandard (in module GSASIIlattice), 40
findOffset () (in module GSASIImath), 143
FindPlotTab () (GSASIIPlot.G2PlotNoteBook method), 153
findSSOffset () (in module GSASIImath), 143
fit_fixed_points () (GSASIIscriptable.G2PwdrData method), 202
FitDetector () (in module GSASIImage), 131
FitHKL () (in module GSASIIndex), 150
FitHKLT () (in module GSASIIndex), 150
FitHKLTT () (in module GSASIIndex), 150
FitHKLZ () (in module GSASIIndex), 150
FitMultiDist () (in module GSASIImage), 132
FitStrain () (in module GSASIImage), 132
FitStrata () (in module GSASIImage), 132
fixedDict () (in module GSASIImapvars), 131
fixedVarList () (in module GSASIImapvars), 131
fixMono () (in module GSASIIspec), 53
FixValence () (in module GSASHElem), 36
FlagSetDialog (class in GSASIIctrlGUI), 66
Flnh () (in module GSASIIlattice), 40
FmtAtomType () (in module G2export_CIF), 232
fmtVarDescr () (in module GSASIImodule), 27
font (gltext.TextElement attribute), 59
font_size (gltext.Text attribute), 59
foreground (gltext.Text attribute), 59
foreground (gltext.TextElement attribute), 59
FormatPadValue () (in module GSASIIpy3), 88
FormatSigFigs () (in module GSASIIpy3), 88
FormatValue () (in module GSASIIpy3), 89
FormFactors (module), 61
FormulaEval () (in module GSASIIpy3), 89
Fourier4DMap () (in module GSASIImath), 135
FourierMap () (in module GSASIImath), 135
fp_ReaderClass (class in G2pwd_FP), 219
FPcalc () (in module GSASIIElem), 36
freeVars (GSASIIobj.ExpressionObj attribute), 20
from_dict_and_names () (GSASIIscriptable.G2Project class method), 196
fullIntegrate (in module config_example), 36
fxnPKgdict (GSASIIobj.ExpressionCalcObj attribute), 18

G
G2_ReaderClass (class in G2img_SumG2), 223
G2App (class in GSASII), 4
G2AtomRecord (class in GSASIIscriptable), 183
G2CheckBox (class in GSASIIctrlGUI), 66
G2ChoiceButton (class in GSASIIctrlGUI), 67
G2ColumnDDialog (class in GSASIIctrlGUI), 67
G2DataWindow (class in GSASIIdataGUI), 89
G2Exception, 20
G2export_CIF (module), 230
G2export_csv (module), 228
G2export_examples (module), 227
G2export_FIT2D (module), 234
G2export_image (module), 229
G2export_map (module), 229
G2export_PDB (module), 229
G2export_pwdr (module), 233
G2export_shelx (module), 230
G2HistoDataDialog (class in GSASIIctrlGUI), 68
g2home (in module GSASIIpath), 29
G2HtmlWindow (class in GSASIIctrlGUI), 68
G2Image (class in GSASIIscriptable), 184
G2img_1TIF (module), 224
G2img_ADSC (module), 223
G2img_CBF (module), 225
G2img_CheMin (module), 225
G2img_EDF (module), 223
G2img_GE (module), 223
G2img_HDF5 (module), 225
G2img_MAR (module), 224
G2img_Rigaku (module), 224
G2img_SFRM (module), 226
G2img_SumG2 (module), 223
G2ImportException, 187
G2LoggedButton (class in GSASIIctrlGUI), 68
G2logList (in module GSASIILog), 32
G2MessageBox () (in module GSASIIctrlGUI), 68
G2MultiChoiceDialog (class in GSASIIctrlGUI), 69
G2ObjectWrapper (class in GSASIIscriptable), 187
G2PDF (class in GSASIIscriptable), 187
G2pdf_gr (module), 216
G2phase (class in GSASIIscriptable), 189
G2phase (module), 216
G2phase_CIF (module), 217
G2phase_GPX (module), 217
G2phase_INS (module), 217
G2Plot3D (class in GSASIIplot), 153
G2PlotMpl (class in GSASIIplot), 153
G2PlotNoteBook (class in GSASIIplot), 153
G2PlotOgl (class in GSASIIplot), 154
G2Print () (in module GSASIIfiles), 54
G2printlnLevel (in module GSASIIfiles), 55
G2Project (class in GSASIIscriptable), 192
G2pwd_BrukerRAW (module), 219
G2pwd_CIF (module), 218
G2pwd_csv (module), 219
G2pwd_FP (module), 219
G2pwd_fxye (module), 218
getCWsigDeriv() (in module GSASIImath), 144
GetData () (GSASIIctrlGUI.DisAgDispDialog method), 66
GetData () (GSASIIctrlGUI.G2HistoDataDialog method), 68
GetData () (GSASIIpksGUI.AddHatomDialog method), 99
GetData () (GSASIIpksGUI.SetUpFullmcDialog method), 99
GetData () (GSASIIpksGUI.SetUpRMCPProfileDialog method), 99
getdata () (GSASIIscriptable.G2PwrdData method), 202
GetDATSig () (in module GSASIImath), 136
getDensity () (in module GSASIImath), 144
GetDependentVars () (in module GSASIImapvars), 129
getDepsVoigt () (in module GSASIIpwd), 164
GetDepVar () (GSASIIexprGUI.ExpressionDialog method), 106
GetDepVar () (GSASIIobj.ExpressionObj method), 19
getDescr () (in module GSASIIobj), 27
GetDetectorXY () (in module GSASIIimage), 132
GetDetXYfromThAzm () (in module GSASIIimage), 132
getFCJVoigt3 () (in module GSASIIpwd), 164
GetDisplay () (in module GSASIIdataGUI), 97
getDistDerv () (in module GSASIImath), 144
GetDistSig () (in module GSASIImath), 136
detMmax () (in module GSASIIindex), 150
detMmin () (in module GSASIIindex), 150
detDspVoigt () (in module GSASIIpwd), 164
detDsp () (in module GSASIIimage), 132
detEdfData () (in module G2img_EDF), 223
detEllipse () (in module GSASIIimage), 132
detEllipse2 () (in module GSASIIimage), 132
detEpsVoigt () (in module GSASIIpwd), 163
detExportPath () (in module GSASIIctrlGUI), 71
detFCJVoigt () (in module GSASIIpwd), 163
detFCJVoigt3 () (in module GSASIIpwd), 163
detFFc5 () (in module GSASIIelem), 36
detFFtable () (in module GSASIIelem), 36
detFFvalues () (in module GSASIIelem), 38
getFileList () (GSASIIdataGUI.GSASII method), 91
detFobsSq () (in module GSASIIstrMath), 111
detFormFactorSq () (in module GSASIIelem), 37
detFPrime () (in module GSASIIstrIO), 117
detFullGPX () (in module GSASIIstrIO), 117
detFWHM () (in module GSASIIpwd), 163
detgamFW () (in module GSASIIpwd), 164
detGenSym () (in module GSASIIspc), 49
detGEsumData () (in module G2img_GE), 224
getGPXtreeDataNames () (in module GSASIIdataGUI), 97
getGPXtreeItemId () (in module GSASIIdataGUI), 97
getHAPvalues () (GSASIIscriptable.G2Phase method), 190
GetHistogramData () (in module GSASIIstrIO), 117
GetHistogramNames () (GSASIIdataGUI.GSASII method), 91
GetHistogramNames () (in module GSASIIstrIO), 117
GetHistogramNamesID () (GSASIIdataGUI.GSASII method), 91
GetHistogramPhaseData () (in module GSASIIstrIO), 118
GetHistograms () (in module GSASIIstrIO), 118
GetHistLikeSelected () (in module GSASIIPwrdGUI), 103
GetHKLDatafromTree () (GSASIIdataGUI.GSASII method), 91
getHKLmax () (in module GSASIIlattice), 45
getHKLpeak () (in module GSASIIpwd), 163
getHKLpeak () (in module GSASIIpwd), 163
getHStrainShift () (in module GSASIImath), 111
getHStrainShiftDerv () (in module GSASIImath), 112
GetImageData () (in module GSASIImath), 86
GetImageLoc () (GSASIIctrlGUI.G2TreeCtrl method), 70
GetImageZ () (in module GSASIIimgGUI), 102
GetImgData () (in module G2img_ADSC), 223
GetImportFile () (in module GSASIIctrlGUI), 71
GetImportPath () (in module GSASIIctrlGUI), 71
GetIndependentVars () (GSASIIobj.ExpressionObj method), 19
GetIndependentVars () (in module GSASIImapvars), 129
getIntensityCorr () (in module GSASIImath), 112
GetIntensityDerv () (in module GSASIImath), 112
GetItemOrder () (in module GSASIIctrlGUI), 72
GetKc1 () (in module GSASIIlattice), 41
GetKc1Ksl () (in module GSASIIlattice), 41
GetKNsym () (in module GSASIIspec), 49
GetKsl () (in module GSASIIlattice), 41
GetLittleGrpOps () (in module GSASIIspec), 49
GetMagFormFacCoeff () (in module GSASIIelem), 37
GetMAR345Data () (in module G2img_MAR), 224
getMass () (GSASIIscriptable.G2Image method), 185
getMass () (in module GSASIImath), 144
getMeanWave () (in module GSASIImath), 145
GetMFtable() in module GSASHIElem, 37
getMFvalues() in module GSASHIElem, 38
GetNewCellParms() in module GSASHIstrMath, 112
GetNumDensity() in module GSASHIpwd, 160
GetNXUPQsym() in module GSASHIspc, 49
GetOprName() in module GSASHIspc, 49
GetOprPtrName() in module GSASHIspc, 49
GetOprPtrNumber() in module GSASHIspc, 49
GetPawleyConstr() in module GSASHIstrIO, 118
getPeakPos() in module GSASHIattice, 45
getPeakProfile() in module GSASHIpwd, 164
getPeakProfileDervMP() in module GSASHIstrMath, 116
getPowderProfileDervMP() in module GSASHIpwd, 164
getPowderProfileDerv() in module GSASHIpwd, 164
GetPhaseData() in module GSASHIstrMath, 91
GetPhaseData() in module GSASHIstrIO, 118
GetPhaseInfofromTree() in module GSASHIstrMath, 21
GetPhaseNames() in module GSASHIstrMath, 92
GetPhaseNames() in module GSASHIstrMath, 92
GetPowderIparm() in module GSASHIstrMath, 118
GetPowderPeaks() in module GSASHIIstrMath, 86
getPeakProfile() in module GSASHIstrMath, 116
getAddressElement() in module GSASHIstrMath, 116
GetPrefOri() in module GSASHIstrMath, 112
getPsVoigt() in module GSASHIstrMath, 112
GetPwdrExt() in module GSASHIstrMath, 112
calculatedHistNum() in module GSASHIstrMath, 112
GetPWDRdatafromTree() in module GSASHIstrMath, 112
getRGammaDeriv() in module GSASHIstrMath, 145
getRBTransMat() in module GSASHIstrMath, 145
GetRelativeHistNum() in module GSASHIstrMath, 112
calculateRelativePhaseNum() in module GSASHIstrMath, 112
getRestAngle() in module GSASHIstrMath, 145
getRestChiral() in module GSASHIstrMath, 145
getRestDeriv() in module GSASHIstrMath, 145
getRestDist() in module GSASHIstrMath, 145
getRestPlane() in module GSASHIstrMath, 146
getRestPolefig() in module GSASHIstrMath, 146
getAddressElement() in module GSASHIstrMath, 116
getRestRama() in module GSASHIstrMath, 146
getRestTorsion() in module GSASHIstrMath, 146
getRho() in module GSASHIstrMath, 146
getRhos() in module GSASHIstrMath, 146
GetRigaku() in module GS2img_Rigaku, 224
GetRigidBody() in module GSASHIstrIO, 118
GetRigidBodyModels() in module GSASHIstrIO, 118
GetSampleSigGam() in module GSASHIstrMath, 112
GetSampleSigGamDeriv() in module GSASHIstrMath, 112
GetSelectedRows() in module GSASHIstrrestrGUI, 105
GetSeqCell() in module GSASHIIIO.ExportBaseclass, 84
GetSeqResult() in module GSASHIstrIO, 118
GetSFRMData() in module GS2img_SFRM, 226
GetSGSpin() in module GSASHIspc, 49
GetSHCoeff() in module GSASHIstrMath, 136
getsvnProxy() in module GSASHIstrMath, 29
GetSymEquiv() in module GSASHIstrMath, 129
getSyXYZ() in module GSASHIstrMath, 146
GetTabIndex() in module GSASHIIplot.G2PlotNoteBook, 153
getTextElement() in gltext.Text, 59
getAddress() in gltext.Text, 59
getAddress() in gltext.Text, 59
GetTifData() in module GS2img_IJTIF, 224
calculateTOFalpha() in module GSASHIstrMath, 146
calculateTOFalphaDeriv() in module GSASHIstrMath, 147
calculateTOFbeta() in module GSASHIstrMath, 147
calculateTOFbetaDeriv() in module GSASHIstrMath, 147
calculateTOFgamma() in module GSASHIstrMath, 147
calculateTOFgammaDeriv() in module GSASHIstrMath, 147
calculateTOFsig() in module GSASHIstrMath, 147
calculateTOFsigDeriv() in module GSASHIstrMath, 147
calculateTorsionDeriv() in module GSASHIstrMath, 147
calculateTorsionSig() in module GSASHIstrMath, 136
calculateTth() in module GSASHIIimage, 132
calculateTthAzm() in module GSASHIIimage, 132
calculateTthAzmDisp() in module GSASHIIimage, 132
calculateTthAzmG() in module GSASHIIimage, 133
GetHistogramsAndPhases() in module GSASHIstrIO, 119
GetHistogramsAndPhasesfromTree() in module GSASHIstrMath, 92
GetValue() (GSASIIctrlGUI.StrMain), 80
GetValue() (GSASIIctrlGUI.MultiStringDialog method), 74
getVarDescr() (in module GSASIIobj), 27
GetVaried() (GSASIIobj.ExpressionObj method), 19
GetVariedVarVal() (GSASIIobj.ExpressionObj method), 19
getVarStep() (in module GSASIIobj), 27
getVary() (GSASIIscriptable.G2Image method), 185
getVCov() (in module GSASIImath), 148
getVersion() (GSASIIctrlGUI.downdate method), 82
GetVersionNumber() (in module GSASIIpath), 28
getWave() (in module GSASIImath), 148
getWidthsCW() (in module GSASIIpwd), 164
getWidthsTOP() (in module GSASIIpwd), 164
GetXZDcCoef() (in module GSASIIElem), 37
GetXZD1st() (in module GSASIImath), 137
Glhn() (in module GSASIIlattice), 41
gltex (module), 59
Gmat2A() (in module GSASIIlattice), 42
Gmat2AB() (in module GSASIIlattice), 42
GPXBackup() (in module GSASIIstrIO), 117
gpxSize (in module GSASIIstrIO), 121
GramSchmidtOrtho() (in module GSASIImapvars), 129
GridFractionEditor (class in GSASIIctrlGUI), 72
GroupConstraints() (in module GSASIImapvars), 129
GSAS2_ReaderClass (class in G2pwd_GPX), 218
GSAS_ReaderClass (class in G2pwd_fxye), 218
GSASII (class in GSASIIdataGUI), 90
GSASII (module), 4
GSASII.CopyDialog (class in GSASIIdataGUI), 90
GSASII.SumDialog (class in GSASIIdataGUI), 96
GSASIIindexGUI (module), 101
GSASIIctrlGUI (module), 64
GSASIIdata (module), 54
GSASIIdataGUI (module), 89
GSASIIdataGUI (module), 100
GSASIIElem (module), 36
GSASIIElemGUI (module), 100
GSASIIElemGUI (module), 105
GSASIIFiles (module), 54
GSASIIFpaGUI (module), 108
GSASIImage (module), 131
GSASIImage (module), 101
GSASIIindex (module), 150
GSASIIO (module), 82
GSASIIO (module), 38
GSASIlog (module), 31
GSASIMain() (in module GSASIIdataGUI), 97
GSASIImapvars (module), 121
GSASIImath (module), 134
GSASIImpsubs (module), 58
GSASIIobj (module), 4
GSASIIpath (module), 28
GSASIIphsGUI (module), 98
GSASIIplot (module), 151
GSASIIpwd (module), 159
GSASIIpwdGUI (module), 103
GSASIIpy3 (module), 88
GSASIIrestrGUI (module), 105
GSASIIsaad (module), 165
GSASIscriptable (module), 169
GSASIIspc (module), 47
GSASIstrIO (module), 117
GSASIIstrMain (module), 110
GSASIIstrMath (module), 111
GSASIItestplot (module), 211
GSASIItoolplot (class in GSASIIplot), 154
GGrid (class in GSASIIctrlGUI), 71
GSNoteBook (class in GSASIIctrlGUI), 71
GUIpatches() (in module GSASIIdataGUI), 97
H
halfCell() (in module GSASIIindex), 150
HardSpheresSF() (in module GSASIIsaad), 166
HDF5_Reader (class in G2img_HDF5), 225
Help_mode (in module config_example), 34
HelpButton (class in GSASIIctrlGUI), 72
HessianLSQ() (in module GSASIImath), 137
HessianSVD() (in module GSASIImath), 138
HessRefine() (in module GSASIIstrIO), 112
HillSortElements() (in module G2export_CIF), 232
HistIdLookup (in module GSASIIobj), 21
histogram() (GSASIIscriptable.G2Project method), 197
histograms() (GSASIIscriptable.G2Phase method), 191
histograms() (GSASIIscriptable.G2Project method), 197
histograms() (GSASIIscriptable.G2SeqRefRes method), 205
HistranIdLookup (in module GSASIIobj), 21
HKL2SpAng() (in module GSASIIlattice), 42
HKL2ReaderClass (class in G2fact), 220
HKLMP_ReaderClass (class in G2fact), 220
hold_many() (GSASIIscriptable.G2Project method), 197
HorizontalLine() (in module GSASIIctrlGUI), 72
HowDidIgetHere() (in module GSASIIobj), 22
HstrainNames() (in module GSASIIspc), 49
Hx2Rh() (in module GSASIIlattice), 42

Index

247
Index

image() (GSASIIscriptable.G2Project method), 198
image: Image data object description, 14
imageCalibrants (module), 61
ImageCalibrate() (in module GSASIIimage), 133
ImageCompress() (in module GSASIIimage), 133
ImageIntegrate() (in module GSASIIimage), 133
ImageLocalMax() (in module GSASIIimage), 133
imageMultiDistCalib() (GSASIIscriptable.G2Project method), 198
ImageReccalibrate() (in module GSASIIimage), 133
images() (GSASIIscriptable.G2Project method), 198
import_directory (class in GSASIIimgGUI), 102
import_generic() (in module config_example), 34
import_selfclass (module GSASIIscriptable).
import_imageMultiDistCalib() (GSASIIscriptable.G2Project method), 198
ImportBaseclass (class in GSASIIobj), 22
ImportBaseclass.ImportException, 22
ImportImage (class in GSASIIobj), 22
ImportPDFData (class in GSASIIobj), 23
ImportPhase (class in GSASIIobj), 23
ImportPowderData (class in GSASIIobj), 23
ImportReflectometryData (class in GSASIIobj), 23
ImportSmallAngleData (class in GSASIIobj), 24
ImportStructFactor (class in GSASIIobj), 24
IndexAllIds() (in module GSASIIobj), 24
IndexGFX() (in module GSASIIstrIO), 119
IndexParmDict() (in module GSASIIexprGUI), 108
IndexPeakListSave() (in module GSASIIIO), 86
IndexPeaks() (in module GSASIIIndex), 150
IndexSSPeaks() (in module GSASIIIndex), 150
indexParmList (module GSASIImapvars), 131
InitExport() (GSASIIIO.ExportBaseclass method), 84
InitFobsGfxGlobals() (in module GSASIImpsubs), 59
initMasks() (GSASIIscriptable.G2Image method), 186
InitMP() (in module GSASIImpsubs), 59
InitParameters() (GSASIIobj.ImportImage method), 23
InitParameters() (GSASIIobj.ImportStructFactor method), 24
InitPwdrProfGlobals() (in module GSASIImpsubs), 59
InitVars() (in module GSASIImapvars), 130
InstallGridToolTip() (GSASIIctrlGUI.GSGrid method), 71
Instprm_default (in module config_example), 34
InstrumentParameters (GSASIIscriptable.G2PwdrData attribute), 201
IntegParmTable (class in GSASIIimgGUI), 102
Integrate() (GSASIIscriptable.G2Image method), 185
IntegrateImage() (GSASIIimgGUI.AutoIntFrame method), 102
InterPrecipitateSF() (in module GSASIIasdasd), 166
inarrayList (module GSASIImapvars), 131
invcel12Gmat() (in module GSASIIlattice), 46
InvokeDebugOpts() (in module GSASIIpath), 28
InvokeMenuCommand() (in module GSASIIlog), 32
InvokeTreeItem() (GSASIIplot.G2PlotNoteBook method), 154
invpolfcalc() (in module GSASIIlattice), 46
invV() (in module GSASIImath), 148
IPG() (in module GSASIIasdasd), 166
IPyBreak() (in module GSASIIpath), 28
IPyBreak_base() (in module GSASIIpath), 28
IPyBrowse() (in module GSASIIscriptable), 206
isBound() (gltext.TextElement method), 61
IsHistogramInAnyPhase() (in module GSASIIpwdGUI), 103
ISODISTORT_proc() (G2phase_CIF.CIFPhaseReader method), 217
ItemSelectedListener() (in module GSASIIctrlGUI), 72
iter_refinements() (GSASIIscriptable.G2Project method), 198

J
JANA_ReaderClass (class in G2phase), 216

L
label (GSASIIscriptable.G2AtomRecord attribute), 184
lastError (GSASIIobj.ExpressionObj attribute), 20
Latt2text() (in module GSASIIisp), 49
LaueUnique() (in module GSASIIlattice), 42
LaueUnique2() (in module GSASIIlattice), 42
lblLookup (GSASIIobj.ExpressionCalcObj attribute), 18
link_histogram_phase() (GSASIIscriptable.G2Project method), 198
listLogged (class in GSASIIlog), 33
load_iprms() (in module GSASIIscriptable), 210
load_pwd_from_reader() (in module GSASIIscriptable), 210
loadCIFdict() (in module G2export_CIF), 233
LoadConfigFile() (in module GSASIIpath), 28
loadControls() (GSASIIscriptable.G2Image method), 186
LoadControls() (in module GSASIIfiles), 55
LoadDefaultExpressions() (in module GSASI-lexpGUI), 108
LoadDictFromProjFile() (in module GSASIIscriptable), 207
LoadExportRoutines() (in module GSASIIfiles), 55
LoadExpression() (GSASIIobj.ExpressionObj method), 19
LoadG2fil() (in module GSASIIscriptable), 207
LoadImage() (GSASIIobj.ImportImage method), 23
LoadImage2Tree() (in module GSASIIIO), 86
LoadImportRoutines() (in module GSASIIfiles), 55
loadMasks() (GSASIIscriptable.G2Image method), 186
loadParmDict() (GSASIIIO.ExportBaseclass method), 85
LoadProfile() (GSASIIIO.ExportBaseclass.G2PwdrData method), 201
loadTree() (GSASIIIO.ExportBaseclass method), 85
LogEntry (class in GSASIIlog), 30
logging_debug (in module config_example), 36
LogInfo (in module GSASIIlog), 32
LogNormalCume() (in module GSASIIIsas), 166
LogNormalDist() (in module GSASIIIsas), 166
LogOff() (in module GSASIIlog), 32
LogOn() (in module GSASIIlog), 32
LogVarChange() (in module GSASIIlog), 32
LookupAtomId() (in module GSASIIobj), 24
LookupAtomLabel() (in module GSASIIobj), 25
LookupHistId() (in module GSASIIobj), 25
LookupHistName() (in module GSASIIobj), 25
LookupPhaseId() (in module GSASIIobj), 25
LookupPhaseName() (in module GSASIIobj), 25
LookupWildCard() (in module GSASIIobj), 25
LorchWeight() (in module GSASIIpwpd), 160
LSWCume() (in module GSASIIIsas), 166
LSWDist() (in module GSASIIIsas), 166

M
M90_ReaderClass (class in G2fact), 221
MacStartGSASII() (in module GSASIIpath), 29
MagFF (in module atmdata), 63
MagMod() (in module GSASIImath), 138
MagScatFac() (in module GSASIIelem), 38
MagSSText2MTS() (in module GSASIIspc), 50
MagStructureFactor2() (in module GSASIIstr-Math), 112
MagStructureFactorDerv() (in module GSASIIstr-Math), 113
MagStructureFactorDerv2() (in module GSASIIstr-Math), 113
MagSytsym() (in module GSASIIspc), 50
MagText2MTS() (in module GSASIIspc), 50
main() (in module GSASIIscriptable), 210
main() (in module GSASIIstrMain), 111
main() (in module scanCCD), 212
main() (in module testDeriv), 211
Main_Pos (in module config_example), 34
Main_Size (in module config_example), 34
MakeZThetaAzimuthMap() (in module GSASIIimage), 133
make_empty_project() (in module GSASIIscriptable), 210
make_var_obj() (GSASIIscriptable.G2Project method), 198
makeBat() (module), 212
MakeButtonLog() (in module GSASIIlog), 32
MakeByte2str() (in module GSASIIpath), 29
makeFFTsizeList() (in module GSASIIpwpd), 164
makeLinux() (module), 212
MakeLSParmDict() (GSASIIdataGUI.GSASII method), 92
makeMacApp() (module), 212
makeMat() (in module GSASIIimage), 133
makeMEMfile() (in module GSASIIpwpd), 164
makePRFfile() (in module GSASIIpwpd), 164
MakePWDRfilename() (GSASIIIO.ExportBaseclass method), 84
makeQuat() (in module GSASIImath), 148
makeRing() (in module GSASIIimage), 134
MakeSimSizer() (in module GSASIIpapGUI), 108
MakeTabLog() (in module GSASIIlog), 32
MakeTopasFPASizer() (in module GSASIIpapGUI), 108
MakeTreeLog() (in module GSASIIlog), 32
makeTutorial() (module), 212
MakeUniqueLabel() (in module GSASIIobj), 25
makeWaves() (in module GSASIImath), 148
makeWavesDerv() (in module GSASIImath), 148
Map2Dict() (in module GSASIImapvars), 130
MAR_ReaderClass(class in G2img_MAR), 224
marFrame (class in ReadMarCCDFrame), 88
MaxEnt_SB() (in module GSASIIss), 166
MaxEntException, 166
MaxIndex() (in module GSASIIlattice), 43
mcsaSearch() (in module GSASIImath), 148
MEMupdateRef1Data() (in module GSASIIpwpd), 160
MenuBinding() (GSASIIdataGUI.GSASII method), 92
MenuBindingLookup() (in module GSASIIlog), 32
MenuLogEntry (class in GSASIIlog), 32
MergeDialog (class in GSASIIdataGUI), 97
Modulation() (in module GSASIImath), 138
ModulationDerv() (in module GSASIImath), 138
ModulationPlot() (in module GSASIIpplot), 155
ModulationTw() (in module GSASIImath), 138
Index

monoCellReduce() (in module GSASIindex), 151
MoveConfEquiv() (in module GSASIImapvars), 130
MoveToUnitCell() (in module GSASIIndex), 50
MoveTreeItems() (GSASIdataGUI.GSASIItem method), 92
Movie_fps (in module config_example), 34
Movie_time (in module config_example), 34
MPLsubplots() (in module GSASIPlot), 155
MT2text() (in module GSASIIndex), 49
mu() (GSASIscriptable.G2Phase method), 191
Muiso2Shkl() (GSASIscriptable.G2AtomRecord method), 184
MultiColumnSelection (class in GSASIctrGUI), 73
MultiDataDialog (class in GSASIctrlGUI), 73
MultiIntegerDialog (class in GSASIctrlGUI), 73
MultipleBlockSelector() (in module GSASI- II0), 87
MultipleChoicesDialog (class in GSASHO), 87
MultipleChoicesSelector() (in module GSASII- II0), 87
Multiprocessing_cores (in module config_example), 34
MultiStringDialog (class in GSASIctrlGUI), 73
MustrainCoeff() (in module GSASIIndex), 50
MustrainNames() (in module GSASIIndex), 50
MyHelp (class in GSASIctrlGUI), 74
MyHtmlPanel (class in GSASIctrlGUI), 74

N
NCScattDen() (in module GSASHmath), 139
NIST_hb3a_INT_ReaderClass (class in G2sfact), 221
NISTparms (in module GSASHfpaGUI), 109
norm_gen (class in GSASHpvd), 165
norm() (in module GSASHmath), 148
NT_HKLF2_ReaderClass (class in G2sfact), 221
NT_JANA2K_ReaderClass (class in G2sfact), 221
NumberValidator (class in GSASIctrlGUI), 74

O
objectScan() (in module GSASHIIO), 88
Oblique() (in module GSASHpvd), 160
occupancy (GSASIscriptable.G2AtomRecord attribute), 184
oddPeak() (in module GSASHindex), 151
Odfchk() (in module GSASHlattice), 43
OmitMap() (in module GSASHmath), 139
OnAddPhase() (GSASIdataGUI.GSASImethod), 92
OnAddRow() (G2export_CIF.EditCIFpanel method), 231
OnApplyChanges() (GSASIctrlGUI.SelectConfigSetting method), 78
OnBoolSelect() (GSASIctrlGUI.SelectConfigSetting method), 78
OnChange() (GSASIctrlGUI.SelectConfigSetting method), 78
OnChar() (GSASIctrlGUI.ASCIIValidator method), 65
OnChar() (GSASIctrlGUI.G2MultiChoiceDialog method), 69
OnChar() (GSASIctrlGUI.NumberValidator method), 75
OnChar() (GSASIdataGUI.GSASI.SumDialog method), 97
OnChar() (GSASIexprGUI.ExpressionDialog method), 106
OnCheck() (GSASIctrlGUI.G2MultiChoiceDialog method), 69
OnCheckUpdates() (GSASIctrlGUI.MyHelp method), 74
OnChoice() (GSASIctrlGUI.OrderBox method), 76
OnChoice() (GSASIexprGUI.ExpressionDialog method), 107
OnColMetaTest() (GSASIdataGUI.GSASI method), 93
OnDataDelete() (GSASIdataGUI.GSASI method), 93
OnDataTreeSelChanged() (GSASIdataGUI.GSASI method), 93
OnDeletePhase() (GSASIdataGUI.GSASI method), 93
OnDepChoice() (GSASIexprGUI.ExpressionDialog method), 107
OnDouble() (GSASIimgGUI.ImgIntLstCtrl method), 102
OnDummyPowder() (GSASIdataGUI.GSASI method), 93
OnExportPDF() (GSASIdataGUI.GSASI method), 93
OnFileClose() (GSASIdataGUI.GSASI method), 93
OnFileOpen() (GSASIdataGUI.GSASI method), 93
OnFileSave() (GSASIdataGUI.GSASI method), 93
OnFileSaveas() (GSASIdataGUI.GSASI method), 93
OnFilter() (GSASIdataGUI.GSASI.SumDialog method), 96
OnGPXtreeItemActivated() (GSASI-IdaGUI.GSASI method), 93
OnGPXtreeItemCollapsed() (GSASI-IdaGUI.GSASI method), 93
OnGPXtreeItemDelete() (GSASIdataGUI.GSASI method), 93
OnGPXtreeItemExpanded() (GSASI-IdaGUI.GSASI method), 93
OnGPXtreeKeyDown() (GSASIdataGUI.GSASI method), 93

250
GSAS-II Developers Documentation, Release version 4249

Index 251

P
Panalytical_ReaderClass (class in G2pwd_Panalytical), 219
Parameter dictionary, 17
Parameters (GSASHIobj.ImportStructFactor attribute), 24
paramPrefix (in module GSASIImapvars), 131
parmDict (GSASHIexprGUI.ExpressionDialog attribute), 107
dparmDict (GSASHIobj.ExpressionCalcObj attribute), 19
parmDict (in module GSASHIpaGUI), 109
ParseExpression() (GSASHIobj.ExpressionObj method), 20
PDF_ReaderClass (class in G2phase), 216
pdbBreak() (in module GSASIIpath), 29
pdf() (GSASHIscriptable.G2Project method), 199
PDF_ReaderClass (class in G2phase), 216
Q2Mat() (in module GSASIImath), 139

Reader() (in module GSASIIindex), 139

Reader Class (G2pwd_BrukerRAW), 219

RandomQ() (GSASIIindex.G2PlotNotebook method), 154
ran2axis() (in module GSASIIindex), 151
ranAbyR() (in module GSASIIindex), 151
ranAbyV() (in module GSASIIindex), 151
ranaxis() (in module GSASIIindex), 151
rancell() (in module GSASIIindex), 149
randomAVdeg() (in module GSASIImath), 149
ranId(GSASIIscriptable.G2AtomRecord attribute), 184
raw_ReaderClass (class in G2pwd_BrukerRAW), 219

ReadCheckConstraints() (in module GSASI-Istr10), 119
ReadCIF() (in module GSASIIobj), 26
readColMetadata() (in module GSASIIfiles), 26
readColMetadataLabels() (in module GSASIIfiles), 26
readControls() (in module GSASIIimgGUI), 103
readDataset() (G2img_HDF5.HDF5_Reader method), 225
Reader() (G2img_ITIF.TIF_ReaderClass method), 225
Reader() (G2img_CBF.CBF_ReaderClass method), 225
Reader() (G2img_CheMin.png_ReaderClass method), 225
Reader() (G2img_GE.GE_ReaderClass method), 224
Reader() (G2img_GE.GEsum_ReaderClass method), 224
Reader() (G2img_HDF5.HDF5_Reader method), 225
Reader() (G2img_SFRM.SFRM_ReaderClass method), 226
Reader() (G2img_SumG2.G2_ReaderClass method), 223
Reader() (G2phase.EXP_ReaderClass method), 216
Reader() (G2phase.JANA_ReaderClass method), 216
Reader() (G2phase.PDB_ReaderClass method), 216
Reader() (G2phase.PDF_ReaderClass method), 217
Reader() (G2phase_GPX.PhaseReaderClass method), 217
Reader() (G2phase_INS.PhaseReaderClass method), 217
Reader() (G2pwd_BrukerRAW.raw_ReaderClass method), 219
Reader() (G2pwd_CIF.CIFpwdReader method), 219
Reader() (G2pwd_csv.cswcsw_ReaderClass method), 220
Reader() (G2pwd_FP.fp_ReaderClass method), 219
Reader() (G2pwd_fxye.GSAS_ReaderClass method), 218
Reader() (G2pwd_GPX.GSAS2_ReaderClass method), 218
Reader() (G2pwd_Panalitical.Panalitical_ReaderClass method), 219
Reader() (G2pwd_igaku.Rigaku_rasReaderClass method), 220
Reader() (G2pwd_igaku.Rigaku_txtReaderClass method), 220
Reader() (G2pwd_xye.xye_ReaderClass method), 218
Reader() (G2sfact.HKL_ReaderClass method), 220
Reader() (G2sfact.M90_ReaderClass method), 221
Reader() (G2sfact.NIST_hb3a_INT_ReaderClass method), 221
Reader() (G2sfact.NT_JANA2K_ReaderClass method), 221
Reader() (G2sfact.NT_JANA2K_ReaderClass method), 221
Reader() (G2sfact.NT_JANA2K_ReaderClass method), 221
Reader() (G2sfact.NIST_hb3a_INT_ReaderClass method), 221
ReadEXPParse() (G2phase.EXP_ReaderClass method), 216
ReadFiles() (GSASIIimgGUI.IntegParmTable method), 103
ReadIntegParmTable() (GSASIIimgGUI.IntegParmTable method), 103
ReadImages() (in module GSASIIfiles), 87
ReadINSPhase() (G2phase_INS.PhaseReaderClass method), 217
ReadJANAPhase() (G2phase.JANA_ReaderClass method), 216
ReadMarCCDFrame (module), 88
ReadMask() (in module GSASIIimgGUI), 103
readMasks() (in module GSASIIfiles), 58
ReadPDFPhase() (G2phase.PDB_ReaderClass method), 216
ReadPDFPhase() (G2phase.PDF_ReaderClass method), 217
ReaderInstprm() (GSASIIdataGUI.GSASII method), 96
ReaderInstprm() (in module GSASIIfiles), 55
ReadPowderInstprm() (GSASIIdataGUI.GSASII method), 96
Recalibrate() (GSASIIscriptable.G2Image method), 185
scaleAbyV() (in module GSASIIindex), 151
scanCCD (class in scanCCD), 212
scanCCD (module), 211
scanCCDmain (class in scanCCD), 212
ScatFac() (in module GSASIIElem), 38
SECextration() (in module GSASIIstrMath), 113
SchulzZimmCume() (in module GSASIIssas), 167
SchulzZimmDist() (in module GSASIIssas), 167
ScrollMultiEditor (class in GSASIIctrlGUI), 76
SearchMap() (in module GSASIImath), 140
sec2HMS() (in module GSASIIlattice), 46
SelectAndDownload() (GSASIIctrlGUI.OpenTutorial method), 76
SelectConfigSetting (class in GSASIIctrlGUI), 78
SelectDataTreeWidgetItem() (in module GSASIIdataGUI), 97
SelectDownloadLoc() (GSASIIctrlGUI.OpenTutorial method), 76
SelectEdit1Var() (in module GSASIIctrlGUI), 78
SelectG2var() (GSASIIexprGUI.ExpressionDialog method), 107
selections() (in module GSASIIlattice), 46
selftestlist (in module GSASIIlattice), 46
selftestlist (in module GSASIIisp), 53
seqref() (GSASIIscriptable.G2Project method), 199
SeqRefine() (in module GSASIIstrMain), 111
set_background() (GSASIIscriptable.G2PDF method), 188
set_background() (GSASIIscriptable.G2PwdrData method), 203
set_Cons() (GSASIIscriptable.G2Project method), 199
set_formula() (GSASIIscriptable.G2PDF method), 189
set_HAP_refinements() (GSASIIscriptable.G2Phase method), 191
set_peakFlags() (GSASIIscriptable.G2PwdrData method), 203
set_refinement() (GSASIIscriptable.G2Project method), 200
set_refinements() (GSASIIscriptable.G2Phase method), 192
set_refinements() (GSASIIscriptable.G2PwdrData method), 204
SetBackgroundParms() (in module GSASIIpwd), 161
SetBinaryPath() (in module GSASIIpath), 29
setCalibrant() (GSASIIscriptable.G2Image method), 186
setControlFile() (GSASIIscriptable.G2Image method), 186
setControls() (GSASIIscriptable.G2Image method), 187
SetCopyNames() (in module GSASIIpwdGUI), 104
SetCu2Wave() (in module GSASIIspdGUI), 109
SetDataMenuBar() (in module GSASIIdataGUI), 97
SetDataSize() (GSASIIdataGUI.G2DataWindow method), 90
SetDataSize() (GSASIIdataGUI.GSASIImethod), 96
setDefaultData() (in module GSASIIscriptable), 208
setDefaultRFDM() (in module GSASIIpwdGUI), 104
setDefaultSample() (in module GSASIIobj), 26
setDefaultSASDModel() (in module GSASIIpwdGUI), 104
setDefaultSubstances() (in module GSASIIpwdGUI), 104
SetDepVar() (GSASIIobj.ExpressionObj method), 20
SetDrawingDefault() (in module GSASIIphsGUI), 99
setEvalResult() (GSASIIexprGUI.ExpressionDialog method), 107
setFont() (gltext.Text method), 59
setFont_size() (gltext.Text method), 60
setForeground() (gltext.Text method), 60
setHAPvalues() (GSASIIscriptable.G2Phase method), 191
SetHistogramData() (in module GSASIIstro), 120
SetHistogramPhaseData() (in module GSASIIstro), 120
setMaskes() (GSASIIscriptable.G2Image method), 187
SetMolCent() (in module GSASIImath), 140
SetNewPhase() (in module GSASIIobj), 26
SetNoDelete() (GSASIIPlot.G2PlotNoteBook method), 154
setPeakparms() (in module GSASIImath), 149
SetPhaseData() (in module GSASIIstro), 120
SetPowderInstParms() (in module GSASIIfiles), 56
SetPrintLevel() (in module GSASIIscriptable), 208
SetRange() (GSASIIctrlGUI.G2MultiChoiceDialog method), 69
SetRigidBodyModels() (in module GSASIIstro), 120
SetSelectionNoRefresh() (GSASIIPlot.G2PlotNoteBook method), 154
SetSelections() (GSASIIctrlGUI.G2MultiChoiceDialog method), 69
SetSeqRef() (GSASIIIO.ExportBaseclass method), 84
SetSeqResult () (in module GSASIistrIO), 120
SetSourceDir () (GSASIimgGUI.AutoInitFrame method), 102
svnProxy () (in module GSASIpath), 30
setText () (gtext.Text method), 60
SetTitleByGPX () (GSASIdataGUI.GSASII method), 96
SetTutorialPath () (GSASIctrlGUI.OpenTutorial method), 76
SetupCalc () (GSASIobj.ExpressionCalcObj method), 18
SetupFFAcalc () (in module GSASIpagaGUI), 98
SetUpFullrdmcDialog (class in GSASIphsGUI), 99
SetUpGeneral () (in module GSASIscriptable), 208
SetUpRMCPProfileDialog (class in GSASIphs-GUI), 99
SetupSampleLabels () (in module GSASIppwdGUI), 104
SetupSeqSavePhases () (in module GSASIppwdGUI), 120
SetUsedHistogramsAndPhases () (in module GSASIistrIO), 120
setVary () (GSASIscriptable.G2Image method), 187
SetVersionNumber () (in module GSASIpath), 29
sfloat () (in module GSASIpxe), 218
sfloat () (in module GSASIfiles), 58
sfloat () (in module GSASIIO), 88
SFRM_ReaderClass (class in G2img_SFRM), 226
sgequiv_2002_orthorhombic (in module GSASIlpiscspc), 53
SErrors () (in module GSASIiscspc), 50
SGMagSpinBox (class in GSASIctrlGUI), 76
SGMessageBox (class in GSASIctrlGUI), 76
SGpolar () (in module GSASIiscspc), 51
SGPrint () (in module GSASIiscspc), 50
SGProd () (in module GSASIiscspc), 50
SGPGroup () (in module GSASIiscspc), 50
SHELX4_ReaderClass (class in G2sfact), 221
SHELX5_ReaderClass (class in G2sfact), 221
SHELX6_ReaderClass (class in G2sfact), 221
ShortHistNames (in module GSASIobj), 26
ShortPhaseNames (in module GSASIobj), 26
Show () (GSASIctrlGUI.MultiStringDialog method), 74
Show () (GSASIctrlGUI.SGMagSpinBox method), 76
Show () (GSASIctrlGUI.GSMessageBox method), 76
Show () (GSASIctrlGUI.SingleStringDialog method), 80
Show () (GSASIexprGUI.ExpressionDialog method), 107
Show _tim ing (in module config_example), 35
ShowBanner () (in module GSASIistrIO), 120
ShowControls () (in module GSASIistrIO), 121
showError () (GSASIexprGUI.ExpressionDialog method), 107
ShowHelp () (in module GSASIctrlGUI), 79
ShowLogStatus () (in module GSASIistrlog), 33
ShowLSParms () (class in GSASIctrlGUI), 79
ShowMatchingFiles () (GSASIimgGUI.AutoInitFrame method), 102
ShowStringValidity () (GSASIctrlGUI.ValidatedTxtCtrl method), 82
ShowTiming () (in module GSASIIobj), 26
ShowValidity () (GSASIctrlGUI.NumberValidator method), 75
ShowVersions () (in module GSASIdataGUI), 98
ShowWebPage () (in module GSASIctrlGUI), 79
SHPOcal () (in module GSASIistrMath), 113
SHPOcalDrv () (in module GSASIistrMath), 113
SHTXcal () (in module GSASIistrMath), 113
SHTXcalDrv () (in module GSASIistrMath), 113
spParms () (in module GSASIpagaGUI), 109
Single Crystal data object description, 13
Single Crystal reflection object description, 14
SingleFloatDialog (class in GSASIctrlGUI), 79
SingleIntDialog (class in GSASIctrlGUI), 79
SingleStringDialog (class in GSASIctrlGUI), 80
sint () (in module GSASIpxe), 218
sint () (in module GSASIIO), 88
sortArray () (in module GSASImath), 149
sortHKLd () (in module GSASIindex), 46
sortM02 () (in module GSASIindex), 151
SortVariables () (in module GSASIIobj), 26
Space Group Data description, 9
SpaceGroup () (in module GSASIiscspc), 51
SpGroup () (in module GSASIiscspc), 52
spg2origins () (in module GSASIiscspc), 54
spgbyNum () (in module GSASIiscspc), 54
spglist () (in module GSASIiscspc), 54
SphereEnclosure (class in GSASIphsGUI), 99
SphereFF () (in module GSASIlsads), 167
SphereVol () (in module GSASIlsads), 167
SphericalShellFF () (in module GSASIlsads), 167
SphericalShellVol () (in module GSASIlsads), 167
SpheroffDFF () (in module GSASIlsads), 167
SpheroidVol () (in module GSASIlsads), 167
splitSSsym () (in module GSASIiscspc), 54
Spot_mask_diameter (in module config_example), 35
SquareWellSF () (in module GSASIlsads), 167
SsChargeFlip () (in module GSASImath), 140
SSChoice () (in module GSASIiscspc), 51
SSModCheck () (in module GSASIiscspc), 51
SGPrint () (in module GSASIiscspc), 51
SSLatt2text () (in module GSASIiscspc), 51
SMT2text () (in module GSASIiscspc), 51
Tick_width (in module config_example), 35
TIF_ReaderClass (class in G2img_ITIF), 224
TLS2Uij() (in module GSASIImath), 140
TOF2dsp() (in module GSASIIlattice), 43
ToggleMultiSpotMask() (in module GSASIIplot), 159
Trans2Text() (in module GSASIIspc), 53
TransConstraints() (in module GSASIIcontr-GUI), 101
TransferFromWindow() (GSSIctrl-GUI.ASCIIValidator method), 65
TransferFromWindow() (GSSIctrl-GUI.NumberValidator method), 75
TransferToWindow() (GSSIctrl-GUI.ASCIIValidator method), 65
TransferToWindow() (GSSIctrl-GUI.NumberValidator method), 75
TransformCell() (in module GSASIIlattice), 43
TransformDialog (class in GSASIIlattPSGSI), 100
TransformPhase() (in module GSASIIlattice), 43
Transmission() (in module GSASIIlattPWP), 162
Transpose (in module config_example), 35
transposeHKLF() (in module GSASIIlattice), 47
TreeLogEntry (class in GSASIIlog), 33
trim() (in module GSASIIHIO), 88
Tutorial_location (in module config_example), 35
tutorialIndex (in module GSASIIctrlGUI), 82
txt_CWNeutronReaderClass (class in G2sad_xye), 222
txt_NeutronReaderClass (class in G2sad_xye), 226
txt_nmCWNeutronReaderClass (class in G2sad_xye), 222
txt_nmXRayReaderClass (class in G2sad_xye), 226
txt_PDFReaderClass (class in G2pdf_gr), 226
txt_PDFReaderClassG (class in G2pdf_gr), 226
txt_XRayReaderClass (class in G2sad_xye), 227
txt_XRayReaderClass (class in G2sad_xye), 222
txt_XRayThetaReaderClass (class in G2rfd_xye), 227
type (GSASIIscriptable.G2AtomRecord attribute), 184
U
U6toUij() (in module GSASIIlattice), 44
Uij2betaij() (in module GSASIIlattice), 44
Uij2Ueqv() (in module GSASIIlattice), 44
UijtoU6() (in module GSASIIlattice), 44
uiso (GSASIIscriptable.G2AtomRecord attribute), 184
UniDiskFF() (in module GSASIIlattPSGSI), 168
UniDiskVol() (in module GSASIIlattPSGSI), 168
uniqueCombinations() (in module GSASIIlattice), 47
UniRodARFF() (in module GSASIIlattPSGSI), 168
UniRodARVol() (in module GSASIIlattPSGSI), 168
UniRodFF() (in module GSASIIlattPSGSI), 168
UniRodVol() (in module GSASIIlattPSGSI), 168
UniSphereFF() (in module GSASIIlattPSGSI), 168
UniSphereVol() (in module GSASIIlattPSGSI), 168
UniRodARVol() (in module GSASIIlattPSGSI), 168
UniRodFF() (in module GSASIIlattPSGSI), 168
UniRodVol() (in module GSASIIlattPSGSI), 168
UniSphereFF() (in module GSASIIlattPSGSI), 168
UniSphereVol() (in module GSASIIlattPSGSI), 168
unit_tests (module), 213
UniTubeFF() (in module GSASIIlattPSGSI), 168
UniTubeVol() (in module GSASIIlattPSGSI), 168

update_ids() (GSASIIscriptable.G2Project method), 200
UpdateBackground() (in module GSASIIpwdGUI), 104
UpdateComments() (in module GSASIIdataGUI), 98
UpdateConstraints() (in module GSASIIcontr-GUI), 101
UpdateControllers() (in module GSASIIdataGUI), 98
UpdateData() (in module GSASIIdataGUI), 100
UpdateDict() (GSASIIobj.ExpressionCalcObj method), 18
UpdateDownloaded() (GSSIctrl-GUI.OpenTutorial method), 76
UpdateImageControls() (in module GSASIIimg-GUI), 103
UpdateImageLoc() (GSSIctrlGUI.G2TreeCtrl method), 70
UpdateIndexPeaksGrid() (in module GSASIIpwdGUI), 104
UpdateInstrumentGrid() (in module GSASIIpwdGUI), 104
UpdateLimitsGrid() (in module GSASIIpwdGUI), 104
UpdateMasks() (in module GSASIIimg-GUI), 103
UpdateMCASxyz() (in module GSASIIlattPSGSI), 140
UpdateModelsGrid() (in module GSASIIpwdGUI), 104
UpdateNotebook() (in module GSASIIdataGUI), 98
UpdateParameters() (GSASI-obj.ImportStructFactor method), 24
UpdatePDFGrid() (in module GSASIIpwdGUI), 104
UpdatePeakGrid() (in module GSASIIpwdGUI), 104
UpdatePhaseData() (in module GSASIIphsGUI), 100
UpdatePolygon() (in module GSASIIplot), 159
UpdatePWHKPlot() (in module GSASIIdataGUI), 98
UpdateRBXYZ() (in module GSASIIphsGUI), 140
UpdateRBXYZ() (in module GSASIImath), 140
UpdateREFDModelsGrid() (in module GSASIIpwdGUI), 104
UpdateReflectionGrid() (in module GSASIIpwdGUI), 104
UpdateRestraints() (in module GSASIIrestrGUI), 105
UpdateRigidBodies() (in module GSASIIconstrGUI), 101
UpdateSampleGrid() (in module GSASIIpwdGUI), 105
UpdateSeqResults() (in module GSASIIdataGUI), 98
UpdateStressStrain() (in module GSASIIimgGUI), 103
UpdateSubstanceGrid() (in module GSASIIpwdGUI), 105
UpdateSytSym() (in module GSASIImsp), 53
UpdateUnitCellsGrid() (in module GSASIIpwdGUI), 105
UpdateVariedVars() (GSASIIObj.ExpressionObj method), 20
UpdateVars() (GSASIIObj.ExpressionCalcObj method), 18
usedVars (GSASIIexprGUI.ExpressionDialog attribute), 107
UseMagAtomDialog (class in GSASIIphysGUI), 100

V
ValEsd() (in module GSASIImath), 140
ValidateAscii() (G2export_CIF.ExportCIF method), 232
ValidatedTxtCtrl (class in GSASIICtrlGUI), 80
Values2A() (in module GSASIIIndex), 150
Values2Dict() (in module GSASIIpwd), 162
Values2Dict() (in module GSASIIstrMath), 116
VarDescr() (in module GSASIIObj), 27
VarKeys() (in module GSASIImapvars), 130
VarLogEntry (class in GSASIIlog), 33
varLookup (GSASIIObj.ExpressionCalcObj attribute), 19
varName (GSASIIObjG2VarObjG2VarObj attribute), 108
varname() (GSASIIObj.G2VarObj method), 21
varRefflag (GSASIIObjG2VarObjG2VarObj attribute), 108
VarRemapShow() (in module GSASIImapvars), 130
varSelect (GSASIIObjG2VarObjG2VarObj attribute), 108
varValue (GSASIIObjG2VarObjG2VarObj attribute), 108
visit() (G2img_HDF5.HDF5_Reader method), 225
Vol2Den() (in module GSASIImath), 141

W
wavekE() (in module GSASIImath), 150
whichsvn() (in module GSASIIpath), 31
Write() (G2export_map.ExportMapCCP4 method), 229
Write() (GSASIIIO.ExportBaseclass method), 84
Write2csv() (in module GSASIIplot), 159
WriteAtomsMagnetic() (in module G2export_CIF), 233
WriteAtomsNuclear() (in module G2export_CIF), 233
WriteCIFItem() (in module G2export_CIF), 233
WriteComposition() (in module G2export_CIF), 233
WriteControls() (in module GSASIIfiles), 56
WriteInstFile() (G2export_pwdr.ExportPowderFXYE method), 234
writer() (G2export_CIF.ExportPowderFXYE method), 232
writer() (G2export_pwdr.ExportPowderFXYE method), 234
wxInspector (in module config_example), 36

X
XferFPsettings() (in module GSASIIfpagGUI), 109
XScattDen() (in module GSASIImath), 141
xye.READERCLASS (class in G2pwd_xye), 218
XYsave() (in module GSASIIO), 88

Y
y_calc() (GSASIIScriptable.G2PwdrData method), 204

Index 259