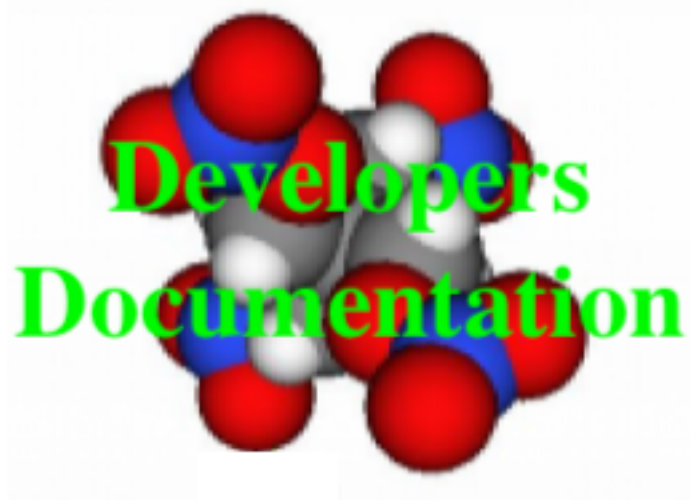

GSAS-II



GSAS-II Scripting Documentation

Release bfae21 17-May-2024 21:54 (#5786)

Robert B. Von Dreele and Brian H. Toby

May 18, 2024

CONTENTS

| | | |
|----------|--|-----------|
| 1 | GSAS-II Requirements, Python Packages and External Software | 3 |
| 1.1 | Supported Platforms | 3 |
| 1.2 | Version Control | 4 |
| 1.3 | Python Requirements | 4 |
| 1.4 | GUI Requirements | 5 |
| 1.5 | Scripting Requirements | 6 |
| 1.6 | Optional Python Packages | 7 |
| 1.7 | Required Binary Files | 7 |
| 1.8 | Supported Externally-Developed Software | 7 |
| 2 | <i>GSASIIobj: Data objects & Docs</i> | 11 |
| 2.1 | <i>Summary/Contents</i> | 11 |
| 2.2 | Variable names in GSAS-II | 12 |
| 2.3 | Constraints Tree Item | 15 |
| 2.4 | Covariance Tree Item | 16 |
| 2.5 | Phase Tree Items | 17 |
| 2.6 | Rigid Body Objects | 21 |
| 2.7 | Space Group Objects | 21 |
| 2.8 | Phase Information | 23 |
| 2.9 | Powder Diffraction Tree Items | 25 |
| 2.10 | Powder Reflection Data Structure | 28 |
| 2.11 | Single Crystal Tree Items | 29 |
| 2.12 | Single Crystal Reflection Data Structure | 29 |
| 2.13 | Image Data Structure | 30 |
| 2.14 | Parameter Dictionary | 32 |
| 2.15 | Texture implementation | 33 |
| 2.16 | ISODISTORT implementation | 34 |
| 2.17 | Parameter Limits | 35 |
| 2.18 | <i>GSASIIobj Classes and routines</i> | 36 |
| 3 | <i>GSASIIscriptable: Scripting Interface</i> | 55 |
| 3.1 | <i>Summary/Contents</i> | 55 |
| 3.2 | Installation of GSASIIscriptable | 56 |
| 3.3 | Application Interface (API) Summary | 56 |
| 3.4 | Refinement parameters | 63 |
| 3.5 | Specifying Refinement Parameters | 65 |
| 3.6 | Access to other parameter settings | 68 |
| 3.7 | Code Examples | 70 |
| 3.8 | GSASIIscriptable Command-line Interface | 81 |
| 3.9 | API: Complete Documentation | 82 |

| | | |
|----------|--|------------|
| 4 | References to the GSAS-II Developer's Documentation | 127 |
| 5 | References to the GSAS-II Developer's Documentation | 129 |
| | Python Module Index | 131 |
| | Index | 133 |

This is a subset of the [Developer's Documentation](#) covering the *GSASIIscriptable* package, used to create Python scripts for running GSAS-II. Note that most data structures used in GSAS-II are defined in module *GSASIIobj*, also included here.

GSAS-II REQUIREMENTS, PYTHON PACKAGES AND EXTERNAL SOFTWARE

1.1 Supported Platforms

It should be possible to run GSAS-II on any computer where Python 3.7+ and the appropriate required packages are available, as discussed below, but GSAS-II also requires that some code must be compiled. For the following platforms, binary images for this compiled code are currently provided:

- Windows-10: 64-bit Intel-compatible processors
- MacOS: Intel processors
- MacOS: ARM processors, aka Apple Silicon (M1, etc)
- Linux: 64-bit Intel-compatible processors
- Linux: ARM processors (64-bit Raspberry Pi OS only)

Details for GSAS-II use on these specific platforms follows below:

- **Windows:** self-Installation kits are provided for 64-bit Windows-10 and -11 [here](#). Less testing has been done with Windows-11, but both appear to working interchangeably with respect to GSAS-II.

In theory it should be possible to run GSAS-II on older versions of Windows, including 32-bit OS versions, but no current installation kit can be provided. Installing GSAS-II will require locating a compatible version (or compiling) Python and the required packages. It may be necessary to recompile the GSAS-II binaries.

- **MacOS:** GSAS-II can run natively on Intel (or ARM (“M1”-“M3” aka “Apple Silicon”) processors with relatively current versions of MacOS, with self-installers that can be run from the command-line available for download [here](#). The Intel version will run on both types of Mac processors, but the native ARM versions offer the highest GSAS-II performance we see on any platform.

It appears that this installer can be used with MacOS versions 11.0 and later. Macs older than Catalina (10.15) will likely require older distributions of Python.

- **Intel Linux:** Note that GSAS-II does not get a lot of testing in Linux by us, but is used fairly widely on this platform nonetheless. We provide an installer [here](#) that includes Python and needed packages for Intel-compatible Linuxes, but compatibility with older and very new versions of Linux can sometimes be tricky as compatibility libraries may be needed – not always easy to do. It may be better to use your Linux distribution’s versions of Python and packages (typically done with a software tool such as apt or yum or pip. See <https://advancedphotonsource.github.io/GSAS-II-tutorials/install-pip.html> for more information.
- **Non-Intel Linux:** Will GSAS-II run on Linux with other types of CPUs? That will mostly depend on support for Python and wxPython on that CPU. If those can be used, you can likely build the GSAS-II binaries with gcc & gfortran. Expect to modify the SConstruct file.

Raspberry Pi (ARM) Linux: GSAS-II has been installed on both 32-bit and the 64-bit version of the Raspberry Pi OS (formerly called Raspbian) and compiled binaries are provided at present for both, but the 32-bit support may not continue. It is expected that these binaries will also function on Ubuntu Linux for Raspberry Pi, but this has not been tried. The performance of GSAS-II on a Raspberry Pi is not blindingly fast, but one can indeed run GSAS-II on a motherboard that costs only \$15 (perhaps even one that costs \$5) and uses <5 Watts!

Note that the 64-bit OS is preferred on the models where it can be run (currently including models 3A+, 3B, 3B+, 4, 400, CM3, CM3+, CM4, and Zero 2 W) . With the 32-bit Raspberry Pi OS, which does run on all Raspberry Pi models, it is necessary to use the OS distribution's versions of Python and its packages, [see here for more information](#). With 64-bit Pi OS it may be possible for us to provide a GSAS2FULL installer (which will need to include a custom-supplied wxPython wheel, since that is not available in conda-forge) or else pip must be used to download and build wx. Please let us know if you are intending to use GSAS-II on a Raspberry Pi for a classroom, etc and would need this.

1.2 Version Control

The master version of the source code for GSAS-II resides on GitHub at URL and the git version control system (VCS) is usually used to install the files needed by GSAS-II. When GSAS-II is installed in this manner, the software can be easily updated, as git commands can download only the changed sections of files that need to be updated. It is likewise possible to use git to regress to an older version of GSAS-II, though there are some limitations on how far back older versions of GSAS-II will be with current versions of Python. While git is not required for use of GSAS-II, special procedures must be used to install GSAS-II without it and once installed without git, updates of GSAS-II must be done manually.

We are currently in a transition period to GitHub from a previous subversion server. As we migrate to GitHub, updates will be made in parallel to both servers.

1.3 Python Requirements

GSAS-II requires a standard Python interpreter to be installed, as well as several separately-developed packages that are not supplied with Python, as are described below. While for some packages, we have not seen much dependence on versions, for others we do find significant differences; this is also discussed further below. The GSAS-II GUI will warn about Python and packages versions that are believed to be problematic, as defined in variable `GSASIIdataGUI.versionDict`, but for new installations we are currently recommending the following interpreter/package versions:

- Python 3.11 is recommended. GSAS-II should run with any Python version from 3.7 or later, but you will need to locate (from the old subversion server) or locate binaries to match that Python version.
- wxPython 4.2 or later is recommended, but with Python <=3.9 any wx4.x version should be OK. However, there may be problems with newer sections of the GUI with wx <4.0.
- NumPy 1.26 recommended, but anything from 1.17 on is likely fine, but does need to approximately match the GSAS-II binaries.
- matplotlib 3.6 is recommended, but 3.4 or later is preferred.
- pyOpenGL: no version-related problems have been seen.
- SciPy: no version-related problems have been seen, but in at least one case multiple imports are tried to account for where function names have changed.

For more details on problems noted with specific versions of Python and Python packages, see comments below and details here: `GSASIIdataGUI.versionDict`,

Note that GSAS-II is currently being developed using Python 3.11. We are seeing compilation problems with Python 3.12 that will be addressed later via the build migration to meson. We are no longer supporting Python 2.7 and <=3.6,

and strongly encourage that systems running GSAS-II under these older Python versions reinstall Python. Typically this is done by reinstalling GSAS-II from a current self-installer.

There are a number of ways to install Python plus the packages needed by GSAS-II. See <https://advancedphotonsource.github.io/GSAS-II-tutorials/install.html> and links therein for a discussion of installation.

Python package requirements depend on how GSAS-II will be run, as will be discussed in the next section. In order to run the GUI for GSAS-II, a much larger number of packages are required. Several more packages are optional, but some functionally will not be available without those optional packages. Far fewer packages are required to run GSAS-II on a compute server via the scripting interface and without a GUI.

1.4 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 various wxPython versions over the years. We encourage use of 4.x with Python 3.x, but with Py>=3.10 you must use wxPython 4.2.0 or later.
- 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>).

GSAS-II will not start if the above packages are not available. In addition, several Python packages are referenced in sections of the GUI code, but are not required. If these packages are not present, warning messages may be generated if they would be needed, or menu items may be omitted, but the vast bulk of GSAS-II will function normally. These optional packages are:

- gitpython: (<https://gitpython.readthedocs.io> and <https://github.com/gitpython-developers/GitPython>). This package provides a bridge between the git version control system and Python. It is required for the standard GSAS-II installation process and for GSAS-II to update itself from GitHub. If your computer does not already have git in the path, also include the git package to obtain that binary (if you are not sure, it does not hurt to do this anyway).
- requests: this package simplifies http access (<https://requests.readthedocs.io/>). It is used for access to webpages such as ISODISTORT and for some internal software downloads. It is required for support of git updating and installation.
- Pillow (<https://pillow.readthedocs.org>) or PIL (<http://www.pythonware.com/products/pil/>). This is used to read and save 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. This is optional and is offered for plotting superspace (modulated) structures.
- win32com (windows only): this module is used to install GSAS-II on windows machines. GSAS-II can be used on Windows without this, but the installation will offer less integration into Windows. Conda provides this under the name pywin32.
- conda: the conda package allows access to package installation, etc. features from inside Python. It is not required but is helpful to have, as it allows GSAS-II to install some packages that are not supplied initially. The conda package is included by default in the base miniconda and anaconda installations, but if you create an environment for GSAS-II (`conda create -n <env> package-list...`), it will not be added to that environment unless you request it specifically.

Conda command:

Should you wish to install Python and the desired packages yourself, this is certainly possible. For Linux, apt or yum is an option, as is homebrew. Homebrew is a good option on MacOS. However, we recommend use of the miniforge self-installers from conda-forge. Here is a typical conda command used to install a GSAS-II compatible Python interpreter after miniforge has been installed:

```
conda install python=3.11 numpy=1.26 wxpython scipy matplotlib pyopengl pillow_
↳h5py imageio requests git gitpython -c conda-forge
```

for development environments, it is useful to have build and debugging tools available, so here is a more extensive list of useful packages:

```
conda create -n py311 python=3.11 numpy=1.26 matplotlib scipy wxpython pyopengl_
↳imageio h5py hdf5 pillow requests ipython conda spyder-kernels scons sphinx_
↳sphinx-rtd-theme jupyter git gitpython -c conda-forge
```

To find out what packages have been directly installed in a conda environment this command can be used:

```
conda env export --from-history -n <env>
```

1.5 Scripting Requirements

The GSAS-II scripting interface (*GSASIIscriptable*) will not run without two Python extension packages:

- NumPy (<http://docs.scipy.org/doc/numpy/reference/>),
- SciPy (<http://docs.scipy.org/doc/scipy/reference/>).

These fortunately are common and are easy to install. There are some relatively minor scripting capabilities that will only run when a few additional packages are installed:

- matplotlib (<http://matplotlib.org/contents.html>),
- Pillow (<https://pillow.readthedocs.org>) and/or
- h5py and hdf5

but none of these are required to run scripts and the vast majority of scripts will not need these packages.

Installing a minimal Python configuration:

There are many ways to install a minimal Python configuration. Below, I show some example commands used to install using the the free miniconda installer from Anaconda, Inc., but I now tend to use the Conda-Forge miniforge distributions instead. However, there are also plenty of other ways to install Python, Numpy and Scipy, depending on if they will be used on Linux, Windows and MacOS. For Linux, the standard Linux distributions provide these using yum or apt-get etc., but these often supply package versions that are so new that they probably have not been tested with GSAS-II.

```
bash ~/Downloads/Miniconda3-latest-<platform>-x86_64.sh -b -p /loc/pyg2script
source /loc/pyg2script/bin/activate
conda install numpy scipy matplotlib pillow h5py hdf5
```

Some discussion on these commands follows:

- the 1st command (bash) assumes that the appropriate version of Miniconda has been downloaded from <https://docs.conda.io/en/latest/miniconda.html> and /loc/pyg2script is where I have selected for python to be installed. You might want to use something like ~/pyg2script.
- the 2nd command (source) is needed to access Python with miniconda.

- the 3rd command (conda) installs all possible packages that might be used by scripting, but matplotlib, pillow, and hdf5 are not commonly needed and could be omitted.

Once Python has been installed and is in the path, use these commands to install GSAS-II:

```
git clone https://github.com/AdvancedPhotonSource/GSAS-II.git /loc/GSAS-II
python /loc/GSAS-II/GSASII/GSASIIscriptable.py
```

Notes on these commands:

- the 1st command (git) is used to download the GSAS-II software. /loc/GSASII is the location where I decided to install the software. You can select something different.
- the 2nd command (python) is used to invoke GSAS-II scriptable for the first time, which is needed to load the binary files from the server.

1.6 Optional Python Packages

- Sphinx (<https://www.sphinx-doc.org>) is used to generate the documentation you are currently reading. Generation of this documentation is not generally something needed by users or even most code developers, since the prepared documentation on <https://gsas-ii.readthedocs.io> is usually reasonably up to date.
- SCons (<https://scons.org/>) is used to compile the relatively small amount of Fortran code that is included with GSAS-II. Use of this is discussed in the next section of this chapter.

1.7 Required Binary Files

As noted before, GSAS-II also requires that some code be compiled. For the following platforms, binary images are provided at <https://github.com/AdvancedPhotonSource/GSAS-II-buildtools/releases/latest> for Python 3.11 and NumPy 1.26:

- Windows-10: 64-bit Intel-compatible processors.
- MacOS: Intel processors.
- MacOS: ARM processors, aka Apple Silicon (M1, etc).
- Linux: 64-bit Intel-compatible processors.
- Linux: ARM processors (64-bit and 32-bit Raspberry Pi OS and Ubuntu for Raspberry Pi).

Note that these binaries must match the major versions of both Python and NumPy; Should one wish to run GSAS-II where binary files are not supplied (such as 32-bit Windows or Linux) or with other combinations of Python/NumPy, compilation will be need to be done by the user. See the [compilation information](#) for more information.

1.8 Supported Externally-Developed Software

GSAS-II provides interfaces to use a number of programs developed by others. Some are included with GSAS-II and others must be installed separately. When these programs are accessed, citation information is provided as we hope that users will recognize the contribution made by the authors of these programs and will honor those efforts by citing that work in addition to GSAS-II.

GSAS-II includes copies of the following programs. No additional steps beyond a standard installation are needed to access their functionality.

DIFFaX

Simulate layered structures with faulting. <https://www.public.asu.edu/~mtreacy/DIFFaX.html>

PyCifRW

A software library that reads and writes files using the IUCr's Crystallographic Information Framework (CIF). <https://bitbucket.org/jamesrhester/pycifrw>. GSAS-II uses this to read data and structures from CIF files,

Shapes

Derives the shapes of particles from small angle scattering data.

NIST FPA

Use Fundamental Parameters to determine GSAS-II profile function

NIST*LATTICE

Searches for higher symmetry unit cells and possible relationships between unit cells. An API has been written and this will be integrated into the GSAS-II GUI.

pybaselines

Determines a background for a powder pattern in the "autobackground" option. See <https://pybaselines.readthedocs.io> for more information.

The following web services can also be accessed from computers that have internet access. All software needed for this access is included with GSAS-II.

Bilbao Crystallographic Server (<https://www.cryst.ehu.es>):

GSAS-II can directly access the Bilbao Crystallographic Server to utilize the k-SUBGROUPSMAG, k-SUBGROUPS and PseudoLattice web utilities for computation of space group subgroups, color (magnetic) subgroups & lattice search.

BYU ISOTROPY Software Suite (<https://stokes.byu.edu/iso/isotropy.php>):

GSAS-II directly accesses capabilities in the ISOTROPY Software Suite from Brigham Young University for representational analysis and magnetism analysis.

At the request of the program authors, other programs that can be accessed within GSAS-II are not included as part of the GSAS-II distribution and must be installed separately:

Dysnomia

Computes enhanced Fourier maps with Maximum Entropy estimated extension of the reflection sphere. See <https://jp-minerals.org/dysnomia/en/>.

RMCPProfile

Provides large-box PDF & S(Q) fitting. The GSAS-II interface was originally written for use with release 6.7.7 of RMCPProfile, but updates have been made for compatible with 6.7.9 as well. RMCPProfile must be downloaded by the user from <http://rmcprofile.org/Downloads> or https://rmcprofile.pages.ornl.gov/nav_pages/download/

fullrmc

A modern software framework for large-box PDF & S(Q) fitting. Note that the GSAS-II implementation is not compatible with the last open-source version of fullrmc, but rather the version 5.0 must be used, which is distributed only as compiled versions and only for 64-bit Intel-compatible processors running Windows, Linux and MacOS. Download this as a single executable from website <https://github.com/bachiraoun/fullrmc/tree/master/standalones>. GSAS-II will offer to install this software into the binary directory when the fullrmc option is selected on the Phase/RMC tab.

PDFfit2

For small-box fitting of PDFs; see <https://github.com/diffpy/diffpy.pdf2#pdf2>. This code is no longer being updated by the authors, but is still quite useful. It is supplied within GSAS-II for Python 3.7. It is likely best to install a separate Python interpreter specifically for PDFfit2. When GSAS-II is run from a Python installation that includes the conda package manager (the usual installation practice),

the GUI will offer an option to install PDFfit2 via a separate environment when the PDFfit2 option is selected on the Phase/RMC tab.

GSASIIOBJ: DATA OBJECTS & DOCS

2.1 Summary/Contents

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

Section Contents

- GSASIIobj: Data objects & Docs
 - Summary/Contents
 - *Variable names in GSAS-II*
 - *Constraints Tree Item*
 - *Covariance Tree Item*
 - *Phase Tree Items*
 - *Rigid Body Objects*
 - *Space Group Objects*
 - *Phase Information*
 - * *Atom Records*
 - * *Drawing Atom Records*
 - * *Rigid Body Insertions*
 - *Powder Diffraction Tree Items*
 - * *CW Instrument Parameters*
 - * *TOF Instrument Parameters*
 - *Powder Reflection Data Structure*
 - *Single Crystal Tree Items*
 - *Single Crystal Reflection Data Structure*
 - *Image Data Structure*
 - *Parameter Dictionary*
 - *Texture implementation*
 - *ISODISTORT implementation*

- * *Displacive modes*
- * *Occupancy modes*
- * *Mode Computations*
- *Parameter Limits*
- *GSASIIobj Classes and routines*

2.2 Variable names in GSAS-II

Parameter are named using the following pattern, $p:h:<var>:n$, where $<var>$ is a variable name, as shown in the following table. Also, p is the phase number, h is the histogram number, and n is the atom parameter number. If a parameter does not depend on a histogram, phase or atom, h , p and/or n will be omitted, so $p:<var>:n$, $h:<var>$ and $p:h:<var>$ are all valid names.

Table 1: Naming for GSAS-II parameter names, $p:h:<var>:n$

| $<var>$ | usage |
|---------------------------------------|--|
| K (example: a) | Lattice parameter, K , from A_i and D_{jk} ; where K is one of the characters a, b or c. |
| α | Lattice parameter, α , computed from both A_i and D_{jk} . |
| β | Lattice parameter, β , computed from both A_i and D_{jk} . |
| γ | Lattice parameter, γ , computed from both A_i and D_{jk} . |
| Scale | Phase fraction (as $p:h:Scale$) or Histogram scale factor (as $h:Scale$). |
| A_I (example: A0) | Reciprocal metric tensor component I ; where I is a digit between 0 and 5. |
| L_{ol} (example: vol) | Unit cell volume; where L is one of the characters v or V. |
| dAM (example: dAx) | Refined change to atomic coordinate, M ; where M is one of the characters x, y or z. |
| AM (example: Ax) | Fractional atomic coordinate, M ; where M is one of the characters x, y or z. |
| AUiso | Atomic isotropic displacement parameter. |
| $AU_{N_0N_1}$ (example: AU11) | Atomic anisotropic displacement parameter $U_{N_0N_1}$; where N_0 is one of the characters 1, 2 or 3 and N_1 is one of the characters 1, 2 or 3. |
| Afrac | Atomic site fraction parameter. |
| Amul | Atomic site multiplicity value. |
| AM_M (example: AMx) | Atomic magnetic moment parameter, M ; where M is one of the characters x, y or z. |
| $Akappa_O$ (example: Akappa0) | Atomic orbital softness for orbital, O ; where O is one of the characters 0, - or 6. |
| ANe_P (example: ANe0) | Atomic $\langle j0 \rangle$ orbital population for orbital, P ; where P is one of the characters 0 or 1. |
| AD_{O_0,O_1O_0} (example: AD0,00) | Atomic sp. harm. coeff for orbital, 1; where O_0 is one of the characters 0, - or 6 and O_1 is one of the characters 0, - or 6 and O_0 is one of the characters 0, - or 6. |
| $AD_{O_0,-O_1O_0}$ (example: AD0,-00) | Atomic sp. harm. coeff for orbital, 1; where O_0 is one of the characters 0, - or 6 and O_1 is one of the characters 0, - or 6 and O_0 is one of the characters 0, - or 6. |
| Back J (example: Back11) | Background term $\#J$; where J is the background term number. |
| BkPkint; J (example: BkPkint;11) | Background peak $\#J$ intensity; where J is the background peak number. |
| BkPkpos; J (example: BkPkpos;11) | Background peak $\#J$ position; where J is the background peak number. |

continues on next page

Table 1 – continued from previous page

| <var> | usage |
|---|--|
| BkPksig; <i>J</i> (example: BkPksig;11) | Background peak # <i>J</i> Gaussian width; where <i>J</i> is the background peak number. |
| BkPkgam; <i>J</i> (example: BkPkgam;11) | Background peak # <i>J</i> Cauchy width; where <i>J</i> is the background peak number. |
| BF mult | Background file multiplier. |
| Bab Q (example: BabA) | Babinet solvent scattering coef. Q ; where Q is one of the characters A or U. |
| D N_0N_1 (example: D11) | Anisotropic strain coef. N_0N_1 ; where N_0 is one of the characters 1, 2 or 3 and N_1 is one of the characters 1, 2 or 3. |
| Extinction | Extinction coef. |
| MD | March-Dollase coef. |
| Mustrain; <i>J</i> (example: Mustrain;11) | Microstrain coefficient (delta Q/Q x 10**6); where <i>J</i> can be i for isotropic or equatorial and a is axial (uniaxial broadening), a number for generalized (Stephens) broadening or mx for the Gaussian/Lorentzian mixing term (LGMix). |
| Size; <i>J</i> (example: Size;11) | Crystallite size value (in microns); where <i>J</i> can be i for isotropic or equatorial, and a is axial (uniaxial broadening), a number between 0 and 5 for ellipsoidal broadening or mx for the Gaussian/Lorentzian mixing term (LGMix). |
| eA | Cubic mustrain value. |
| Ep | Primary extinction. |
| Es | Secondary type II extinction. |
| Eg | Secondary type I extinction. |
| Flack | Flack parameter. |
| TwinFr | Twin fraction. |
| Layer Disp | Layer displacement along beam. |
| Absorption | Absorption coef. |
| LayerDisp | Bragg-Brentano Layer displacement. |
| Displace R (example: DisplaceX) | Debye-Scherrer sample displacement R ; where R is one of the characters X or Y. |
| Lam | Wavelength. |
| I(L2)/I(L1) | Ka2/Ka1 intensity ratio. |
| Polariz. | Polarization correction. |
| SH/L | FCJ peak asymmetry correction. |
| s (example: U) | Gaussian instrument broadening s ; where s is one of the characters U, V or W. |
| T (example: X) | Cauchy instrument broadening T ; where T is one of the characters X, Y or Z. |
| Zero | Debye-Scherrer zero correction. |
| Shift | Bragg-Brentano sample displ. |
| SurfRoughA | Bragg-Brenano surface roughness A. |
| SurfRoughB | Bragg-Brenano surface roughness B. |
| Transparency | Bragg-Brentano sample transparency. |
| DebyeA | Debye model amplitude. |
| DebyeR | Debye model radius. |
| DebyeU | Debye model Uiso. |
| RBV J (example: RBV11) | Vector rigid body parameter. |
| RBVO U (example: RBVOa) | Vector rigid body orientation parameter U ; where U is one of the characters a, i, j or k. |
| RBVP M (example: RBVPx) | Vector rigid body M position parameter; where M is one of the characters x, y or z. |
| RBVf | Vector rigid body site fraction. |

continues on next page

Table 1 – continued from previous page

| <var> | usage |
|---|--|
| RBV _{V₀W₀W₁} (example: RBVT11) | Residue rigid body group disp. param.; where V_0 is one of the characters T, L or S and W_0 is one of the characters 1, 2, 3, A or B and W_1 is one of the characters 1, 2, 3, A or B. |
| RBVU | Residue rigid body group Uiso param. |
| RBRO _U (example: RBROa) | Residue rigid body orientation parameter U ; where U is one of the characters a, i, j or k. |
| RBRP _M (example: RBRP _x) | Residue rigid body M position parameter; where M is one of the characters x, y or z. |
| RBRT _r ;J (example: RBRT _r ;11) | Residue rigid body torsion parameter. |
| RBRf | Residue rigid body site fraction. |
| RBRV _{V₀W₀W₁} (example: RBRT11) | Residue rigid body group disp. param.; where V_0 is one of the characters T, L or S and W_0 is one of the characters 1, 2, 3, A or B and W_1 is one of the characters 1, 2, 3, A or B. |
| RBRU | Residue rigid body group Uiso param. |
| RBSAtNo | Atom number for spinning rigid body. |
| RBSO _U (example: RBSOa) | Spinning rigid body orientation parameter U ; where U is one of the characters a, i, j or k. |
| RBSP _M (example: RBSP _x) | Spinning rigid body M position parameter; where M is one of the characters x, y or z. |
| RBSShRadius | Spinning rigid body shell radius. |
| RBSShC _X (example: RBSShC1) | Spinning rigid body sph. harmonics term; where X is one of the characters 1, -, 2 or 0, 1, -, 2 or 0. |
| constr _G (example: constr10) | Generated degree of freedom from constraint; where G is one or more digits (0, 1,... 9). |
| nv-(.) | New variable assignment with name 1. |
| mV _H (example: mV0) | Modulation vector component H ; where H is the digits 0, 1, or 2. |
| Fsin | Sin site fraction modulation. |
| Fcos | Cos site fraction modulation. |
| Fzero | Crenel function offset. |
| Fwid | Crenel function width. |
| Tmin | ZigZag/Block min location. |
| Tmax | ZigZag/Block max location. |
| T _{max} (example: X _{max}) | ZigZag/Block max value for T ; where T is one of the characters X, Y or Z. |
| T _{sin} (example: X _{sin}) | Sin position wave for T ; where T is one of the characters X, Y or Z. |
| T _{cos} (example: X _{cos}) | Cos position wave for T ; where T is one of the characters X, Y or Z. |
| U _{N₀N₁} sin (example: U11sin) | Sin thermal wave for U _{N₀N₁} ; where N_0 is one of the characters 1, 2 or 3 and N_1 is one of the characters 1, 2 or 3. |
| U _{N₀N₁} cos (example: U11cos) | Cos thermal wave for U _{N₀N₁} ; where N_0 is one of the characters 1, 2 or 3 and N_1 is one of the characters 1, 2 or 3. |
| M _T sin (example: M _X sin) | Sin mag. moment wave for T ; where T is one of the characters X, Y or Z. |
| M _T cos (example: M _X cos) | Cos mag. moment wave for T ; where T is one of the characters X, Y or Z. |
| PDFpos | PDF peak position. |
| PDFmag | PDF peak magnitude. |
| PDFsig | PDF peak std. dev. |
| Aspect ratio | Particle aspect ratio. |
| Length | Cylinder length. |
| Diameter | Cylinder/disk diameter. |
| Thickness | Disk thickness. |
| Shell thickness | Multiplier to get inner(<1) or outer(>1) sphere radius. |
| Dist | Interparticle distance. |

continues on next page

Table 1 – continued from previous page

| <var> | usage |
|---------------------------------|---|
| VolFr | Dense scatterer volume fraction. |
| epis | Sticky sphere epsilon. |
| Sticky | Stickyness. |
| Depth | Well depth. |
| Width | Well width. |
| Volume | Particle volume. |
| Radius | Sphere/cylinder/disk radius. |
| Mean | Particle mean radius. |
| StdDev | Standard deviation in Mean. |
| G | Guinier prefactor. |
| Rg | Guinier radius of gyration. |
| B | Porod prefactor. |
| P | Porod power. |
| Cutoff | Porod cutoff. |
| PkInt | Bragg peak intensity. |
| PkPos | Bragg peak position. |
| PkSig | Bragg peak sigma. |
| PkGam | Bragg peak gamma. |
| eY_0Y_1 (example: e11) | strain tensor eY_0Y_1 ; where Y_0 is one of the characters 1 or 2 and Y_1 is one of the characters 1 or 2. |
| Dcalc | Calc. d-spacing. |
| Back | background parameter. |
| pos | peak position. |
| int | peak intensity. |
| WgtFrac | phase weight fraction. |
| alpha | TOF profile term. |
| alpha- P (example: alpha-0) | Pink profile term; where P is one of the characters 0 or 1. |
| beta- Z (example: beta-0) | TOF/Pink profile term; where Z is one of the characters 0, 1 or q. |
| sig- a (example: sig-0) | TOF profile term; where a is one of the characters 0, 1, 2 or q. |
| dif b (example: difA) | TOF to d-space calibration; where b is one of the characters A, B or C. |
| CG_0G_1 (example: C10,10) | spherical harmonics preferred orientation coef.; where G_0 is one or more digits (0, 1,... 9) and G_1 is one or more digits (0, 1,... 9). |
| Pressure | Pressure level for measurement in MPa. |
| Temperature | T value for measurement, K. |
| FreePrm N (example: FreePrm1) | User defined measurement parameter N ; where N is one of the characters 1, 2 or 3. |
| Gonio. radius | Distance from sample to detector, mm. |

2.3 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 `GSASIImapvars.ProcessConstraints()` to the form used in `GSASIImapvars`

The keys in the Constraints dict are:

| key | explanation |
|--------|--|
| Hist | This specifies a list of constraints on histogram-related parameters, which will be of form <code>:h:<var>:n</code> . |
| HAP | This specifies a list of constraints on parameters that are defined for every histogram in each phase and are of form <code>p:h:<var>:n</code> . |
| Phase | This specifies a list of constraints on phase parameters, which will be of form <code>p:<var>:n</code> . |
| Global | This specifies a list of constraints on parameters that are not tied to a histogram or phase and are of form <code>::<var>:n</code> . |

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. (Note that in very old .gpx files this might be a str with a 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, $m_1 \times var_1 + m_2 \times var_2 + \dots = c$
 - '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 $newvar = m_1 \times var_1 + m_2 \times var_2 + \dots$

2.4 Covariance Tree Item

The Covariance tree item has results from the last least-squares run. They are stored in a dict with these keys:

| key | sub-key | explanation |
|-------------|---------|---|
| newCellDict | | (dict) ith lattice parameters computed by GSASIIstrMath.GetNewCellParms() |
| title | | (str) Name of gpx file(?) |
| variables | | (list) Values for all N refined variables (list of float values, length N, ordered to match varyList) |
| sig | | (list) Uncertainty values for all N refined variables (list of float values, length N, ordered to match varyList) |
| varyList | | (list of str values, length N) List of directly refined variables |
| newAtomDict | | (dict) atom position values computed in GSASIIstrMath.ApplyXYZshifts() |
| Rvals | | (dict) R-factors, GOF, Marquardt value for last refinement cycle |
| | Nobs | (int) Number of observed data points |
| | Rwp | (float) overall weighted profile R-factor (%) |
| | chisq | (float) $\sum w * (I_{obs} - I_{calc})^2$ for all data. Note: this is not the reduced χ^2 . |
| | lamMax | (float) Marquardt value applied to Hessian diagonal |
| | GOF | (float) The goodness-of-fit, aka square root of the reduced chi squared. |
| covMatrix | | (np.array) The (NxN) covVariance matrix |

2.5 Phase Tree Items

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

| key | sub-key | explanation |
|---------|----------------|---|
| General | | (dict) Overall information for the phase |
| | 3Dproj | (list of str) projections for 3D pole distribution plots |
| | AngleRadii | (list of floats) Default radius for each atom used to compute interatomic angles |
| | AtomMass | (list of floats) Masses for atoms |
| | AtomPtrs | (list of int) four locations (cx,ct,cs & cu) to use to pull info from the atom records |
| | AtomTypes | (l)list of str) Atom types |
| | BondRadii | (list of floats) Default radius for each atom used to compute interatomic distances |
| | Cell | Unit cell parameters & ref. flag (list with 8 items. All but first item are float.) 0: cell refinement flag (True/False), 1-3: a, b, c, (Å) 4-6: alpha, beta & gamma, (degrees) 7: volume (Å ³) |
| | Color | (list of (r,b,g) triplets) Colors for atoms |
| | Compare | (dict) Polygon comparison parameters |
| | Data plot type | (str) data plot type ('Mustrain', 'Size' or 'Preferred orientation') for powder data |

continues on next page

Table 2 – continued from previous page

| key | sub-key | explanation |
|---------|---------------|--|
| | DisAglCtrls | (dDict) with distance/angle search controls, which has keys ‘Name’, ‘Atom-Types’, ‘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)]. |
| | F000X | (float) x-ray F(000) intensity |
| | F000N | (float) neutron F(000) intensity |
| | Flip | (dict) Charge flip controls |
| | HydIds | (dict) geometrically generated hydrogen atoms |
| | Isotope | (dict) Isotopes for each atom type |
| | Isotopes | (dict) Scattering lengths for each isotope combination for each element in phase |
| | MCSA controls | (dict) Monte Carlo-Simulated Annealing controls |
| | Map | (dict) Map parameters |
| | Mass | (float) Mass of unit cell contents in g/mol |
| | Modulated | (bool) True if phase modulated |
| | Mydir | (str) Directory of current .gpx file |
| | Name | (str) Phase name |
| | NoAtoms | (dict) Number of atoms per unit cell of each type |
| | POhkl | (list) March-Dollase preferred orientation direction |
| | Pawley dmin | (float) maximum Q (as d-space) to use for Pawley extraction |
| | Pawley dmax | (float) minimum Q (as d-space) to use for Pawley extraction |
| | Pawley neg wt | (float) Restraint value for negative Pawley intensities |
| | SGData | (object) Space group details as a <i>space group (SGData)</i> object, as defined in <code>GSASIIspc.SpcGroup()</code> . |
| | SH Texture | (dict) Spherical harmonic preferred orientation parameters |
| | Super | (int) dimension of super group (0,1 only) |
| | Type | (str) phase type (e.g. ‘nuclear’) |
| | Z | (dict) Atomic numbers for each atom type |
| | doDysnomia | (bool) flag for max ent map modification via Dysnomia |
| | doPawley | (bool) Flag for Pawley intensity extraction |
| | vdWRadii | (dict) Van der Waals radii for each atom type |
| ranId | | (int) unique random number Id for phase |
| pId | | (int) Phase Id number for current project. |
| Atoms | | (list of lists) 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 <i>Atom Records</i> description. |
| Drawing | | (dict) Display parameters |
| | Atoms | (list of lists) with an entry for each atom that is drawn |
| | Plane | (list) Controls for contour density plane display |
| | Quaternion | (4 element np.array) Viewing quaternion |
| | Zclip | (float) clipping distance in Å |
| | Zstep | (float) Step to de/increase Z-clip |
| | atomPtrs | (list) positions of x, type, site sym, ADP flag in Draw Atoms |
| | backColor | (list) background for plot as and R,G,B triplet (default = [0, 0, 0], black). |
| | ballScale | (float) Radius of spheres in ball-and-stick display |
| | bondList | (dict) Bonds |
| | bondRadius | (float) Radius of binds in Å |
| | cameraPos | (float) Viewing position in Å for plot |
| | contourLevel | (float) map contour level in $e/\text{Å}^3$ |
| | contourMax | (float) map contour maximum |

continues on next page

Table 2 – continued from previous page

| key | sub-key | explanation |
|------------|-----------------|--|
| | depthFog | (bool) True if use depthFog on plot - set currently as False |
| | ellipseProb | (float) Probability limit for display of thermal ellipsoids in % . |
| | magMult | (float) multiplier for magnetic moment arrows |
| | mapSize | (float) x & y dimensions of contourmap (fixed internally) |
| | modelView | (4,4 array) from openGL drawing transformation matrix |
| | oldxy | (list with two floats) previous view point |
| | radiusFactor | (float) 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. |
| | selectedAtoms | (list of int values) List of selected atoms |
| | showABC | (bool) Flag to show view point triplet. True=show. |
| | showHydrogen | (bool) Flag to control plotting of H atoms. |
| | showRigidBodies | (bool) Flag to highlight rigid body placement |
| | showSlice | (bool) flag to show contour map |
| | sizeH | (float) Size ratio for H atoms |
| | unitCellBox | (bool) Flag to control display of the unit cell. |
| | vdwScale | (float) Multiplier of van der Waals radius for display of vdW spheres. |
| | viewDir | (np.array with three floats) cartesian viewing direction |
| | viewPoint | (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 & current atom number viewed (may be [0,0]) |
| ISODISTORT | | (dict) contains controls for running ISODISTORT and results from it |
| | ISOmethod | (int) ISODISTORT method (currently 1 or 4; 2 & 3 not implemented in GSAS-II) |
| | ParentCIF | (str) parent cif file name for ISODISTORT method 4 |
| | ChildCIF | (str) child cif file name for ISODISTORT method 4 |
| | SGselect | (dict) selection list for lattice types in radio result from ISODISTORT method 1 |
| | selection | (int) chosen selection from radio |
| | radio | (list) results from ISODISTORT method 1 |
| | ChildMatrix | (3x3 array) transformation matrix for method 3 (not currently used) |
| | ChildSprGp | (str) child space group for method 3 (not currently used) |
| | ChildCell | (str) cell ordering for nonstandard orthorhombic ChildSprGrp in method 3 (not currently used) |
| | G2ModeList | (list) ISODISTORT mode names |
| | modeDispl | (list) distortion mode values; refinable parameters |
| | ISOmodeDispl | (list) distortion mode values as determined in method 4 by ISODISTORT |
| | NormList | (list) ISODISTORT normalization values; to convert mode value to fractional coordinate displacement |
| | G2parentCoords | (list) full set of parent structure coordinates transformed to child structure; starting basis for mode displacements |
| | G2VarList | (list) |
| | IsoVarList | (list) |
| | G2coordOffset | (list) only adjustable set of parent structure coordinates |
| | G2OccVarList | (list) |
| | Var2ModeMatrix | (array) atom variable to distortion mode transformation |
| | Mode2VarMatrix | (array) distortion mode to atom variable transformation |
| | rundata | (dict) saved input information for use by ISODISTORT method 1 |
| RBModels | | Rigid body assignments (note Rigid body definitions are stored in their own main top-level tree entry.) |
| RMC | | (dict) RMCProfile, PDFfit & fullrmc controls |

continues on next page

Table 2 – continued from previous page

| key | sub-key | explanation |
|------------|------------|---|
| Pawley ref | | (list) Pawley reflections |
| Histograms | | (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) |
| | Babinet | (dict) For protein crystallography. Dictionary with two entries, 'BabA', 'BabU' |
| | Extinction | (list of float, bool) Extinction parameter |
| | Flack | (list of [float, bool]) Flack parameter & refine flag |
| | HStrain | (list of two lists) Hydrostatic strain. 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) |
| | Histogram | (str) The name of the associated histogram |
| | Layer Disp | (list of [float, bool]) Layer displacement in beam direction & refine flag |
| | LeBail | (bool) Flag for LeBail extraction |
| | Mustrain | (list) Microstrain parameters, in order: <ol style="list-style-type: none"> 0. Type, one of u'isotropic', u'uniaxial', u'generalized' 1. Isotropic/uniaxial parameters - list of 3 floats 2. Refinement flags - list of 3 bools 3. Microstrain axis - list of 3 ints, [h, k, l] 4. Generalized mustrain parameters - list of 2-6 floats, depending on space group 5. Generalized refinement flags - list of bools, corresponding to the parameters of (4) |
| | Pref.Ori. | (list) Preferred Orientation. List of eight parameters. Items marked SH are only used for Spherical Harmonics. <ol style="list-style-type: none"> 0. (str) Type, 'MD' for March-Dollase or 'SH' for Spherical Harmonics 1. (float) Value 2. (bool) Refinement flag 3. (list) Preferred direction, list of ints, [h, k, l] 4. (int) SH - number of terms 5. (dict) SH - 6. (list) SH 7. (float) SH |
| | Scale | (list of [float, bool]) Phase fraction & refine flag |
| | Size | List of crystallite size parameters, in order: <ol style="list-style-type: none"> 0. (str) Type, one of u'isotropic', u'uniaxial', u'ellipsoidal' 1. (list) Isotropic/uniaxial parameters - list of 3 floats 2. (list) Refinement flags - list of 3 bools 3. (list) Size axis - list of 3 ints, [h, k, l] 4. (list) Ellipsoidal size parameters - list of 6 floats 5. (list) Ellipsoidal refinement flags - list of bools, corresponding to the parameters of (4) |
| | Use | (bool) True if this histogram is to be used in refinement |
| MCSA | | (dict) Monte-Carlo simulated annealing parameters |

2.6 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.

| key | sub-key | explanation |
|---------|----------|--|
| Vector | RBId | (dict of dict) vector rigid bodies |
| | AtInfo | (dict) Drad, Color: atom drawing radius & color for each atom type |
| | RBname | (str) Name assigned by user to rigid body |
| | VectMag | (list) vector magnitudes in Å |
| | rbXYZ | (list of 3 float Cartesian coordinates for Vector rigid body) |
| | rbRef | (list of 3 int & 1 bool) 3 assigned reference atom nos. in rigid body for origin definition, use center of atoms flag |
| | VectRef | (list of bool refinement flags for VectMag values) |
| | rbTypes | (list of str) Atom types for each atom in rigid body |
| | rbVect | (list of lists) Cartesian vectors for each translation used to build rigid body |
| | useCount | (int) Number of times rigid body is used in any structure |
| Residue | RBId | (dict of dict) residue rigid bodies |
| | AtInfo | (dict) Drad, Color: atom drawing radius & color for each atom type |
| | RBname | (str) Name assigned by user to rigid body |
| | rbXYZ | (list of 3 float) Cartesian coordinates for Residue rigid body |
| | rbTypes | (list of str) Atom types for each atom in rigid body |
| | atNames | (list of str) Names of each atom in rigid body (e.g. C1,N2...) |
| | rbRef | (list of 3 int & 1 bool) 3 assigned reference atom nos. in rigid body for origin definition, use center of atoms flag |
| | rbSeq | (list) Orig,Piv,angle,Riding : definition of internal rigid body torsion; origin atom (int), pivot atom (int), torsion angle (float), riding atoms (list of int) |
| | SelSeq | (int,int) used by SeqSizer to identify objects |
| RBIds | useCount | (int)Number of times rigid body is used in any structure |
| | | (dict) unique Ids generated upon creation of each rigid body |
| | Vector | (list) Ids for each Vector rigid body |
| | Residue | (list) Ids for each Residue rigid body |

2.7 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. Magnetic ones are marked "mag"

| key | explanation |
|------------|--|
| BNSlattsym | mag - (str) BNS magnetic space group symbol and centering vector |
| GenFlg | mag - (list) symmetry generators indices |
| GenSym | mag - (list) names for each generator |
| MagMom | mag - (list) "time reversals" for each magnetic operator |
| MagPtGp | mag - (str) Magnetic point group symbol |
| MagSpGrp | mag - (str) Magnetic space group symbol |
| OprNames | mag - (list) names for each space group operation |
| SGCen | (np.array) Symmetry cell centering vectors. A (n,3) np.array of centers. Will always have at least one row: <code>np.array([[0, 0, 0]])</code> |
| SGFixed | (bool) Only True if phase imported from a magnetic cif file then the space group can not be changed by the user because operator set from cif may be nonstandard |
| SGGen | (list) generators |
| SGGray | (bool) True if space group is a gray group (incommensurate magnetic structures) |
| SGInv | (bool) True if centrosymmetric, False if not |
| SSLatt | (str) Lattice centering type. Will be one of P, A, B, C, I, F, R |
| SLaue | (str) 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 |
| SGOps | (list) symmetry operations as a list of form <code>[[M1, T1], [M2, T2], ...]</code> 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 $X' = M_n * X + T_n$ |
| SGPolax | (str) 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 ?). |
| SGPtGrp | (str) Point group of the space group |
| SGUniq | unique axis if monoclinic. Will be a, b, or c for monoclinic space groups. Will be blank for non-monoclinic. |
| SGSpin | mag - (list) of spin flip operators (+1 or -1) for the space group operations |
| SGSys | (str) symmetry unit cell: type one of 'triclinic', 'monoclinic', 'orthorhombic', 'tetragonal', 'rhombohedral', 'trigonal', 'hexagonal', 'cubic' |
| SSGK1 | (list) Superspace multipliers |
| SpGrp | (str) space group symbol |
| SpnFlp | mag - (list) Magnetic spin flips for every magnetic space group operator |

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

| key | explanation |
|---------|--|
| SSGCen | (list) 4D cell centering vectors [0,0,0,0] at least |
| SSGK1 | (list) Superspace multipliers |
| SSGOps | (list) 4D symmetry operations as [M,T] so that $M*x+T = x'$ |
| SSpGrp | (str) superspace group symbol extension to space group symbol, accidental spaces removed |
| modQ | (list) modulation/propagation vector |
| modSymb | (list of str) Modulation symbols |

2.8 Phase Information

Phase information is placed in one of the following keys:

| key | explanation |
|------------|---|
| General | Overall information about a phase |
| Histograms | Information about each histogram linked to the current phase as well as parameters that are defined for each histogram and phase (such as sample peak widths and preferred orientation parameters). |
| Atoms | Contains a list of atoms, as described in the <i>Atom Records</i> description. |
| Drawing | Parameters that determine how the phase is displayed, including a list of atoms to be included, as described in the <i>Drawing Atom Records</i> description |
| MCSA | Monte-Carlo simulated annealing parameters |
| pId | The index of each phase in the project, numbered starting at 0 |
| ranId | An int value with a unique value for each phase |
| RBModels | A list of dicts with parameters for each rigid body inserted into the current phase, as defined in the <i>Rigid Body Insertions</i> . Note that the rigid bodies are defined as <i>Rigid Body Objects</i> |
| RMC | PDF modeling parameters |
| Pawley ref | Pawley refinement parameters |

2.8.1 Atom Records

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

| location | explanation |
|----------------|--|
| ct-4 | mm - (str) residue number |
| ct-3 | mm - (str) residue name (e.g. ALA) |
| ct-2 | mm - (str) chain label |
| ct-1 | (str) atom label |
| ct | (str) atom type |
| ct+1 | (str) refinement flags; combination of 'F', 'X', 'U', 'M' |
| cx,cx+1,cx+2 | (3 floats) the x,y and z coordinates |
| cx+3 | (float) site occupancy |
| cx+4,cx+5,cx+6 | mg - (list) atom magnetic moment along a,b,c in Bohr magnetons |
| cs | (str) site symmetry |
| cs+1 | (int) site multiplicity |
| cia | (str) ADP flag: Isotropic ('I') or Anisotropic ('A') |
| cia+1 | (float) Uiso |
| cia+2...cia+7 | (6 floats) U11, U22, U33, U12, U13, U23 |
| atom[cia+8] | (int) unique atom identifier |

2.8.2 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', and those for magnetic structures are marked 'mg'

| location | explanation |
|----------------|--|
| ct-4 | mm - (str) residue number |
| ct-3 | mm - (str) residue name (e.g. ALA) |
| ct-2 | mm - (str) chain label |
| ct-1 | (str) atom label |
| ct | (str) atom type |
| cx,cx+1,cx+2 | (3 floats) the x,y and z coordinates |
| cx+3,cx+4,cx+5 | mg - (3 floats) atom magnetic moment along a,b,c in Bohr magnetons |
| cs-1 | (str) Sym Op symbol; sym. op number + unit cell id (e.g. '1,0,-1') |
| cs | (str) atom drawing style; e.g. 'balls & sticks' |
| cs+1 | (str) atom label style (e.g. 'name') |
| cs+2 | (int) atom color (RGB triplet) |
| cs+3 | (str) ADP flag: Isotropic ('I') or Anisotropic ('A') |
| cs+4 | (float) Uiso |
| cs+5...cs+11 | (6 floats) U11, U22, U33, U12, U13, U23 |
| ci | (int) unique atom identifier; matches source atom Id in Atom Records |

2.8.3 Rigid Body Insertions

If `phasedict` points to the phase information in the data tree, then rigid body information is contained in list(s) in `phasedict['RBModels']['Residue']` and/or `phasedict['RBModels']['Vector']` for each rigid body inserted into the current phase.

| key | explanation |
|---------------|---|
| fixOrig | Should the origin be fixed (when editing, not the refinement flag) |
| Ids | Ids for assignment of atoms in the rigid body |
| numChain | Chain number for macromolecular fits |
| Orient | Orientation of the RB as a quaternion and a refinement flag ('', 'A' or 'AV') |
| OrientVec | Orientation of the RB expressed as a vector and azimuthal rotation angle |
| Orig | Origin of the RB in fractional coordinates and refinement flag (bool) |
| RBId | References the unique ID of a rigid body in the <i>Rigid Body Objects</i> |
| RBname | The name for the rigid body (str) |
| AtomFrac | The atom fractions for the rigid body |
| ThermalMotion | The thermal motion description for the rigid body, which includes a choice for the model and can include TLS parameters or an overall Uiso value. |
| Torsions | Defines the torsion angle and refinement flag for each torsion defined in the <i>Rigid Body Object</i> |

2.9 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.

| key | sub-key | explanation |
|-----------------------|---|--|
| Comments | | (list of str) Text strings extracted from the original powder data header. These cannot be changed by the user; it may be empty. |
| Limits | | (list) 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. |
| Reflection Lists | | (dict of dicts) 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. |
| Instrument Parameters | | (dict) The instrument parameters uses different dicts for the constant wavelength (CW) and time-of-flight (TOF) cases. See below for the descriptions of each. |
| wtFactor | | (float) A weighting factor to increase or decrease the leverage of data in the histogram . 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). |
| Sample Parameters | | (dict) 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. |
| | Scale | The histogram scale factor (refinable) |
| | Absorption | The sample absorption coefficient as μr where r is the radius (refinable). Only valid for Debye-Scherrer geometry. |
| | SurfaceRoughA | Surface roughness parameter A as defined by Surotti, <i>J. Appl. Cryst</i> , 5 , 325-331, 1972. (refinable - only valid for Bragg-Brentano geometry) |
| | SurfaceRoughB | Surface roughness parameter B (refinable - only valid for Bragg-Brentano geometry) |
| | DisplaceX, DisplaceY | Sample displacement from goniometer center where Y is along the beam direction and X is perpendicular. Units are μm (refinable). |
| | Phi, Chi, Omega | Goniometer sample setting angles, in degrees. |
| | Gonio. radius | Radius of the diffractometer in mm |
| | InstrName | (str) A name for the instrument, used in preparing a CIF . |
| | Force, Temperature, Humidity, Pressure, Voltage | Variables that describe how the measurement was performed. Not used directly in any computations. |
| | ranId | (int) The random-number Id for the histogram (same value as where top-level key is ranId) |
| | Type | (str) Type of diffraction data, may be 'Debye-Scherrer' or 'Bragg-Brentano' . |
| hId | | (int) The number assigned to the histogram when the project is loaded or edited (can change) |
| ranId | | (int) A random number id for the histogram that does not change |
| Background | | (list) The background is stored as a list with where the first item in the list is list and the second item is a dict. The list contains the background function and its coefficients; the dict contains Debye diffuse terms and background peaks. (TODO: this needs to be expanded.) |
| Data | | (list) The data consist of a list of 6 np.arrays containing in order: 0. the x-positions (two-theta in degrees), 1. the intensity values (Yobs), 2. the weights for each Yobs value, 3. the computed intensity values (Ycalc) 4. the background values 5. Yobs-Ycalc |

2.9.1 CW Instrument Parameters

Instrument Parameters are placed in a list of two dicts, where the keys in the first dict are listed below. Note that the dict contents are different for constant wavelength (CW) vs. time-of-flight (TOF) histograms. 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 not refined are noted as [*]

| key | sub-key | explanation |
|--------------------------|-------------|--|
| Instrument Parameters[0] | Type [*] | (str) Histogram type: * 'PXC' for constant wavelength x-ray * 'PNC' for constant wavelength neutron |
| | Bank [*] | (int) Data set number in a multidata file (usually 1) |
| | Lam | (float) Specifies a wavelength in Å |
| | Lam1 [*] | (float) Specifies the primary wavelength in Å, used in place of Lam when an α_1, α_2 source is used. |
| | Lam2 [*] | (float) Specifies the secondary wavelength in Å, used with Lam1 |
| | I(L2)/I(L1) | (float) Ratio of Lam2 to Lam1, used with Lam1 |
| | Zero | (float) Two-theta zero correction in <i>degrees</i> |
| | Azimuth [*] | (float) Azimuthal setting angle for data recorded with differing setting angles |
| | U, V, W | (float) Cagliotti profile coefficients for Gaussian instrumental broadening, where the FWHM goes as $U \tan^2 \theta + V \tan \theta + W$ |
| | X, Y, Z | (float) Cauchy (Lorentzian) instrumental broadening coefficients |
| | SH/L | (float) Variant of the Finger-Cox-Jephcoat asymmetric peak broadening ratio. Note that this is the sum of S/L and H/L where S is sample height, H is the slit height and L is the goniometer diameter. |
| | Polariz. | (float) Polarization coefficient. |
| Instrument Parameters[1] | | (empty dict) |

2.9.2 TOF Instrument Parameters

Instrument Parameters are also placed in a list of two dicts, where the keys in each dict listed below, but here for time-of-flight (TOF) histograms. 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 not refined are noted as [*]

| key | sub-key | explanation |
|--------------------------|-------------------------|---|
| Instrument Parameters[0] | Type [*] | (str) Histogram type: * 'PNT' for time of flight neutron |
| | Bank | (int) Data set number in a multidata file |
| | 2-theta [*] | (float) Nominal scattering angle for the detector |
| | fltPath [*] | (float) Total flight path source-sample-detector |
| | Azimuth [*] | (float) Azimuth angle for detector right hand rotation from horizontal away from source |
| | difC,difA, difB | (float) Diffractometer constants for conversion of d-spacing to TOF in microseconds |
| | Zero | (float) Zero point offset (microseconds) |
| | alpha | (float) Exponential rise profile coefficients |
| | beta-0 beta-1 beta-q | (float) Exponential decay profile coefficients |
| | sig-0 sig-1 sig-2 sig-q | (float) Gaussian profile coefficients |
| Instrument Parameters[1] | X,Y,Z | (float) Lorentzian profile coefficients |
| | Pdabc | (list of 4 float lists) Originally created for use in gsas as optional tables of d, alp, bet, d-true; for a reflection alpha & beta are obtained via interpolation from the d-spacing and these tables. The d-true column is apparently unused. |

2.10 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.

| index | explanation |
|-------|---|
| 0,1,2 | h,k,l (float) |
| 3 | (int) multiplicity |
| 4 | (float) d-space, Å |
| 5 | (float) pos, two-theta |
| 6 | (float) sig, Gaussian width |
| 7 | (float) gam, Lorentzian width |
| 8 | (float) F_{obs}^2 |
| 9 | (float) F_{calc}^2 |
| 10 | (float) reflection phase, in degrees |
| 11 | (float) intensity correction for reflection, this times F_{obs}^2 or F_{calc}^2 gives Iobs or Icalc |
| 12 | (float) Preferred orientation correction |
| 13 | (float) Transmission (absorption correction) |
| 14 | (float) Extinction correction |

2.11 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 `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.

| key | sub-key | explanation |
|-----------------------|-----------|--|
| Data | | (dict) that contains the reflection table, as described in the Single Crystal Reflections description. |
| Instrument Parameters | | (list) containing two dicts where the possible keys in each dict are listed below. The value for most items is a list containing two values: the initial value, the current value. The first and second values are floats unless otherwise noted. |
| | Lam | (two floats) Specifies a wavelength in Å |
| | Type | (two str values) Histogram type : * ‘SXC’ for constant wavelength x-ray * ‘SNC’ for constant wavelength neutron * ‘SNT’ for time of flight neutron * ‘SEC’ for constant wavelength electrons (e.g. micro-ED) |
| | InstrName | (str) A name for the instrument, used in preparing a CIF |
| wtFactor | | (float) A weighting factor to increase or decrease the leverage of data in the histogram. 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). |
| hId | | (int) The number assigned to the histogram when the project is loaded or edited (can change) |
| ranId | | (int) A random number id for the histogram that does not change |

2.12 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 for non-superspace phases.

| index | 3+1 index | explanation |
|-------|-----------|---|
| 0,1,2 | 0,1,2 | reflection indices, h,k,l |
| | 3 | 3+1 superspace index, m |
| 3 | 4 | flag (0 absent, 1 observed) |
| 4 | 5 | d-space, Å |
| 5 | 6 | F_{obs}^2 |
| 6 | 7 | $\sigma(F_{obs}^2)$ |
| 7 | 8 | F_{calc}^2 |
| 8 | 9 | $F_{obs}^2(T)$ |
| 9 | 10 | $F_{calc}^2(T)$ |
| 10 | 11 | reflection phase, in degrees |
| 11 | 12 | intensity correction for reflection, this times F_{obs}^2 or F_{calc}^2 gives Iobs or Icalc |

Notes:

- The annotation “(T)” in the second set of $F^2(T)$ values stands for “true,” where the values are on an absolute scale through application of the scale factor.
- The left-most column gives the entry index for three dimensional spacegroups, the column to the right of that has the index for 3+1 superspace phases, where there are four reflection indices h, k, l, m.

2.13 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.

| key | sub-key | explanation |
|----------------|------------------|--|
| Comments | | (list of str) Text strings extracted from the original image data header or a metafile. These cannot be changed by the user; it may be empty. |
| Image Controls | azmthOff | (float) The offset to be applied to an azimuthal value. Accommodates detector orientations other than with the detector X-axis horizontal. |
| | background image | (list:str,float) 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 < 0 for subtraction. |
| | calibrant | (str) The material used for determining the position/orientation of the image. The data is obtained from <code>ImageCalibrants()</code> and <code>UserCalibrants.py</code> (supplied by user). |
| | calibdmin | (float) The minimum d-spacing used during the last calibration run. |
| | calibskip | (int) The number of expected diffraction lines skipped during the last calibration run. |
| | center | (list:floats) 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. |
| | centerAzm | (bool) 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. |
| | color | (str) The name of the colormap used to display the image. Default = ‘Paired’. |
| | cutoff | (float) The minimum value of I/I _b for a point selected in a diffraction ring for calibration calculations. See <code>pixLimit</code> for details as how point is found. |
| | DetDepth | (float) Coefficient for penetration correction to distance; accounts for diffraction ring offset at higher angles. Optionally determined by calibration. |
| | DetDepthRef | (bool) If True then refine DetDepth during calibration/recalibration calculation. |
| | distance | (float) The distance (mm) from sample to detector plane. |
| | ellipses | (list:lists) 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 & 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. |

continues on next page

Table 3 – continued from previous page

| key | sub-key | explanation |
|-------|---------------|--|
| | edgemin | (float) Not used; parameter in EdgeFinder code. |
| | fullIntegrate | (bool) If True then integrate over full 360 deg azimuthal range. |
| | GonioAngles | (list:floats) The ‘Omega’, ‘Chi’, ‘Phi’ goniometer angles used for this image. Required for texture calculations. |
| | invert_x | (bool) If True display the image with the x-axis inverted. |
| | invert_y | (bool) If True display the image with the y-axis inverted. |
| | IOth | (list:floats) The minimum and maximum 2-theta values to be used for integration. |
| | LRazimuth | (list:floats) The minimum and maximum azimuth values to be used for integration. |
| | Oblique | (list:float,bool) If True apply a detector absorption correction using the value to the intensities obtained during integration. |
| | outAzimuths | (int) The number of azimuth pie slices. |
| | outChannels | (int) The number of 2-theta steps. |
| | pixelSize | (list:ints) The X,Y dimensions (microns) of each pixel. |
| | pixLimit | (int) A box in the image with 2*pixLimit+1 edges is searched to find the maximum. This value (I) along with the minimum (Ib) in the box is reported by <code>GSASIIimage.ImageLocalMax()</code> and subject to cutoff in <code>GSASIIimage.makeRing()</code> . Locations are used to construct rings of points for calibration calculations. |
| | PolaVal | (list:float,bool) If type=‘SASD’ and if True, apply polarization correction to intensities from integration using value. |
| | rings | (list:lists) Each entry is [X,Y,dsp] where X & Y are lists of x,y coordinates around a diffraction ring with the same d-spacing (dsp) |
| | ring | (list) The x,y coordinates of the >5 points on an inner ring selected by the user, |
| | Range | (list) The minimum & maximum values of the image |
| | rotation | (float) The angle between the x-axis and the vector about which the detector is tilted. Constrained to -180 to 180 deg. |
| | SampleShape | (str) Currently only ‘Cylinder’. Sample shape for Debye-Scherrer experiments; used for absorption calculations. |
| | SampleAbs | (list: float,bool) Value of absorption coefficient for Debye-Scherrer experiments, flag if True to cause correction to be applied. |
| | setDefault | (bool) If True the use the image controls values for all new images to be read. (might be removed) |
| | setRings | (bool) If True then display all the selected x,y ring positions (vida supra rings) used in the calibration. |
| | showLines | (bool) If True then isplay the integration limits to be used. |
| | size | (list:int) The number of pixels on the image x & y axes |
| | type | (str) One of ‘PWDR’, ‘SASD’ or ‘REFL’ for powder, small angle or reflectometry data, respectively. |
| | tilt | (float) The angle the detector normal makes with the incident beam; range -90 to 90. |
| | wavelength | (float) The radiation wavelength (Å) as entered by the user (or someday obtained from the image header). |
| Masks | Arcs | (list: lists) Each entry [2-theta,[azimuth[0],azimuth[1]],thickness] describes an arc mask to be excluded from integration |
| | Frames | (list:lists) 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. |

continues on next page

Table 3 – continued from previous page

| key | sub-key | explanation |
|---------------|------------|---|
| | Points | (list:lists) Each entry [x,y,radius] (mm) describes an excluded spot on the image to be excluded from integration. |
| | Polygons | (list:lists) Each entry is a list of 3+ [x,y] points (mm) that describe a polygon on the image to be excluded from integration. |
| | Rings | (list: lists) Each entry [2-theta,thickness] describes a ring mask to be excluded from integration. |
| | Thresholds | (list:(tuple,list)) [(Imin,Imax),[Imin,Imax]] This gives lower and upper limits for points on the image to be included in integrsation. The tuple is the image intensity limits and the list are those set by the user. |
| | SpotMask | (dict: int & array) 'esdMul'(int) number of standard deviations above mean ring intensity to mask 'spotMask' (bool array) the spot mask for every pixel in image |
| Stress/Strain | Sample phi | (float) Sample rotation about vertical axis. |
| | Sample z | (float) Sample translation from the calibration sample position (for Sample phi = 0) These will be restricted by space group symmetry; result of strain fit refinement. |
| | Type | (str) 'True' or 'Conventional': The strain model used for the calculation. |
| | d-zero | (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 & 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 & azimuth from fit. |

2.14 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).

2.15 Texture implementation

There are two different places where texture can be treated in GSAS-II. One is for mitigating the effects of texture in a structural refinement. The other is for texture characterization.

For reducing the effect of texture in a structural refinement there are entries labeled preferred orientation in each phase's data tab. Two different approaches can be used for this, the March-Dollase model and spherical harmonics.

For the March-Dollase model, one axis in reciprocal space is designated as unique (defaulting to the 001 axis) and reflections are corrected according to the angle they make with this axis depending on the March-Dollase ratio. (If unity, no correction is made). The ratio can be greater than one or less than one depending on if crystallites oriented along the designated axis are overrepresented or underrepresented. For most crystal systems there is an obvious choice for the direction of the unique axis and then only a single term needs to be refined. If the number is close to 1, then the correction is not needed.

The second method for reducing the effect of texture in a structural refinement is to create a crystallite orientation probability surface as an expansion in terms spherical harmonic functions. Only functions consistent with cylindrical diffraction symmetry and having texture symmetry consistent with the Laue class of phase are used and are allowed, so the higher the symmetry the fewer terms that are available for a given spherical harmonics order. To use this correction, select the lowest order that provides refinable terms and perform a refinement. If the texture index remains close to one, then the correction is not needed. If a significant improvement is noted in the profile Rwp, one may wish to see if a higher order expansion gives an even larger improvement.

To characterize texture in a material, generally one needs data collected with the sample at multiple orientations or, for TOF, with detectors at multiple locations around the sample. In this case the detector orientation is given in each histogram's Sample Parameters and the sample's orientation is described with the Euler angles specified on the phase's Texture tab, which is also where the texture type (cylindrical, rolling,...) and the spherical harmonic order is selected. This should not be used with a single dataset and should not be used if the preferred orientations corrections are used.

The coordinate system used for texture characterization is defined where the sample coordinates (Psi, gamma) are defined with an instrument coordinate system (I, J, K) such that K is normal to the diffraction plane and J is coincident with the direction of the incident radiation beam toward the source. We further define a standard set of right-handed goniometer eulerian angles (Omega, Chi, Phi) so that Omega and Phi are rotations about K and Chi is a rotation about J when Omega = 0. Finally, as the sample may be mounted so that the sample coordinate system (Is, Js, Ks) does not coincide with the instrument coordinate system (I, J, K), we define three eulerian sample rotation angles (Omega-s, Chi-s, Phi-s) that describe the rotation from (Is, Js, Ks) to (I, J, K). The sample rotation angles are defined so that with the goniometer angles at zero Omega-s and Phi-s are rotations about K and Chi-s is a rotation about J.

Three typical examples:

- 1) Bragg-Brentano laboratory diffractometer: Chi=0
- 2) Debye-Scherrer counter detector; sample capillary axis perpendicular to diffraction plane: Chi=90
- 3) Debye-Scherrer 2D area detector positioned directly behind sample; sample capillary axis horizontal; Chi=0

NB: The area detector azimuthal angle will equal 0 in horizontal plane to right as viewed from x-ray source and will equal 90 at vertical "up" direction.

2.16 ISODISTORT implementation

CIFs prepared with the ISODISTORT web site https://stokes.byu.edu/iso/isodistort_version5.6.1/isodistort.php [B. J. Campbell, H. T. Stokes, D. E. Tanner, and D. M. Hatch, “ISODISPLACE: An Internet Tool for Exploring Structural Distortions.” J. Appl. Cryst. 39, 607-614 (2006).] can be read into GSAS-II using `import CIF`. This will cause constraints to be established for structural distortion modes read from the CIF. At present, of the five types of modes only `displacive(_iso_displacivemode...)` and `occupancy(_iso_occupancymode...)` are processed. Not yet processed: `_iso_magneticmode...`, `_iso_rotationalmode...` & `_iso_strainmode...`

The CIF importer `G2phase_CIF` implements class `G2phase_CIF.CIFPhaseReader` which offers two methods associated with ISODISTORT (ID) input. Method `G2phase_CIF.CIFPhaseReader.ISODISTORT_test()` checks to see if a CIF block contains the loops with `_iso_displacivemode_label` or `_iso_occupancymode_label` items. If so, method `G2phase_CIF.CIFPhaseReader.ISODISTORT_proc()` is called to read and interpret them. The results are placed into the reader object's `.Phase` class variable as a dict item with key 'ISODISTORT'.

Note that each mode ID has a long label with a name such as `Pm-3m[1/2,1/2,1/2]R5+(a,a,0)[La:b:dsp]T1u(a)`. Function `G2phase_CIF.ISODISTORT_shortLbl()` is used to create a short name for this, such as `R5_T1u(a)` which is made unique by addition of `_n` if the short name is duplicated. As each mode is processed, a constraint corresponding to that mode is created and is added to list in the reader object's `.Constraints` class variable. Items placed into that list can either be a list, which corresponds to a function (new var) type *constraint definition* entry, or an item can be a dict, which provides help information for each constraint.

2.16.1 Displacive modes

The coordinate variables, as named by ISODISTORT, are placed in `.Phase['ISODISTORT']['IsoVarList']` and the corresponding `GSASIIobj.G2VarObj` objects for each are placed in `.Phase['ISODISTORT']['G2VarList']`. The mode variables, as named by ISODISTORT, are placed in `.Phase['ISODISTORT']['IsoModeList']` and the corresponding `GSASIIobj.G2VarObj` objects for each are placed in `.Phase['ISODISTORT']['G2ModeList']`. [Use `str(G2VarObj)` to get the variable name from the `G2VarObj` object, but note that the phase number, *n*, for the prefix “*n*::” cannot be determined as the phase number is not yet assigned.]

Displacive modes are a bit complex in that they relate to delta displacements, relative to an offset value for each coordinate, and because the modes are normalized. While GSAS-II also uses displacements, these are added to the coordinates after each refinement cycle and then the delta values are set to zero. ISODISTORT uses fixed offsets (subtracted from the actual position to obtain the delta values) that are taken from the parent structure coordinate and the initial offset value (in `_iso_deltacoordinate_value`) and these are placed in `.Phase['ISODISTORT']['G2coordOffset']` in the same order as `.Phase['ISODISTORT']['G2ModeList']`, `.Phase['ISODISTORT']['IsoVarList']` and `“.Phase[ISODISTORT]['G2parentCoords’]”`.

The normalization factors (which the delta values are divided by) are taken from `_iso_displacivemodenorm_value` and are placed in `.Phase['ISODISTORT']['NormList']` in the same order as `...['IsoModeList']` and `...['G2ModeList']`.

The CIF contains a sparse matrix, from the loop_ containing `_iso_displacivemodematrix_value` which provides the equations for determining the mode values from the coordinates, that matrix is placed in `.Phase['ISODISTORT']['Mode2VarMatrix']`. The matrix is inverted to produce `.Phase['ISODISTORT']['Var2ModeMatrix']`, which determines how to compute the mode values from the delta coordinate values. These values are used for the in `GSASIIconstrGUI.ShowIsoDistortCalc()`, which shows coordinate and mode values, the latter with s.u. values.

2.16.2 Occupancy modes

The delta occupancy variables, as named by ISODISTORT, are placed in `.Phase['ISODISTORT']['OccVarList']` and the corresponding `GSASIIobj.G2VarObj` objects for each are placed in `.Phase['ISODISTORT']['G2OccVarList']`. The mode variables, as named by ISODISTORT, are placed in `.Phase['ISODISTORT']['OccModeList']` and the corresponding `GSASIIobj.G2VarObj` objects for each are placed in `.Phase['ISODISTORT']['G2OccModeList']`.

Occupancy modes, like Displacive modes, are also refined as delta values. However, GSAS-II directly refines the fractional occupancies. Offset values for each atom, are taken from `_iso_occupancy_formula` and are placed in `.Phase['ISODISTORT']['ParentOcc']`. (Offset values are subtracted from the actual position to obtain the delta values.) Modes are normalized (where the mode values are divided by the normalization factor) are taken from `_iso_occupancymodenorm_value` and are placed in `.Phase['ISODISTORT']['OccNormList']` in the same order as `...['OccModeList']` and `...['G2OccModeList']`.

The CIF contains a sparse matrix, from the `loop_` containing `_iso_occupancymodematrix_value`, which provides the equations for determining the mode values from the coordinates. That matrix is placed in `.Phase['ISODISTORT']['Occ2VarMatrix']`. The matrix is inverted to produce `.Phase['ISODISTORT']['Var2OccMatrix']`, which determines how to compute the mode values from the delta coordinate values.

2.16.3 Mode Computations

Constraints are processed after the CIF has been read in `GSASIIdataGUI.GSASII.OnImportPhase()` or `GSASIIscriptable.G2Project.add_phase()` by moving them from the reader object's `.Constraints` class variable to the Constraints tree entry's `['Phase']` list (for list items defining constraints) or the Constraints tree entry's `['_Explain']` dict (for dict items defining constraint help information)

The information in `.Phase['ISODISTORT']` is used in `GSASIIconstrGUI.ShowIsoDistortCalc()` which shows coordinate and mode values, the latter with s.u. values. This can be called from the Constraints and Phase/Atoms tree items.

Before each refinement, constraints are processed as *described elsewhere*. After a refinement is complete, `GSASIIstrIO.PrintIndependentVars()` shows the shifts and s.u.'s on the refined modes, using GSAS-II values, but `GSASIIstrIO.PrintISOmodes()` prints the ISODISTORT modes as computed in the web site.

2.17 Parameter Limits

One of the most often requested “enhancements” for GSAS-II would be the inclusion of constraints to force parameters such as occupancies or Uiso values to stay within expected ranges. While it is possible for users to supply their own restraints that would perform this by supplying an appropriate expression with the “General” restraints, the GSAS-II authors do not feel that use of restraints or constraints are a good solution for this common problem where parameters refine to non-physical values. This is because when this occurs, most likely one of the following cases is occurring:

1. there is a significant problem with the model, for example for an x-ray fit if an O atom is placed where a S is actually present, the Uiso will refine artificially small or the occupancy much larger than unity to try to compensate for the missing electrons; or
2. the data are simply insensitive to the parameter or combination of parameters, for example unless very high-Q data are included, the effects of a occupancy and Uiso value can have compensating effects, so an assumption must be made; likewise, with neutron data natural-abundance V atoms are nearly invisible due to weak coherent scattering. No parameters can be fit for a V atom with neutrons.

- the parameter is non-physical (such as a negative Uiso value) but within two sigma (sigma = standard uncertainty, aka e.s.d.) of a reasonable value, in which case the value is not problematic as it is experimentally indistinguishable from an expected value.
- there is a systematic problem with the data (experimental error)

In all these cases, this situation needs to be reviewed by a crystallographer to decide how to best determine a structural model for these data. An implementation with a constraint or restraint is likely to simply hide the problem from the user, making it more probable that a poor model choice is obtained.

What GSAS-II does implement is to allow users to specify ranges for parameters that works by disabling refinement of parameters that refine beyond either a lower limit or an upper limit, where either or both may be optionally specified. Parameters limits are specified in the Controls tree entry in dicts named as `Controls['parmMaxDict']` and `Controls['parmMinDict']`, where the keys are `G2VarObj` objects corresponding to standard GSAS-II variable (see `getVarDescr()` and `CompileVarDesc()`) names, where a wildcard (*) may optionally be used for histogram number or atom number (phase number is intentionally not allowed as a wildcard as it makes little sense to group the same parameter together different phases). Note that `prmLookup()` is used to see if a name matches a wildcard. The upper or lower limit is placed into these dicts as a float value. These values can be edited using the window created by the Calculate/"View LS parms" menu command or in scripting with the `GSASIIscriptable.G2Project.set_Controls()` function. In the GUI, a checkbox labeled "match all histograms/atoms" is used to insert a wildcard into the appropriate part of the variable name.

When a refinement is conducted, routine `GSASIIstrMain.dropOOBvars()` is used to find parameters that have refined to values outside their limits. If this occurs, the parameter is set to the limiting value and the variable name is added to a list of frozen variables (as a `G2VarObj` objects) kept in a list in the `Controls['parmFrozen']` dict. In a sequential refinement, this is kept separate for each histogram as a list in `Controls['parmFrozen'][histogram]` (where the key is the histogram name) or as a list in `Controls['parmFrozen']['FrozenList']` for a non-sequential fit. This allows different variables to be frozen in each section of a sequential fit. Frozen parameters are not included in refinements through removal from the list of parameters to be refined (`varyList`) in `GSASIIstrMain.Refine()` or `GSASIIstrMain.SeqRefine()`. The data window for the Controls tree item shows the number of Frozen variables and the individual variables can be viewed with the Calculate/"View LS parms" menu window or obtained with `GSASIIscriptable.G2Project.get_Frozen()`. Once a variable is frozen, it will not be refined in any future refinements unless the the variable is removed (manually) from the list. This can also be done with the Calculate/"View LS parms" menu window or `GSASIIscriptable.G2Project.set_Frozen()`.

See also:

```
G2VarObj getVarDescr() CompileVarDesc() prmLookup() GSASIIctrlGUI.ShowLSParms
GSASIIctrlGUI.VirtualVarBox GSASIIstrIO.SetUsedHistogramsAndPhases()
GSASIIstrIO.SaveUpdatedHistogramsAndPhases() GSASIIstrIO.SetSeqResult()
GSASIIstrMain.dropOOBvars() GSASIIscriptable.G2Project.set_Controls()
GSASIIscriptable.G2Project.get_Frozen() GSASIIscriptable.G2Project.set_Frozen()
```

2.18 GSASIIobj Classes and routines

Classes and routines defined in `GSASIIobj` follow.

`GSASIIobj.AddPhase2Index(rdObj, filename)`

Add a phase to the index during reading Used where constraints are generated during import (ISODISTORT CIFs)

`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. These variables are also used in script *makeVarTbl.py* which creates the table in section 3.2 of the Sphinx docs (*Variable names in GSAS-II*).

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 U\1',
```

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

```
GSASIIobj.CreatePDFItems(G2frame, PWDRtree, ElList, Qlimits, numAtm=1, FltBkg=0, PDFnames=[])
```

Create and initialize a new set of PDF tree entries

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

```
GSASIIobj.DefaultControls = {'Author': 'no name', 'Copy2Next': False, 'F**2': False, 'FreePrm1': 'Sample humidity (%)', 'FreePrm2': 'Sample voltage (V)', 'FreePrm3': 'Applied load (MN)', 'HatomFix': False, 'Reverse Seq': False, 'SVDtol': 1e-06, 'ShowCell': False, 'UsrReject': {'MaxD': 500.0, 'MaxDF/F': 100.0, 'MinD': 0.05, 'MinExt': 0.01, 'minF/sig': 0.0}, 'deriv type': 'analytic Hessian', 'max cyc': 3, 'min dM/M': 0.001, 'newLeBail': False, '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:

```
[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

__init__(exprObj)**__weakref__**

list of weak references to the object

compiledExpr

The expression as compiled byte-code

eObj

The expression and mappings; a *ExpressionObj* object

exprDict

dict that defines values for labels used in expression and packages referenced by functions

fxnpkgdict

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

lblLookup

Lookup table that specifies the expression label name that is tied to a particular GSAS-II parameters in the parmDict.

parmDict

A copy of the parameter dictionary, for distance and angle computation

su

Standard error evaluation where supplied by the evaluator

varLookup

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*, *varRefflag*)

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
- **varRefflag** (*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*, *varRefflag*)

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

- **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

__init__ ()**__weakref__**

list of weak references to the object

assgnVars

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

The expression as a text string

freeVars

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

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

GSASIIobj.FindFunction (*f*)

Find the object corresponding to function f

Parameters

f (*str*) – a function name such as ‘numpy.exp’

Returns

(pkgdict, pkgobj) where pkgdict contains a dict that defines the package location(s) and where pkgobj defines the object associated with the function. If the function is not found, pkgobj is None.

exception GSASIIobj.G2Exception (*msg*)

A generic GSAS-II exception class

__init__ (*msg*)

`__str__()`

Return str(self).

`__weakref__`

list of weak references to the object

exception `GSASIIobj.G2RefineCancel(msg)`

Raised when Cancel is pressed in a refinement dialog

`__init__(msg)`

`__str__()`

Return str(self).

`__weakref__`

list of weak references to the object

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 `GSASIIstrIO.GetUsedHistogramsAndPhases()`, which calls `IndexAllIds()` (or `GSASIIscriptable.G2Project.index_ids()`) 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 `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:

Parameters

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

__eq__ (*other*)

Allow comparison of G2VarObj to other G2VarObj objects or strings. If any field is a wildcard (“*”) that field matches.

__hash__ ()

Allow G2VarObj to be a dict key by implementing hashing

__init__ (*args)

__repr__ ()

Return the detailed contents of the object

__str__ ()

Return str(self).

__weakref__

list of weak references to the object

_show ()

For testing, shows the current lookup table

fmtVarByMode (*seqmode*, *note*, *warnmsg*)

Format a parameter object for display. Note that these changes are only temporary and are only shown only when the Constraints data tree is selected.

- In a non-sequential refinement or where the mode is ‘use-all’, the name is converted unchanged to a str
- In a sequential refinement when the mode is ‘wildcards-only’ the name is converted unchanged to a str but a warning is added for non-wildcarded HAP or Histogram parameters
- In a sequential refinement or where the mode is ‘auto-wildcard’, a histogram number is converted to a wildcard (*) and then converted to str

Parameters

- **mode** (*str*) – the sequential mode (see above)
- **note** (*str*) – value displayed on the line of the constraint/equiv.
- **warnmsg** (*str*) – a message saying the constraint is not used

Returns

varname, explain, note, warnmsg (all str values) where:

- varname is the parameter expressed as a string,
- explain is blank unless there is a warning explanation about the parameter or blank
- note is the previous value unless overridden
- warnmsg is the previous value unless overridden

varname (*hist=None*)

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

Parameters

hist (*str/int*) – if specified, overrides the histogram number with the specified value

Returns

the variable name as a str

`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:

```
varlist = ['0::AUiso:0', '0::AUiso:1', '1::AUiso:0']
```

then the output will be this:

```
wildList = ['*::AUiso:0', '0::AUiso:*']
```

Parameters

varlist (*list*) – an input list of GSAS-II variable names (such as 0::AUiso:0)

Returns

wildList, the generated list of wild card variable names.

`GSASIIobj.GetPhaseNames (fl)`

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

Parameters

fl (*file*) – opened .gpx file

Returns

list of phase names

`GSASIIobj.HistIdLookup = {}`

dict listing histogram name and random Id, keyed by sequential histogram index as a str; best to access this using `LookupHistName()`

`GSASIIobj.HistRanIdLookup = {}`

dict listing histogram sequential index keyed by histogram random Id; best to access this using `LookupHistId()`

`GSASIIobj.HowDidIgetHere (whererecalledonly=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 ImportError

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.

__weakref__

list of weak references to the object

ReInitialize()

Reinitialize the Reader to initial settings

__init__ (*formatName, longFormatName=None, extensionlist=[], strictExtension=False*)

__weakref__

list of weak references to the object

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 `GSASIIIO.ReadImages()`, which puts image info directly into the data tree.
- Images are reloaded with `GSASIIIO.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 Å.
 - `'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*, *imtag=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

__init__ (*formatName*, *longFormatName=None*, *extensionlist=[]*, *strictExtension=False*)

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

__init__ (*formatName*, *longFormatName=None*, *extensionlist=[]*, *strictExtension=False*)

class GSASIIobj.**ImportPhase** (*formatName*, *longFormatName=None*, *extensionlist=[]*, *strictExtension=False*)

Defines a base class for the reading of files with coordinates

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.

__init__ (*formatName*, *longFormatName=None*, *extensionlist=[]*, *strictExtension=False*)

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/G2pwd_*.py` etc.) will be used in `GSASIIdataGUI.GSASII.OnImportPowder()` 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

__init__ (*formatName*, *longFormatName=None*, *extensionlist=[]*, *strictExtension=False*)

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

__init__ (*formatName*, *longFormatName=None*, *extensionlist=[]*, *strictExtension=False*)

```
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

```
__init__ (formatName, longFormatName=None, extensionlist=[], strictExtension=False)
```

```
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

`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

`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

```
__init__ (formatName, longFormatName=None, extensionlist=[], strictExtension=False)
```

```
GSASIIobj.IndexAllIds (Histograms, 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 when phases & histograms are looked up in these places (only):

- `GSASIIstrIO.GetUsedHistogramsAndPhases ()` (which loads the histograms and phases from a GPX file),
- `GetUsedHistogramsAndPhasesfromTree ()` (which does the same thing but from the data tree.)
- `OnFileClose ()` (clears out an old project)

Note that globals *PhaseIdLookup* and *PhaseRanIdLookup* are also set in *AddPhase2Index()* to temporarily assign a phase number as a phase is being imported.

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

GSASIIobj.LookupAtomId (*pId*, *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

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()`

`__init__()`

`__weakref__`

list of weak references to the object

`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.TestIndexAll ()`

Test if `IndexAllIds ()` has been called to index all phases and histograms (this is needed before `G2VarObj ()` can be used.

Returns

Returns True if indexing is needed.

`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._lookup(dic, key)`

Lookup a key in a dictionary, where None returns an empty string but an unmatched key returns a question mark.

Used in `G2VarObj`

`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.prmLookup(name, prmDict)`

Looks for a parameter in a min/max dictionary, optionally considering a wild card for histogram or atom number (use of both will never occur at the same time).

Parameters

- **name** – a GSAS-II parameter name (str, see `getVarDescr()` and `CompileVarDesc()`) or a `G2VarObj` object.

- **prmDict** (*dict*) – a min/max dictionary, (parmMinDict or parmMaxDict in Controls) where keys are *G2VarObj* objects.

Returns

Two values, (**matchname**, **value**), are returned where:

- **matchname** (*str*) is the *G2VarObj* object corresponding to the actual matched name, which could contain a wildcard even if **name** does not; and
- **value** (*float*) which contains the parameter limit.

```

GSASIIobj.reVarDesc = {re.compile('([UVW])$'): 'Gaussian instrument broadening
\\1', re.compile('([XYZ])$'): 'Cauchy instrument broadening \\1',
re.compile('([XYZ])cos'): 'Cos position wave for \\1',
re.compile('([XYZ])max'): 'ZigZag/Block max value for \\1',
re.compile('([XYZ])sin'): 'Sin position wave for \\1', re.compile('([abc])$'):
'Lattice parameter, \\1, from Ai and Djk', re.compile('([vV]ol)': 'Unit cell
volume', re.compile('A([0-5])'): 'Reciprocal metric tensor component \\1',
re.compile('A([xyz])$'): 'Fractional atomic coordinate, \\1',
re.compile('AD\\([0-6],-[0-6]\\)([0-6])'): 'Atomic sp. harm. coeff for
orbital, \\1', re.compile('AD\\([0-6],[0-6]\\)([0-6])'): 'Atomic sp. harm.
coeff for orbital, \\1', re.compile('AM([xyz])$'): 'Atomic magnetic moment
parameter, \\1', re.compile('ANe([01])'): 'Atomic <j0> orbital population for
orbital, \\1', re.compile('AU([123][123])'): 'Atomic anisotropic displacement
parameter U\\1', re.compile('AUiso'): 'Atomic isotropic displacement
parameter', re.compile('Absorption'): 'Absorption coef.', re.compile('Afrac'):
'Atomic site fraction parameter', re.compile('Akappa([0-6])'): 'Atomic
orbital softness for orbital, \\1', re.compile('Amul'): 'Atomic site
multiplicity value', re.compile('Aspect ratio'): 'Particle aspect ratio',
re.compile('B$'): 'Porod prefactor', re.compile('BF mult'): 'Background file
multiplier', re.compile('Bab([AU])'): 'Babinet solvent scattering coef. \\1',
re.compile('Back$'): 'background parameter', re.compile('Back(.)'):
'Background term #\\1', re.compile('BkPkgam;(.*)'): 'Background peak #\\1
Cauchy width', re.compile('BkPkint;(.*)'): 'Background peak #\\1 intensity',
re.compile('BkPkpos;(.*)'): 'Background peak #\\1 position',
re.compile('BkPkSig;(.*)'): 'Background peak #\\1 Gaussian width',
re.compile('C\\([0-9]*,[0-9]*\\)'): 'spherical harmonics preferred orientation
coef.', re.compile('Cutoff'): 'Porod cutoff', re.compile('D([123][123])'):
'Anisotropic strain coef. \\1', re.compile('Dcalc'): 'Calc. d-spacing',
re.compile('DebyeA'): 'Debye model amplitude', re.compile('DebyeR'): 'Debye
model radius', re.compile('DebyeU'): 'Debye model Uiso', re.compile('Depth'):
'Well depth', re.compile('Diameter'): 'Cylinder/disk diameter',
re.compile('Displace([XY])'): 'Debye-Scherrer sample displacement \\1',
re.compile('Dist'): 'Interparticle distance', re.compile('Eg$'): 'Secondary
type I extinction', re.compile('Ep$'): 'Primary extinction',
re.compile('Es$'): 'Secondary type II extinction', re.compile('Extinction'):
'Extinction coef.', re.compile('Fcos'): 'Cos site fraction modulation',
re.compile('Flack'): 'Flack parameter', re.compile('FreePrm([123])'): 'User
defined measurement parameter \\1', re.compile('Fsin'): 'Sin site fraction
modulation', re.compile('Fwid'): 'Crenel function width', re.compile('Fzero'):
'Crenel function offset', re.compile('G$'): 'Guinier prefactor',
re.compile('Gonio. radius'): 'Distance from sample to detector, mm',
re.compile('I\\(L2\\)\\(I\\(L1\\)\\)'): 'Ka2/Ka1 intensity ratio',
re.compile('Lam'): 'Wavelength', re.compile('Layer Disp'): 'Layer displacement
along beam', re.compile('LayerDisp'): 'Bragg-Brentano Layer displacement',
re.compile('Length'): 'Cylinder length', re.compile('M([XYZ])cos$'): 'Cos mag.
moment wave for \\1', re.compile('M([XYZ])sin$'): 'Sin mag. moment wave for
\\1', re.compile('MD'): 'March-Dollase coef.', re.compile('Mean'): 'Particle
mean radius', re.compile('Mustrain;.*'): 'Microstrain coefficient (delta Q/Q x
10**6)', re.compile('P$'): 'Porod power', re.compile('PDFmag'): 'PDF peak
magnitude', re.compile('PDFpos'): 'PDF peak position', re.compile('PDFsig'):
'PDF peak std. dev.', re.compile('PkGam'): 'Bragg peak gamma',
re.compile('PkInt'): 'Bragg peak intensity', re.compile('PkPos'): 'Bragg peak
position', re.compile('PkSig'): 'Bragg peak sigma', re.compile('Polariz.'):
'Polarization correction', re.compile('Pressure'): 'Pressure level for
measurement in MPa', re.compile('RBR([TLS])([123AB][123AB])'): 'Residue rigid
body group disp. param.', re.compile('RBRO([aijk])'): 'Residue rigid body
orientation parameter \\1', re.compile('RBRTr;.*'): 'Residue rigid body
position parameter', re.compile('RBRTr;.*'): 'Residue rigid body torsion
parameter', re.compile('RBRU'): 'Residue rigid body group Uiso param.',
re.compile('RBRf'): 'Residue rigid body site fraction', re.compile('RBSatNo'):

```


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 = {re.compile('([UVW])$'): 1e-05, re.compile('([XYZ])$'):
1e-05, re.compile('A([0-5])'): 1e-05, re.compile('AU([123][123])'): 0.0001,
re.compile('AUiso'): 0.0001, re.compile('Afrac'): 1e-05,
re.compile('Displace([XY])'): 0.1, re.compile('I\\(L2\\)\\(L1\\)'): 0.001,
re.compile('Lam'): 1e-06, re.compile('Polariz.'): 0.001, re.compile('SH/L'):
0.0001, re.compile('dA([xyz])$'): 1e-06}
```

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()`.

`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

`GSASIIobj.validateAtomDrawType(typ, generalData={})`

Confirm that the selected Atom drawing type is valid for the current phase. If not, use ‘vdW balls’. This is currently used only for setting a default when atoms are added to the atoms draw list.

GSASIISCRIPTABLE: SCRIPTING INTERFACE

3.1 Summary/Contents

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 *G2ObjectWrapper*) for a growing number of data tree items.

GSASIIscriptable can be used in two ways. It offers a command-line mode, but the more widely used and more powerful mode of GSASIIscriptable is used is via Python scripts that call the module's application interface (API), these are summarized immediately below and are documented in the *complete API documentation* section.

While the command-line mode provides access a number of features without writing Python scripts via shell/batch commands (see *GSASIIscriptable Command-line Interface*), use in practice seems somewhat clumsy. Command-line mode is no longer being developed and its use is discouraged.

Scripting Documentation Contents

- GSASIIscriptable: Scripting Interface
 - Summary/Contents
 - *Installation of GSASIIscriptable*
 - *Application Interface (API) Summary*
 - *Refinement parameters*
 - *Specifying Refinement Parameters*
 - *Access to other parameter settings*
 - *Code Examples*
 - *GSASIIscriptable Command-line Interface*
 - *API: Complete Documentation*

3.2 Installation of GSASIIscriptable

GSASIIscriptable is included as part of a standard GSAS-II installation that includes the GSAS-II GUI (as described in the [installation instructions](#)). People who will use scripting extensively will still need access to the GUI for some activities, since the scripting API has not yet been extended to all features of GSAS-II and even if that is ever completed, there will still be some things that GSAS-II does with the GUI would be almost impossible to implement without a interactive graphical view of the data.

Nonetheless, there may be times where it does make sense to install GSAS-II without all of the GUI components, for example on a compute server. The minimal requirements for use of GSASIIscriptable are only Python, numpy and scipy, but additional optional packages that can be utilized are described in the [Scripting Requirements](#) section of the requirements chapter, which also provides some installation instructions.

In a standard GSAS-II installation, no changes are made to Python. When the GUI is invoked, a small script or Windows batch file is used to start GSAS-II inside Python. When GSASIIscriptable is used, Python must be provided with the location of the GSAS-II files. There are two ways this can be done:

1. define the GSAS-II installation location in the Python `sys.path`, or
2. install a reference to GSAS-II inside Python.

The latter method requires an extra installation step, but has the advantage that it allows writing portable GSAS-II scripts. This is discussed further in the [Shortcut for Scripting Access](#) section of this chapter.

3.3 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.

3.3.1 Overview of Classes

| Scripting class name | Description |
|----------------------|--|
| <i>G2Project</i> | <i>G2Project</i> : A GSAS-II project file; provides references to objects below, each corresponding to a tree item (exception is <i>G2AtomRecord</i>) |
| <i>G2Phase</i> | <i>G2Phase</i> : Provides phase information access (also provides access to atom info via <i>G2AtomRecord</i>) |
| <i>G2AtomRecord</i> | <i>G2AtomRecord</i> : Access to an atom within a phase |
| <i>G2PwdrData</i> | <i>G2PwdrData</i> : Access to powder histogram info |
| <i>G2Single</i> | <i>G2Single</i> : Access to single crystal histogram info |
| <i>G2Image</i> | <i>G2Image</i> : Access to image info |
| <i>G2PDF</i> | <i>G2PDF</i> : PDF histogram info |
| <i>G2SeqRefRes</i> | <i>G2SeqRefRes</i> : The sequential results table |

3.3.2 Independent Functions

A small number of Scriptable routines do not require use of objects.

| method | Use |
|-----------------------------------|---|
| <i>GenerateReflections()</i> | Generates a list of unique powder reflections |
| <i>SetPrintLevel()</i> | Sets the amount of output generated when running a script |
| <i>installScriptingShortcut()</i> | Installs GSASIIscriptable within Python as G2script |

3.3.3 Class G2Project

All GSASIIscriptable 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:

| method | Use |
|---|--|
| <code>save()</code> | Writes the current project to disk. |
| <code>add_powder_histogram()</code> | Used to read in powder diffraction data into a project file. |
| <code>add_simulated_powder_histogram()</code> | Defines a “dummy” powder diffraction data that will be simulated after a refinement step. |
| <code>add_image()</code> | Reads in an image into a project. |
| <code>add_phase()</code> | Adds a phase to a project |
| <code>add_PDF()</code> | Adds a PDF entry to a project (does not compute it) |
| <code>add_single_histogram()</code> | Used to read in a single crystal diffraction dataset into a project file. |
| <code>histogram()</code> | Finds a histogram from an object, name or random id reference, returning a <i>G2PwdrData</i> or <i>G2Single</i> object. |
| <code>histograms()</code> | Provides a list of histograms in the current project, as <i>G2PwdrData</i> or as <i>G2Single</i> objects. |
| <code>histType()</code> | Determines the histogram type from an object, name or random id reference. |
| <code>phases()</code> | Provides a list of phases defined in the current project, as <i>G2Phase</i> objects |
| <code>images()</code> | Provides a list of images in the current project, as <i>G2Image</i> objects |
| <code>pdfs()</code> | Provides a list of PDFs in the current project, as <i>G2PDF</i> objects |
| <code>seqref()</code> | Returns a <i>G2SeqRefRes</i> object if there are Sequential Refinement results |
| <code>do_refinements()</code> | 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 <i>Project-level Parameter Dict</i>). Also see <i>Refinement recipe</i> . |
| <code>set_refinement()</code> | This is passed a single dict which is used to set parameters and flags. These actions can be performed also in <code>do_refinements()</code> . |
| <code>get_Variable()</code> | Retrieves the value and esd for a parameter |
| <code>get_Covariance()</code> | Retrieves values and covariance for a set of refined parameters |
| <code>set_Controls()</code> | Set overall GSAS-II control settings such as number of cycles and to set up a sequential fit. (Also see <code>get_Controls()</code> to read values.) |
| <code>imageMultiDistCalib()</code> | Performs a global calibration fit with images at multiple distance settings. |
| <code>get_Constraints()</code> | Retrieves <i>constraint definition</i> entries. |
| <code>add_HoldConstr()</code> | Adds a hold constraint on one or more variables |
| <code>add_EquivConstr()</code> | Adds an equivalence constraint on two or more variables |
| <code>add_EqnConstr()</code> | Adds an equation-type constraint on two or more variables |
| <code>add_NewVarConstr()</code> | Adds a new variable as a constraint on two or more variables |
| <code>ComputeWorstFit()</code> | Determines the parameters that will have the greatest impact on the fit if refined |

3.3.4 Class G2Phase

Another common object in GSASIIscriptable scripts is *G2Phase*, used to encapsulate each phase in a project, with commonly used methods:

| method | Use |
|---------------------------------|---|
| <i>set_refinements()</i> | Provides a mechanism to set values and refinement flags for the phase. See <i>Phase parameters</i> for more details. This information also can be supplied within a call to <i>do_refinements()</i> or <i>set_refinement()</i> . |
| <i>clear_refinements()</i> | Unsets refinement flags for the phase. |
| <i>set_HAP_refinements()</i> | Provides a mechanism to set values and refinement flags for parameters specific to both this phase and one of its histograms. See <i>Histogram-and-phase parameters</i> . This information also can be supplied within a call to <i>do_refinements()</i> or <i>set_refinement()</i> . |
| <i>clear_HAP_refinements()</i> | Clears refinement flags specific to both this phase and one of its histograms. |
| <i>getHAPvalues()</i> | Returns values of parameters specific to both this phase and one of its histograms. |
| <i>copyHAPvalues()</i> | Copies HAP settings between from one phase/histogram and to other histograms in same phase. |
| <i>HAPvalue()</i> | Sets or retrieves values for some of the parameters specific to both this phase and one or more of its histograms. |
| <i>atoms()</i> | Returns a list of atoms in the phase |
| <i>atom()</i> | Returns an atom from its label |
| <i>add_atom()</i> | Adds an atom to a phase |
| <i>histograms()</i> | Returns a list of histograms linked to the phase |
| <i>get_cell()</i> | Returns unit cell parameters (also see <i>get_cell_and_esd()</i>) |
| <i>export_CIF()</i> | Writes a CIF for the phase |
| <i>setSampleProfile()</i> | Sets sample broadening parameters |
| <i>clearDistRestraint()</i> | Clears any previously defined bond distance restraint(s) for the selected phase |
| <i>addDistRestraint()</i> | Finds and defines new bond distance restraint(s) for the selected phase |
| <i>setDistRestraintWeight()</i> | Sets the weighting factor for the bond distance restraints |

3.3.5 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 |
|----------------------------------|--|
| <code>set_refinements()</code> | Provides a mechanism to set values and refinement flags for the powder histogram. See <i>Histogram parameters</i> for details. |
| <code>clear_refinements()</code> | Unsets refinement flags for the powder histogram. |
| <code>residuals()</code> | Reports R-factors etc. for the powder histogram (also see <code>get_wR()</code>) |
| <code>add_back_peak()</code> | Adds a background peak to the histogram. Also see <code>del_back_peak()</code> and <code>ref_back_peak()</code> . |
| <code>fit_fixed_points()</code> | Fits background to the specified fixed points. |
| <code>set_background()</code> | Sets a background histogram that will be subtracted (point by point) from the current histogram. |
| <code>calc_autobkg()</code> | Estimates the background and sets the fixed background points from that. |
| <code>getdata()</code> | Provides access to the diffraction data associated with the histogram. |
| <code>reflections()</code> | Provides access to the reflection lists for the histogram. |
| <code>Export()</code> | Writes the diffraction data or reflection list into a file |
| <code>add_peak()</code> | Adds a peak to the peak list. Also see <i>Peak Fitting</i> . |
| <code>set_peakFlags()</code> | Sets refinement flags for peaks |
| <code>refine_peaks()</code> | Starts a peak/background fitting cycle, returns refinement results |
| <code>Peaks</code> | Provides access to the peak list data structure |
| <code>PeakList</code> | Provides the peak list parameter values |
| <code>Export_peaks()</code> | Writes the peak parameters to a text file |

3.3.6 Class G2Single

A less object in GSASIIscriptable scripts is *G2Single*, which encapsulate each single crystal diffraction histogram in a project, with commonly used methods:

| method | Use |
|----------------------------------|---|
| <code>set_refinements()</code> | Provides a mechanism to set refinement flags for the single crystal histogram. See <i>Histogram parameters</i> for details. |
| <code>clear_refinements()</code> | Unsets refinement flags for the single crystal powder histogram. |

3.3.7 Class G2Image

When working with images, there will be a *G2Image* object for each image (also see *add_image()* and *images()*).

| method | Use |
|----------------------------|---|
| <i>Recalibrate()</i> | Invokes a recalibration fit starting from the current Image Controls calibration coefficients. |
| <i>Integrate()</i> | Invokes an image integration All parameters Image Controls will have previously been set. |
| <i>GeneratePixelMask()</i> | Searches for “bad” pixels creating a pixel mask. |
| <i>setControl()</i> | Set an Image Controls parameter in the current image. |
| <i>getControl()</i> | Return an Image Controls parameter in the current image. |
| <i>findControl()</i> | Get the names of Image Controls parameters. |
| <i>loadControls()</i> | Load controls from a .imctrl file (also see <i>saveControls()</i>). |
| <i>loadMasks()</i> | Load masks from a .immask file. |
| <i>setVary()</i> | Set a refinement flag for Image Controls parameter in the current image. (Also see <i>getVary()</i>) |
| <i>setCalibrant()</i> | Set a calibrant type (or show choices) for the current image. |
| <i>setControlFile()</i> | Set a image to be used as a background/dark/gain map image. |
| <i>getControls()</i> | Returns the Image Controls dict for the current image. |
| <i>setControls()</i> | Updates the Image Controls dict for the current image with specified key/value pairs. |
| <i>getMasks()</i> | Returns the Masks dict for the current image. |
| <i>setMasks()</i> | Updates the Masks dict for the current image with specified key/value pairs. |
| <i>IntThetaAzMap()</i> | Computes the set of 2theta-azimuth mapping matrices to integrate the current image. |
| <i>IntMaskMap()</i> | Computes the masking map for the current image for integration. |
| <i>MaskThetaMap()</i> | Computes the 2theta mapping matrix to determine a pixel mask. |
| <i>MaskFrameMask()</i> | Computes the Frame mask needed to determine a pixel mask. |
| <i>TestFastPixelMask()</i> | Returns True if fast pixel masking is available. |
| <i>clearImageCache()</i> | Clears a saved image from memory, if one is present. |
| <i>clearPixelMask()</i> | Clears a saved Pixel map from the project, if one is present. |

3.3.8 Class G2PDF

To work with PDF entries, object *G2PDF*, encapsulates a PDF entry with methods:

| method | Use |
|-------------------------|---|
| <i>export()</i> | Used to write G(r), etc. as a file |
| <i>calculate()</i> | Computes the PDF using parameters in the object |
| <i>optimize()</i> | Optimizes selected PDF parameters |
| <i>set_background()</i> | Sets the histograms used for sample background, container, etc. |
| <i>set_formula()</i> | Sets the chemical formula for the sample |

3.3.9 Class G2SeqRefRes

To work with Sequential Refinement results, object *G2SeqRefRes*, encapsulates the sequential refinement table with methods:

| method | Use |
|---------------------------|---|
| <i>histograms()</i> | Provides a list of histograms used in the Sequential Refinement |
| <i>get_cell_and_esd()</i> | Returns cell dimensions and standard uncertainties for a phase and histogram from the Sequential Refinement |
| <i>get_Variable()</i> | Retrieves the value and esd for a parameter from a particular histogram in the Sequential Refinement |
| <i>get_Covariance()</i> | Retrieves values and covariance for a set of refined parameters for a particular histogram |

3.3.10 Class G2AtomRecord

When working with phases, *G2AtomRecord* methods provide access to the contents of each atom in a phase. This provides access to atom values via class “properties” that can be used to get values of much of the atoms associated settings, as below. Most can also be used to set values via “setter” methods. See the *G2AtomRecord* docs and source code.

| method/property | Use |
|-------------------------|--|
| <i>label</i> | Reference as <code><atom>.label`</code> to get or set label value for atom |
| <i>type</i> | Reference as <code><atom>.G2AtomRecord.type</code> to get or set the atom type |
| <i>element</i> | Reference as <code><atom>.G2AtomRecord.element</code> to get the element symbol associated with an atom (change with <code><atom>.G2AtomRecord.type</code> , see <i>type</i>) |
| <i>refinement_flags</i> | Reference class property <code><atom>.G2AtomRecord.refinement_flags</code> to get or set the refinement flags associated with an atom |
| <i>coordinates</i> | Reference as <code><atom>.G2AtomRecord.coordinates</code> to get or set the three coordinates associated with an atom |
| <i>occupancy</i> | Reference class property <code><atom>.G2AtomRecord.occupancy</code> to get or set the site occupancy associated with an atom |
| <i>mult</i> | Reference as <code><atom>.G2AtomRecord.mult</code> to get an atom site multiplicity (value cannot be changed in script) |
| <i>ranId</i> | Reference as <code><atom>.G2AtomRecord.ranId</code> to get an atom random Id number (value cannot be changed in script) |
| <i>adp_flag</i> | Reference as <code><atom>.G2AtomRecord.adp_flag</code> to get either ‘U’ or ‘I’ specifying that an atom is set as anisotropic or isotropic (value cannot be changed in script) |
| <i>uiso</i> | Reference pseudo class variable <code><atom>.G2AtomRecord.uiso</code> to get or set the Uiso value associated with an atom |

3.4 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.

3.4.1 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 `set_refinement()` (or `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 `set_refinement()`, but if not specified, the default is to use all defined phases and histograms.

As an example:

```
pardict = {'set': { 'Limits': [0.8, 12.0],
                  'Sample Parameters': ['Absorption', 'Contrast', 'DisplaceX'],
                  'Background': {'type': 'chebyshev-1', 'refine': True,
                                'peaks': [[0, True], [1, 1, 1]] }},
          'clear': {'Instrument Parameters': ['U', 'V', 'W']}}
my_project.set_refinement(pardict)
```

3.4.2 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 `do_refinements()`.

As an example, this code performs the same actions as in the example in the section above:

```
pardict = {'set': { 'Limits': [0.8, 12.0],
                  'Sample Parameters': ['Absorption', 'Contrast', 'DisplaceX'],
                  'Background': {'type': 'chebyshev-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:

```
my_project.do_refinements([pardict, pardict1])
```

The keys defined in the following table may be used in a dict supplied to `do_refinements()`. Note that keys `histograms` and `phases` are used to limit actions to specific sets of parameters within the project.

| key | explanation |
|------------|---|
| set | Specifies a dict with keys and subkeys as described in the <i>Specifying Refinement Parameters</i> section. Items listed here will be set to be refined. |
| clear | Specifies a dict, as above for set, except that parameters are cleared and thus will not be refined. |
| once | 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. |
| skip | 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. |
| output | If a file name is specified for output it will be used to save the current refinement. |
| histograms | Should contain a list of histogram(s) to be used for the set/clear/once action(s) on <i>Histogram parameters</i> or <i>Histogram-and-phase parameters</i> . Note that this will be ignored for <i>Phase parameters</i> . Histograms may be specified as a list of strings [(‘PWDR ...’),...], indices [0,1,2] or as list of objects [hist1, hist2]. |
| phases | Should contain a list of phase(s) to be used for the set/clear/once action(s) on <i>Phase parameters</i> or <i>Histogram-and-phase parameters</i> . Note that this will be ignored for <i>Histogram parameters</i> . Phases may be specified as a list of strings [(‘Phase name’),...], indices [0,1,2] or as list of objects [phase0, phase2]. |
| call | 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>iter_refinements()</code> where <code>self</code> references the project.) Nothing is called if this is not specified. |
| callargs | 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><tt>G2Project</tt></code> is passed as a single argument. |

An example that performs a series of refinement steps follows:

```
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.

3.4.3 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.

3.5 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.

3.5.1 Histogram parameters

This table describes the dictionaries supplied to `set_refinements()` and `clear_refinements()`. As an example,

```
hist.set_refinements({"Background": {"no. coeffs": 3, "refine": True},
                     "Sample Parameters": ["Scale"],
                     "Limits": [10000, 40000]})
```

With `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:

```
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.

| key | subkey | explanation |
|-----------------------|--------------------------------|--|
| Limits | | 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] Available for powder histograms only. |
| | low | Sets the low limit |
| | high | Sets the high limit |
| Sample Parameters | | Should be provided as a list of subkeys to set or clear refinement flags for, e.g. ['DisplaceX', 'Scale'] Available for powder histograms only. |
| | Absorption | |
| | Contrast | |
| | DisplaceX | Sample displacement along the X direction |
| | DisplaceY | Sample displacement along the Y direction |
| | Scale | Histogram Scale factor |
| Background | | Sample background. Value will be a dict or a boolean. If True or False, the refine parameter for background is set to that. Available for powder histograms only. Note that background peaks are not handled via this; see <code>ref_back_peak()</code> instead. When value is a dict, supply any of the following keys: |
| | type | The background model, e.g. 'chebyshev-1' |
| | refine | The value of the refine flag, boolean |
| | 'no. coeffs' | Number of coefficients to use, integer |
| | coeffs | List of floats, literal values for background |
| | FixedPoints | List of (2-theta, intensity) values for fixed points |
| | 'fit fixed points' | 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. |
| | peaks | 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 & gam (fewer than 4 values are allowed). |
| Instrument Parameters | | As in Sample Parameters, provide as a list of subkeys to set or clear refinement flags, e.g. ['X', 'Y', 'Zero', 'SH/L'] Available for powder histograms only. |
| | U, V, W | Gaussian peak profile terms |
| | X, Y, Z | Lorentzian peak profile terms |
| | alpha, beta-0, beta-1, beta-q, | TOF profile terms |
| | sig-0, sig-1, sig-2, sig-q | TOF profile terms |
| | difA, difB, difC | TOF Calibration constants |
| | Zero | Zero shift |
| | SH/L | Finger-Cox-Jephcoat low-angle peak asymmetry |
| | Polariz. | Polarization parameter |
| | Lam | Lambda, the incident wavelength |
| Single xtal | | As in Sample Parameters, provide as a list of subkeys to set or clear refinement flags, e.g. [...]. Available for single crystal histograms only. |
| | Scale | Single crystal scale factor |
| | BabA, BabU | Babinet A & U parameters |
| | Eg, Es, Ep | Extinction parameters |
| | Flack | Flack absolute configuration parameter |

3.5.2 Phase parameters

This table describes the dictionaries supplied to `set_refinements()` and `clear_refinements()`. With `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).

| key | explanation |
|--------|---|
| Cell | Whether or not to refine the unit cell. |
| Atoms | 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 site fraction, 'X' for position, and 'U' for Debye-Waller factor |
| LeBail | Enables LeBail intensity extraction. |

3.5.3 Histogram-and-phase parameters

This table describes the dictionaries supplied to `set_HAP_refinements()` and `clear_HAP_refinements()`. When supplied to `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.

| key | subkey | explanation |
|------------|-----------|--|
| Babinet | | Should be a list of the following subkeys. If not, assumes both BabA and BabU |
| | BabA | |
| | BabU | |
| Extinction | | Boolean, True to refine. |
| HStrain | | Boolean or list/tuple, True to refine all appropriate D_{ij} terms or False to not refine any. If a list/tuple, will be a set of True & False values for each D_{ij} term; number of items must match number of terms. |
| Mustrain | | |
| | type | Mustrain model. One of 'isotropic', 'uniaxial', or 'generalized'. This should be specified to change the model. |
| | direction | For uniaxial only. A list of three integers, the [hkl] direction of the axis. |
| | refine | 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. |
| Size | | |
| | type | Size broadening model. One of 'isotropic', 'uniaxial', or 'ellipsoid'. This should be specified to change from the current. |
| | direction | For uniaxial only. A list of three integers, the [hkl] direction of the axis. |
| | refine | Boolean, True to refine. |
| | value | float, size value in microns |
| Pref.Ori. | | Boolean, True to refine |
| Show | | Boolean, True to refine |
| Use | | Boolean, True to refine |
| Scale | | Phase fraction; Boolean, True to refine |

3.5.4 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 *set_refinements()* or *clear_refinements()* for histogram parameters; *set_refinements()* or *clear_refinements()* for phase parameters; and *set_HAP_refinements()* or *clear_HAP_refinements()*. As an example, if *some_histogram* is a histogram object (of type *G2PwdrData*), use this to set parameters in that histogram:

```
params = { 'Limits': [0.8, 12.0],
           'Sample Parameters': ['Absorption', 'Contrast', 'DisplaceX'],
           'Background': {'type': 'chebyshev-1', 'refine': True}}
some_histogram.set_refinements(params)
```

Likewise to turn refinement flags on, use code such as this:

```
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.):

```
params = { 'Instrument Parameters': ['U', 'V', 'W']}
some_histogram.clear_refinements(params)
```

For phase parameters, use code such as this:

```
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:

```
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).

3.6 Access to other parameter settings

There are several hundred different types of values that can be stored in a GSAS-II project (.gpx) file. All can be changed from the GUI but only a subset have direct mechanism implemented for change from the GSASIIscriptable API. In practice all parameters in a .gpx file can be edited via scripting, but sometimes determining what should be set to implement a parameter change can be complex. Several routines, *getHAPentryList()*, *getPhaseEntryList()* and *getHistEntryList()* (and their related get...Value and set.Value entries), provide a mechanism to discover what the GUI is changing inside a .gpx file.

As an example, a user in changing the data type for a histogram from Debye-Scherrer mode to Bragg-Brentano. This capability is not directly exposed in the API. To find out what changes when the histogram type is changed we can create a short script that displays the contents of all the histogram settings:


```
from __future__ import division, print_function
import os, sys
sys.path.insert(0, '/Users/toby/software/G2/GSASII')
import GSASIIscriptable as G2sc
gpx = G2sc.G2Project('/tmp/test.gpx')
h = gpx.histograms()[0]
for h in h.getHistEntryList():
    print(h)
```

This can be run with a command like this:

```
python test.py > before.txt
```

(This will create file `before.txt`, which will contain hundreds of lines.)

At this point open the project file, `test.gpx` in the GSAS-II GUI and change in Histogram/Sample Parameters the diffractometer type from Debye-Scherrer mode to Bragg-Brentano and then save the file.

Rerun the previous script creating a new file:

```
python test.py > after.txt
```

Finally look for the differences between files `before.txt` and `after.txt` using a tool such as `diff` (on Linux/OS X) or `fc` (in Windows).

in Windows:

```
Z:\>fc before.txt after.txt
Comparing files before.txt and after.txt
***** before.txt
    fill_value = 1e+20)
, 'PWDR Co_PCP_Act_d900-00030.fxye Bank 1', 'PWDR Co_PCP_Act_d900-00030.fxye Bank 1'])
(['Comments'], <class 'list'>, ['Co_PCP_Act_d900-00030.tif #0001 Azm= 180.00'])
***** AFTER.TXT
    fill_value = 1e+20)
, 'PWDR Co_PCP_Act_d900-00030.fxye Bank 1', 'PWDR Co_PCP_Act_d900-00030.fxye Bank 1', 'PWDR Co_PCP_Act_d900-00030.fxye Bank 1'])

(['Comments'], <class 'list'>, ['Co_PCP_Act_d900-00030.tif #0001 Azm= 180.00'])
*****

***** before.txt
(['Sample Parameters', 'Scale'], <class 'list'>, [1.276313196832068, True])
(['Sample Parameters', 'Type'], <class 'str'>, 'Debye-Scherrer')
(['Sample Parameters', 'Absorption'], <class 'list'>, [0.0, False])
***** AFTER.TXT
(['Sample Parameters', 'Scale'], <class 'list'>, [1.276313196832068, True])
(['Sample Parameters', 'Type'], <class 'str'>, 'Bragg-Brentano')
(['Sample Parameters', 'Absorption'], <class 'list'>, [0.0, False])
*****
```

in Linux/Mac:

```
bht14: toby$ diff before.txt after.txt
103c103
< , 'PWDR Co_PCP_Act_d900-00030.fxye Bank 1', 'PWDR Co_PCP_Act_d900-00030.fxye Bank 1
-> '])
```

(continues on next page)

(continued from previous page)

```

---
> , 'PWDR Co_PCP_Act_d900-00030.fxye Bank 1', 'PWDR Co_PCP_Act_d900-00030.fxye Bank 1
→', 'PWDR Co_PCP_Act_d900-00030.fxye Bank 1'])
111c111
< (['Sample Parameters', 'Type'], <class 'str'>, 'Debye-Scherrer')
---
> (['Sample Parameters', 'Type'], <class 'str'>, 'Bragg-Brentano')

```

From this we can see there are two changes that took place. One is fairly obscure, where the histogram name is added to a list, which can be ignored, but the second change occurs in a straight-forward way and we discover that a simple call:

```
h.setHistEntryValue(['Sample Parameters', 'Type'], 'Bragg-Brentano')
```

can be used to change the histogram type.

3.7 Code Examples

Contents for Scripting Examples

- *Shortcut for Scripting Access*
- *Peak Fitting*
- *Pattern Simulation*
- *Simple Refinement*
- *Sequential Refinement*
- *Image Processing*
- *Image Calibration*
- *Optimized Image Integration*
- *Multicore Image Integration*
- *Histogram Export*
- *Automatic Background*

3.7.1 Shortcut for Scripting Access

As is seen in a number of the code examples, the location where GSAS-II is specified in the GSAS-II script using commands such as

```

import sys
sys.path.insert(0, '/Users/toby/software/G2/GSASII') # needed to "find" GSAS-II modules
import GSASIIscriptable as G2sc

```

An alternative to this is to “install” the current GSAS-II installation into the current Python interpreter. Once this has been done a single time, this single command can be used to replace the three commands listed above for all future uses of GSASIIscripting:

```
import G2script as G2sc
```

There are two ways this installation can be done. The most easy way is to invoke the “Install GSASIIscriptable shortcut” command in the GSAS-II GUI File menu. Alternatively it can be accomplished from within GSASIIscriptable using these commands:

```
import sys
sys.path.insert(0, '/Users/toby/software/G2/GSASII') # update this for your
↳ installation
import GSASIIscriptable as G2sc
G2sc.installScriptingShortcut()
```

An even simpler way to do this is from the command-line, from the GSAS-II directory. A full path for Python is only needed if the Python to be used with GSAS-II is not in the path.

```
terrier:toby> cd /home/beams1/TOBY/gsas2full/GSASII/
terrier:toby> /mypath/bin/python -c "import GSASIIscriptable as G2sc; G2sc.
↳ installScriptingShortcut()"
GSAS-II binary directory: /home/beams1/TOBY/gsas2full/GSASII/bindist
Created file /home/beams1/TOBY/gsas2full/lib/python3.10/site-packages/G2script.py
setting up GSASIIscriptable from /home/beams1/TOBY/gsas2full/GSASII
success creating /home/beams1/TOBY/gsas2full/lib/python3.10/site-packages/G2script.py
```

Note the shortcut only installs use of GSAS-II with the current Python installation. If more than one Python installation will be used with GSAS-II (for example because different conda environments are used), a shortcut should be created from within each Python environment.

If more than one GSAS-II installation will be used with a Python installation, a shortcut can only be used with one of them.

3.7.2 Peak Fitting

Peak refinement is performed with routines `add_peak()`, `set_peakFlags()` and `refine_peaks()`. Method `Export_peaks()` and properties `Peaks` and `PeakList` provide ways to access the results. Note that when peak parameters are refined with `refine_peaks()`, the background may also be refined. Use `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://advancedphotonsource.github.io/GSAS-II-tutorials/LabData/data/>).

```
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 = 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': 'chebyshev-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)
```

(continues on next page)

(continued from previous page)

```

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)
res = 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

```

3.7.3 Pattern Simulation

This shows two examples 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 first example uses a CW neutron instrument parameter file. The pattern is computed over a 2θ range of 5 to 120 degrees with 1000 points. The pattern and reflection list are written into files. 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(newgpx='PbSO4sim.gpx') # create a project
phase0 = gpx.add_phase(PathWrap("PbSO4-Wyckoff.cif"),
    phasename="PbSO4",fmthint='CIF') # add a phase to the project
# 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.,Npoints=1000,
    phases=gpx.phases(),scale=500000.)
gpx.do_refinements() # calculate pattern
gpx.save()
# save results
gpx.histogram(0).Export('PbSO4data','.csv','hist') # data
gpx.histogram(0).Export('PbSO4refl','.csv','refl') # reflections

```

This example uses bank#2 from a TOF neutron instrument parameter file. The pattern is computed over a TOF range of 14 to 35 milliseconds with the default of 2500 points. This uses the same CIF as in the example before, but the instrument is found in the [TOF-CW Joint Refinement Tutorial](#) tutorial.

```

import os,sys
sys.path.insert(0,'/Users/toby/software/G2/GSASII')
import GSASIIscriptable as G2sc
cifdir = "/Users/toby/software/G2/Tutorials/PythonScript/data"
datadir = "/Users/toby/software/G2/Tutorials/TOF-CW Joint Refinement/data"
gpx = G2sc.G2Project(newgpx='/tmp/PbSO4simT.gpx') # create a project
phase0 = gpx.add_phase(os.path.join(cifdir,"PbSO4-Wyckoff.cif"),
    phasename="PbSO4",fmthint='CIF') # add a phase to the project
hist1 = gpx.add_simulated_powder_histogram("PbSO4 simulation",
    os.path.join(datadir,"POWGEN_1066.instprm"),14.,35.,
    phases=gpx.phases(),ibank=2)

```

(continues on next page)

(continued from previous page)

```
gpx.do_refinements([{}])
gpx.save()
```

3.7.4 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.

```
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': 'chebyshev-1', 'refine': True,
                                'peaks': [[0, True]]}}}
gpx.set_refinement(pardict)
```

3.7.5 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:

```
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({'Scale': False})
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'])
```

(continues on next page)

(continued from previous page)

```

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()

```

3.7.6 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](#).

```

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(newgpx=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'))
pwrdrList = imlst[0].Integrate()
gpx.save()

```

This example shows a computation similar to what is done in tutorial [Area Detector Calibration with Multiple Distances](#)

```

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(newgpx='/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():
    img.setCalibrant('Si SRM640c')
    img.setVary('*', False)
    img.setVary(['det-X', 'det-Y', 'phi', 'tilt', 'wave'], True)
    img.setControls(defImgVals)
    img.Recalibrate()
    img.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()):
    img = images[key]
    c = img.getControls()
    print(c['distance'], c['wavelength'])

```

3.7.7 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 *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](#)).

```
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(newgpx='/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('*', 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(verbose=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(verbose=True)
# create image control files
for img in gpx.images():
    img.saveControls(os.path.splitext(img.name)[0]+'.imctrl')
gpx.save()
```

3.7.8 Optimized Image Integration

This example shows how image integration, including pixel masking of outliers, can be accomplished for a series of images where the calibration and other masking (Frame, Spots, etc) are the same for all images. This code has been optimized significantly so that computations are cached and are not repeated where possible. For one set of test data, processing of the first image takes ~5 seconds, but processing of subsequent takes on the order of 0.7 sec.

This code uses an `import G2script as G2sc` statement to access `GSASIIscriptable` without referencing the GSAS-II installation directory. This requires installing a reference to the GSAS-II location into the current a Python installation, which can be done from the GUI or with scripting commands, as is discussed in *Shortcut for Scripting Access*. Here function `installScriptingShortcut()` was used to create the `G2script` module. That code has been retained here as comments to show what was done.

To simplify use of this script, it is assumed that the script will be placed in the same directory as where the data files will be collected. Other customization is done in variables at the beginning of the code. Note that the beamline where these data are collected opens the output .tif files before the data collection for that image is complete. Once the .metadata file has been created, the image may be read.

Processing progresses as follows:

- Once a set of images are found, a project is created. This is never written and will be deleted after the images are processed.
- For each image file, routine `add_image()` is used to add image(s) from that file to the project. The .tif format can only hold one image, but others can have more than one.
- When the first image is processed, calibration and mask info is read; a number of computations are performed and cached.
- For subsequent images cached information is used.
- Pixel masking is performed in `GeneratePixelMask()` and the mask is saved into the image.
- Image integration is performed in `Integrate()`.
- Note that multiple powder patterns could be created from one image, so creation of data files is done in a loop with `Export()`.
- To reduce memory demands, cached versions of the Pixel map and the Image are deleted and the image file is moved to a separate directory so note that it has been processed.
- The project (.gpx file) is deleted and recreated periodically so that the memory footprint for this script does not grow.

The speed of this code will depend on many things, but the number of pixels in the image is primary, as well as CPU speed. With ~9 Mb images, I have seen average times in the range of 0.7 to 0.9 sec/image, after the first image is processed and the cached arrays are computed. With the Apple M1 chip the time is closer to 0.6 sec/image. There is also a possible tuning parameter that may change speed based on the speed of the CPU vs. memory constraints in variable `GSASIIscriptable.blkSize`. This value should be a power of two and defaults to 128. You might find that a larger or smaller value will improve performance for you.

```
import os,glob,time,shutil

#### Create G2script: do this once #####
#import sys
#sys.path.insert(0,'/Users/toby/software/G2/GSASII') # update with your install loc
#import GSASIIscriptable as G2sc
#G2sc.installScriptingShortcut()
#####

import G2script as G2sc
```

(continues on next page)

(continued from previous page)

```

G2sc.blkSize = 2**8 # computer-dependent tuning parameter
G2sc.SetPrintLevel('warn') # reduces output

cache = {} # place to save intermediate computations
# define location & names of files
dataLoc = os.path.abspath(os.path.split(__file__)[0]) # data in location of this file
PathWrap = lambda fil: os.path.join(dataLoc,fil) # convenience function for file paths
imgctrl = PathWrap('Si_ch3_d700-00000.imgctrl')
imgmask = PathWrap('Si_ch3_d700-00000.immask')
globPattern = PathWrap("*_d700-*.tif")

def wait_for_metadata(tifname):
    '''A .tif file is created before it can be read. Wait for the
    metadata file to be created before trying to read both.
    '''
    while not os.path.exists(tifname + '.metadata'):
        time.sleep(0.05)

# make a subfolder to store integrated images & integrated patterns
pathImg = os.path.join(dataLoc,'img')
if not os.path.exists(pathImg): os.mkdir(pathImg)
pathxye = os.path.join(dataLoc,'xye')
if not os.path.exists(pathxye): os.mkdir(pathxye)

while True: # Loop will never end, stop with ctrl+C
    tiflist = sorted(glob.glob(globPattern),key=lambda x: os.path.getctime(x)) # get
    ↳ images sorted by creation time, oldest 1st
    if not tiflist:
        time.sleep(0.1)
        continue
    gpx = G2sc.G2Project(newgpx=PathWrap('integration.gpx')) # temporary use
    for tifname in tiflist:
        starttime = time.time()
        wait_for_metadata(tifname)
        for img in gpx.add_image(tifname,fmtHint="TIF",cacheImage=True): # loop
            ↳unneeded for TIF (1 image/file)
                if not cache: # load & compute controls & 2theta values once
                    img.loadControls(imgctrl) # set controls/calibrations/masks
                    img.loadMasks(imgmask)
                    cache['Image Controls'] = img.getControls() # save controls & masks
                    ↳contents for quick reload
                    cache['Masks'] = img.getMasks()
                    cache['intMaskMap'] = img.IntMaskMap() # calc mask & TA arrays to
                    ↳save for integrations
                    cache['intTAmapping'] = img.IntThetaAzMap()
                    cache['FrameMask'] = img.MaskFrameMask() # calc Frame mask & T array
                    ↳to save for Pixel masking
                    cache['maskTmap'] = img.MaskThetaMap()
                else:
                    img.setControls(cache['Image Controls'])
                    img.setMasks(cache['Masks'],True) # True: reset threshold masks
                    img.GeneratePixelMask(esdMul=3,ThetaMap=cache['maskTmap'],FrameMask=cache[
                    ↳'FrameMask'])
                    for pwrdr in img.Integrate(MaskMap=cache['intMaskMap'],ThetaAzimMap=cache[
                    ↳'intTAmapping']):
                        pwrdr.Export(os.path.join(pathxye,os.path.split(tifname)[1]),'.xye')
            ↳# '.tif in name ignored

```

(continues on next page)

(continued from previous page)

```

        img.clearImageCache() # save some space
        img.clearPixelMask()
        shutil.move(tifname, pathImg) # move file after integration so that it_
↪is not searchable
        shutil.move(tifname + '.metadata', pathImg)
        print('*== processing complete, time=',time.time()-starttime,'sec\n')
    del gpx

```

3.7.9 Multicore Image Integration

The previous example (*Optimized Image Integration*) can be accelerated even further on a multicore computer using the following script. In this example, the image integration is moved to a function, *integrate_tif*, that accepts a filename to integrate. Note that with the multiprocessing module is used, the script will be read on each core that will be used, but only on the primary (controller) process will this `__name__ == '__main__'` be True. Thus the code following the if statement runs on the primary process. The primary process uses the `mp.Pool()` statement to create a set of secondary (worker) processes that are intended to run on other cores. The primary process locates .tif files, if the corresponding .tif.metadata is also found, both are moved to a separate directory where they will be processed in a secondary process. When the secondary process starts, the script is imported and then *integrate_tif* is called with the name of the image file from the primary process. The *integrate_tif* routine will initially have an empty cache and thus the code preceeded by “load & compute controls & 2theta values” will be computed once for every secondary process, which should be on an independent core. The size of the pool determines how many images will be processed simultaneously.

The script as given below uses the first argument on the command line to specify the number of cores to be used, where 0 is used to mean run *integrate_tif* directly rather than through a pool. This facilitates timing comparisons. This code seems to have a maximum speed using slightly less than the total number of available cores and does benefit partially from hyperthreading. A two- to three-fold speedup is seen with four cores and a six-fold speedup has been seen with 16 cores.

```

import os,sys,glob,time,shutil
scriptstart = time.time()

if len(sys.argv) >= 2:
    nodes = int(sys.argv[1])
else:
    nodes = 4

if nodes == 0:
    print('no multiprocessing')
else:
    print(f'multiprocessing with {nodes} cores')

import G2script as G2sc
G2sc.blkSize = 2**8 # computer-dependent tuning parameter
#G2sc.SetPrintLevel('warn')

cache = {} # place to save intermediate computations

# define location & names of files
dataLoc = '/dataserv/inttest' # images found here
globPattern = os.path.join(dataLoc,"*_d700-*.tif")
calibLoc = os.path.abspath(os.path.split(__file__)[0]) # calib in location of this_
↪file
imgctrl = os.path.join(calibLoc,'Si_ch3_d700-00000.imctrl')
imgmask = os.path.join(calibLoc,'Si_ch3_d700-00000.immask')
# locations to put processed files

```

(continues on next page)

(continued from previous page)

```
pathImg = os.path.join(dataLoc, 'img')
pathxye = os.path.join(dataLoc, 'xye')

def integrate_tif(tifname):
    starttime = time.time()
    gpx = G2sc.G2Project(newgpx='integration.gpx') # temporary use, not written
    for img in gpx.add_image(tifname, fmthint="TIF", cacheImage=True): # loop unneeded
        for TIF (1 image/file)
            img.setControl('pixelSize', [150, 150])
            if not cache: # load & compute controls & 2theta values once
                print('Initializing cache for', tifname)
                img.loadControls(imgctrl) # set controls/calibrations/masks
                img.loadMasks(imgmask)
                cache['Image Controls'] = img.getControls() # save file contents for
            quick reload
                cache['Masks'] = img.getMasks()
                cache['intMaskMap'] = img.IntMaskMap() # calc mask & TA arrays to save
            for integrations
                cache['intTAmapping'] = img.IntThetaAzMap()
                cache['FrameMask'] = img.MaskFrameMask() # calc Frame mask & T array to
            save for Pixel masking
                cache['maskTmap'] = img.MaskThetaMap()
            else:
                img.setControls(cache['Image Controls'])
                img.setMasks(cache['Masks'], True) # not using threshold masks
                img.GeneratePixelMask(esdMul=3, ThetaMap=cache['maskTmap'], FrameMask=cache[
            'FrameMask'])
                for pwrdr in img.Integrate(MaskMap=cache['intMaskMap'], ThetaAzimMap=cache[
            'intTAmapping']):
                    pwrdr.Export(os.path.join(pathxye, os.path.split(tifname)[1]), '.xye') #
            tif in name ignored
                img.clearImageCache() # save some space
                img.clearPixelMask()

    print(f'=== image processed, time={time.time()-starttime:.3f} sec\n')
    del gpx

if __name__ == '__main__':
    if nodes > 0: import multiprocessing as mp

    # make folder to store integrated images & integrated patterns if needed
    if not os.path.exists(pathImg): os.mkdir(pathImg)
    if not os.path.exists(pathxye): os.mkdir(pathxye)

    if nodes > 0: pool = mp.Pool(nodes)

    while True: # Loop will never end, stop with ctrl+C
        tiflist = sorted(glob.glob(globPattern), key=lambda x: os.path.getctime(x)) #
        get images sorted by creation time, oldest 1st
        if not tiflist:
            time.sleep(0.1)
            continue
        intlist = [] # list of images read to process
        for tifname in tiflist:
            if not os.path.exists(tifname + '.metadata'): continue
            shutil.move(tifname, pathImg) # move file before integration so that it
            is not found in another search
```

(continues on next page)

(continued from previous page)

```

        shutil.move(tifname + '.metadata', pathImg)
        intlist.append(os.path.join(pathImg, os.path.split(tifname)[1]))
    if nodes == 0:
        for newtifname in intlist: integrate_tif(newtifname)
    else:
        pool.map(integrate_tif, intlist)

    if nodes > 0: pool.close()
    print(f'Total elapsed time={time.time()-scriptstart:.3f} sec')

```

3.7.10 Histogram Export

This example shows how to export a series of histograms from a collection of .gpx (project) files. The Python `glob()` function is used to find all files matching a wildcard in the specified directory (`dataloc`). For each file there is a loop over histograms in that project and for each histogram `Export()` is called to write out the contents of that histogram as CSV (comma-separated variable) file that contains data positions, observed, computed and background intensities as well as weighting for each point and Q. Note that for the `Export` call, there is more than one choice of exporter that can write .csv extension files, so the export hint must be specified.

```

import os, sys, glob
sys.path.insert(0, '/Users/toby/software/G2/GSASII') # change this
import GSASIIscriptable as G2sc

dataloc = "/Users/toby/Scratch/" # where to find data
PathWrap = lambda fil: os.path.join(dataloc, fil) # EZ way 2 add dir to filename

for f in glob.glob(PathWrap('bkg*.gpx')): # put filename prefix here
    print(f)
    gpx = G2sc.G2Project(f)
    for i, h in enumerate(gpx.histograms()):
        hfil = os.path.splitext(f)[0]+'_'+str(i) # file to write
        print('\t', h.name, hfil+'.csv')
        h.Export(hfil, '.csv', 'histogram CSV')

```

3.7.11 Automatic Background

This example shows how to use the automatic background feature in GSAS-II to compute an approximate background and set fixed background points from that background. This approximately example follows that of the [Autobackground Tutorial](#). In this example, a new project is created and the data files from the tutorial are read. Note that scripting is not able to read files from inside a zip archive or use defaulted instrument parameters. The histograms are then processed in turn. The first step is to use `calc_autobkg` to compute the fixed background points. The refinement flag is then set for the Chebyshev polynomial terms and three background peaks are added with the width flag set for refinement. The first call to `fit_fixed_points()` will refine the three Chebyshev terms and the intensities of the three background peaks to fit the fixed background points. The refinement flags for the widths of the three background peaks are then set as well and the refinement is repeated. The location of the third background peaks is added and the refinement is repeated. Finally, the number of Chebyshev polynomial terms is increased to six and the refinement is repeated.

```

import os, glob
import G2script as G2sc
PathWrap = lambda fil: os.path.join('/tmp', fil)
gpx = G2sc.G2Project(newgpx=PathWrap('autobkg.gpx'))
for i in glob.glob(PathWrap('test_RampDown-*.xye')):

```

(continues on next page)

(continued from previous page)

```

hist = gpx.add_powder_histogram(i, PathWrap('testData.instprm'))
for hist in gpx.histograms('PWDR'):
    hist.calc_autobkg(logLam=3.5)
    hist.set_refinements({"Background": {"no. coeffs": 3, "refine": True}})
    for pk in [2.4, 3.1, 4.75]:
        hist.add_back_peak(pk, 1000, 1000, 0, [False, True, False, False])
    hist.fit_fixed_points()
    for i in [0, 1, 2]: hist.ref_back_peak(i, [False, True, True, False])
    hist.fit_fixed_points()
    hist.ref_back_peak(2, [True, True, True, False])
    hist.fit_fixed_points()
    hist.set_refinements({"Background": {"no. coeffs": 6, "refine": True}})
    hist.fit_fixed_points()
gpx.save()

```

3.8 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 mode of accessing GSAS-II scripting does not appear to get much use and is no longer being developed. Please do communicate to the developers if keeping this mode of access would be of value in your work.

To use the command-line mode is done with a command like this:

```
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.

3.8.1 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](#).

3.9 API: Complete Documentation

Classes and routines defined in *GSASIIscriptable* follow. 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. Allows many atom properties to be access and changed. See the *Atom Records description* for the details on what information is contained in an atom record.

Scripts should not try to create a *G2AtomRecord* object directly as these objects are created via access from a *G2Phase* object.

Example showing some uses of *G2AtomRecord* methods:

```
>>> atom = some_phase.atom("O3")
>>> # We can access the underlying data structure (a list):
>>> atom.data
['O3', 'O-2', '', ... ]
>>> # We can also use wrapper accessors to get or change atom info:
>>> atom.coordinates
(0.33, 0.15, 0.5)
>>> atom.coordinates = [1/3, .1, 1/2]
>>> atom.coordinates
(0.3333333333333333, 0.1, 0.5)
>>> atom.refinement_flags
'FX'
>>> atom.ranId
4615973324315876477
>>> atom.occupancy
1.0
```

property ADP

Get or set the associated atom's Uiso or Uaniso value(s). Use as `x = atom.ADP` to obtain the value(s) and `atom.ADP = x` to set the value(s). For isotropic atoms a single float value is returned (or used to set). For anisotropic atoms a list of six values is used.

See also:

`adp_flag()` `uiso()`

property adp_flag

Get the associated atom's iso/aniso setting. The value will be 'T' or 'A'. No API provision is offered to change this.

property coordinates

Get or set the associated atom's coordinates. Use as `x = atom.coordinates` to obtain a tuple with the three (x,y,z) values and `atom.coordinates = (x,y,z)` to set the values.

Changes needed to adapt for changes in site symmetry have not yet been implemented:

property element

Parses element symbol from the atom type symbol for the atom associated with the current object.

See also:

`type()`

property label

Get the associated atom's label. Use as `x = atom.label` to obtain the value and `atom.label = x` to set the value.

property mult

Get the associated atom's multiplicity value. Should not be changed by user.

property occupancy

Get or set the associated atom's site fraction. Use as `x = atom.occupancy` to obtain the value and `atom.occupancy = x` to set the value.

property ranId

Get the associated atom's Random Id number. Don't change this.

property refinement_flags

Get or set refinement flags for the associated atom. Use as `x = atom.refinement_flags` to obtain the flags and `atom.refinement_flags = "XU"` (etc) to set the value.

property type

Get or set the associated atom's type. Call as a variable (`x = atom.type`) to obtain the value or use `atom.type = x` to change the type. It is the user's responsibility to make sure that the atom type is valid; no checking is done here.

See also:

`element()`

property uiso

A synonym for `ADP()` to be used for Isotropic atoms. Get or set the associated atom's Uiso value. Use as `x = atom.uiso` to obtain the value and `atom.uiso = x` to set the value. A single float value is returned or used to set.

See also:

`adp_flag()` `ADP()`

class GSASIIscriptable.G2Image (data, name, proj, image=None)

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 should not try to create a `G2Image` object directly as `G2Image.__init__()` should be invoked from inside `G2Project`.

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*.
- `G2Image.image`: optionally contains a cached the image to save time in reloading. This is saved only when `cacheImage=True` is specified when `G2Project.add_image()` is called.

Example use of `G2Image`:

```
>>> gpx = G2sc.G2Project(newgpx='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)
>>> pwrList = imlst[0].Integrate()
```

More detailed image processing examples are shown in the *Image Processing* section of this chapter.

```
ControlList = {'bool': ['setRings', 'setDefault', 'centerAzm',
'fullIntegrate', 'DetDepthRef', 'showLines'], 'dict': ['varyList'],
'float': ['cutoff', 'setdist', 'wavelength', 'Flat Bkg', 'azmthOff',
'tilt', 'calibdmin', 'rotation', 'distance', 'DetDepth'], 'int':
['calibskip', 'pixLimit', 'edgemin', 'outChannels', 'outAzimuths'],
'list': ['GonioAngles', 'IOtth', 'LRazimuth', 'Oblique', 'PolaVal',
'SampleAbs', 'center', 'ellipses', 'linescan', '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.

GeneratePixelMask (*esdMul=3.0, tmin=0.0, tmax=180.0, FrameMask=None, ThetaMap=None, fastmode=True, combineMasks=False*)

Generate a Pixel mask with True at the location of pixels that are statistical outliers (in comparison with others with the same 2theta value.) The process for this is that a median is computed for pixels within a small 2theta window and then the median difference is computed from magnitude of the difference for those pixels from that median. The medians are used for this rather than a standard deviation as the computation used here is less sensitive to outliers. (See `GSASIIimage.AutoPixelMask()` and `scipy.stats.median_abs_deviation()` for more details.)

Mask is placed into the `G2image` object where it will be accessed during integration. Note that this increases the .gpx file size significantly; use `clearPixelMask()` to delete this if it need not be saved.

This code is based on `GSASIIimage.FastAutoPixelMask()` but has been modified to recycle expensive computations where possible.

Parameters

- **esdMul** (*float*) – Significance threshold applied to remove outliers. Default is 3. The larger this number, the fewer “glitches” that will be removed.

- **ttmin** (*float*) – A lower 2theta limit to be used for pixel searching. Pixels outside this region may be considered for establishing the medians, but only pixels with 2theta >= ttmin are masked. Default is 0.
- **ttmax** (*float*) – An upper 2theta limit to be used for pixel searching. Pixels outside this region may be considered for establishing the medians, but only pixels with 2theta < ttmax are masked. Default is 180.
- **FrameMask** (*np.array*) – An optional precomputed Frame mask (from `Mask-FrameMask()`). Compute this once for a series of similar images to reduce computational time.
- **ThetaMap** (*np.array*) – An optional precomputed array that defines 2theta for each pixel, computed in `MaskThetaMap()`. Compute this once for a series of similar images to reduce computational time.
- **fastmode** (*bool*) – If True (default) fast Pixel map searching is done if the C module is available. If the module is not available or this is False, the pure Python implementation is used. It is not clear why False is ever needed.
- **combineMasks** (*bool*) – When True, the current Pixel mask will be combined with any previous Pixel map. If False (the default), the Pixel map from the current search will replace any previous ones. The reason for use of this as True would be where different esdMul values are used for different regions of the image (by setting ttmin & ttmax) so that the outlier level can be tuned by combining different searches.

IntMaskMap()

Computes a series of masking arrays for the current image (based on mask input, but not calibration parameters or the image intensities). See `GSASIIimage.MakeMaskMap()` for more details. The output from this is optionally supplied as input to `Integrate()`.

Note this is not the same as pixel mask searching (`GeneratePixelMask()`).

IntThetaAzMap()

Computes the set of blocked arrays for 2theta-azimuth mapping from the controls settings of the current image for image integration. The output from this is optionally supplied as input to `Integrate()`. Note that if not supplied, image integration will compute this information as it is needed, but this is a relatively slow computation so time can be saved by caching and reusing this computation for other images that have the same calibration parameters as the current image.

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 `IntMaskMap()`
- **ThetaAzimMap** (*list*) – from `G2Image.IntThetaAzMap()`

Returns

a list of created histogram (`G2PwdrData`) objects.

MaskFrameMask ()

Computes a Frame mask from map input for the current image to be used for a pixel mask computation in `GeneratePixelMask()`. This is optional, as if not supplied, mask computation will compute this, but this is a relatively slow computation and the results computed here can be reused for other images that have the same calibration parameters.

MaskThetaMap ()

Computes the theta mapping matrix from the controls settings of the current image to be used for pixel mask computation in `GeneratePixelMask()`. This is optional, as if not supplied, mask computation will compute this, but this is a relatively slow computation and the results computed here can be reused for other images that have the same calibration parameters.

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.

TestFastPixelMask ()

Tests to see if the fast (C) code for pixel masking is installed.

Returns

A value of True is returned if fast pixel masking is available. Otherwise False is returned.

clearImageCache ()

Clears a cached image, if one is present

clearPixelMask ()

Removes a pixel map from an image, to reduce the .gpx file size & memory use

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:

```
>>> 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 calibration of 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)

initMasks ()

Initialize Masks, including resetting the Thresholds values

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.

saveControls (filename)

write current controls values to a .imctrl file

Parameters

filename (*str*) – specifies a file to write, which should end with .imctrl

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.

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*.

setControlFile (*typ, imageRef, mult=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 that is used for fitting calibration parameters. 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 bool.

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()*. Scripts should not try to create a *G2PDF* object directly.

Example use of *G2PDF*:

```
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),...) 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 (de-

fault), 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

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.

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*.

Scripts should not try to create a *G2Phase* object directly as `G2Phase.__init__()` should be invoked from inside *G2Project*.

Author: Jackson O'Donnell (jacksonhodonnell .at. gmail.com)

HAPvalue (*param=None, new Value=None, targethistlist='all'*)

Retrieves or sets individual HAP parameters for one histogram or multiple histograms.

Parameters

- **param** (*str*) – is a parameter name, which can be 'Scale' (phase fraction), 'Use', 'Extinction' or 'LeBail'. If not specified or invalid an exception is generated showing the list of

valid parameters. At present, these HAP parameters cannot be access with this function: 'Pref.Ori.', 'Size', 'Mustrain', 'HStrain', 'Babinet'. On request this might be addressed in the future. Some of these values can be set via `G2Phase.set_HAP_refinements()`.

- **newValue** – the value to use when setting the HAP parameter for the appropriate histogram(s). Will be converted to the proper type or an exception will be generated if not possible. If not specified, and only one histogram is selected, the value is retrieved and returned.
- **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.

targethistlist must correspond to a single histogram if a value is to be returned (when argument newValue is not specified).

Returns

the value of the parameter, when argument newValue is not specified.

See also:

`set_HAP_refinements()`

Example:

```
val = ph0.HAPvalue('Scale')
val = ph0.HAPvalue('Scale',targethistlist=[0])
ph0.HAPvalue('Scale',2.5)
```

The first command returns the phase fraction if only one histogram is associated with the current phase, or raises an exception. The second command returns the phase fraction from the first histogram associated with the current phase. The third command sets the phase fraction for all histograms associated with the current phase.

addDistRestraint (*origin, target, bond, factor=1.1, ESD=0.01*)

Adds bond distance restraint(s) for the selected phase

This works by search for interatomic distances between atoms in the origin list and the target list (the two lists may be the same but most frequently will not) with a length between bond/factor and bond*factor. If a distance is found in that range, it is added to the restraints if it was not already found.

Parameters

- **origin** (*list*) – a list of atoms, each atom may be an atom object, an index or an atom label
- **target** (*list*) – a list of atoms, each atom may be an atom object, an index or an atom label
- **bond** (*float*) – the target bond length in Å for the located atom
- **factor** (*float*) – a tolerance factor used when searching for bonds (defaults to 1.1)
- **ESD** (*float*) – the uncertainty for the bond (defaults to 0.01)

Returns

returns the number of new restraints that are found

As an example:

```
gpx = G2sc.G2Project('restr.gpx')
ph = gpx.phases()[0]
ph.clearDistRestraint()
origin = [a for a in ph.atoms() if a.element == 'Si']
target = [i for i, a in enumerate(ph.atoms()) if a.element == 'O']
c = ph.addDistRestraint(origin, target, 1.64)
print(c, 'new restraints found')
ph.setDistRestraintWeight(1000)
gpx.save('restr-mod.gpx')
```

This example locates the first phase in a project file, clears any previous restraints. Then it places restraints on bonds between Si and O atoms at 1.64 Å. Each restraint is weighted 1000 times in comparison to (obs-calc)/sigma for a data point. To show how atom selection can work, the origin atoms are identified here by atom object while the target atoms are identified by atom index. The methods are interchangeable. If atom labels are unique, then:

```
origin = [a.label for a in ph.atoms() if a.element == 'Si']
```

would also work identically.

add_atom (*x*, *y*, *z*, *element*, *lbl*, *occ*=1.0, *uiso*=0.01)

Adds an atom to the current phase

Parameters

- **x** (*float*) – atom fractional x coordinate
- **y** (*float*) – atom fractional y coordinate
- **z** (*float*) – atom fractional z coordinate
- **element** (*str*) – an element symbol (capitalization is ignored). Optionally add a valence (as in Ba+2)
- **lbl** (*str*) – A label for this atom
- **occ** (*float*) – A fractional occupancy for this atom (defaults to 1).
- **uiso** (*float*) – A Uiso value for this atom (defaults to 0.01).

Returns

the *G2AtomRecord* atom object for the new atom

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.

atoms ()

Returns a list of atoms present in the current phase.

Returns

A list of *G2AtomRecord* objects.

See also:

atom() *G2AtomRecord*

clearDistRestraint ()

Deletes any previously defined bond distance restraint(s) for the selected phase

See also:

G2Phase.addDistRestraint ()

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 *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

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.

property 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:

```
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.

property 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`

getHAPentryList (*histname=None*, *keyname=""*)

Returns a dict with HAP values. Optionally a histogram may be selected.

Parameters

- **histname** – 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. If no histogram is specified, all histograms are selected.
- **keyname** (*str*) – an optional string. When supplied only entries where at least one key contains the specified string are reported. Case is ignored, so 'sg' will find entries where one of the keys is 'SGdata', etc.

Returns

a set of HAP dict keys.

Example:

```
>>> p.getHAPentryList(0, 'Scale')
[(['PWDR test Bank 1', 'Scale'], list, [1.0, False])]
```

See also:

getHAPentryValue() *setHAPentryValue()*

getHAPentryValue (*keylist*)

Returns the HAP value associated with a list of keys. Where the value returned is a list, it may be used as the target of an assignment (as in `getHAPentryValue(...)[...] = val`) to set a value inside a list.

Parameters

keylist (*list*) – a list of dict keys, typically as returned by *getHAPentryList()*. Note the first entry is a histogram name. Example: `['PWDR hist1.fxye Bank 1', 'Scale']`

Returns

HAP value

Example:

```
>>> sclEnt = p.getHAPentryList(0, 'Scale')[0]
>>> sclEnt
[(['PWDR test Bank 1', 'Scale'], list, [1.0, False])]
>>> p.getHAPentryValue(sclEnt[0])
[1.0, False]
```

(continues on next page)

(continued from previous page)

```
>>> p.getHAPentryValue(sclEnt[0])[1] = True
>>> p.getHAPentryValue(sclEnt[0])
[1.0, True]
```

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

getPhaseEntryList (*keyname=""*)

Returns a dict with control values.

Parameters

keyname (*str*) – an optional string. When supplied only entries where at least one key contains the specified string are reported. Case is ignored, so 'sg' will find entries where one of the keys is 'SGdata', etc.

Returns

a set of phase dict keys. Note that HAP items, while technically part of the phase entries, are not included.

See *getHAPentryList()* for a related example.

See also:

getPhaseEntryValue() *setPhaseEntryValue()*

getPhaseEntryValue (*keylist*)

Returns the value associated with a list of keys. Where the value returned is a list, it may be used as the target of an assignment (as in *getPhaseEntryValue(...)[...] = val*) to set a value inside a list.

Parameters

keylist (*list*) – a list of dict keys, typically as returned by *getPhaseEntryList()*.

Returns

a phase setting; may be a int, float, bool, list,...

See *getHAPentryValue()* for a related example.

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 Å. Uses GSASIImath.XScattDen which seems to be off by an order of magnitude, which has been corrected here.

setDistRestraintWeight(factor=1)

Sets the weight for the bond distance restraint(s) to factor

Parameters

factor (*float*) – the weighting factor for this phase’s restraints. Defaults to 1 but this value is typically much larger (10**2 to 10**4)

See also:

`G2Phase.addDistRestraint()`

setHAPentryValue(keylist, newvalue)

Sets an HAP value associated with a list of keys.

Parameters

- **keylist** (*list*) – a list of dict keys, typically as returned by `getHAPentryList()`. Note the first entry is a histogram name. Example: ['PWDR hist1.fxye Bank 1', 'Scale']
- **newvalue** – a new value for the HAP setting. The type must be the same as the initial value, but if the value is a container (list, tuple, np.array,...) the elements inside are not checked.

Example:

```
>>> sclEnt = p.getHAPentryList(0, 'Scale')[0]
>>> p.getHAPentryValue(sclEnt[0])
[1.0, False]
>>> p.setHAPentryValue(sclEnt[0], (1, True))
GSASIIscriptable.G2ScriptException: setHAPentryValue error: types do not
agree for keys ['PWDR test.fxye Bank 1', 'Scale']
>>> p.setHAPentryValue(sclEnt[0], [1, True])
>>> p.getHAPentryValue(sclEnt[0])
[1, True]
```

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`.

setPhaseEntryValue (*keylist*, *newvalue*)

Sets a phase control value associated with a list of keys.

Parameters

- **keylist** (*list*) – a list of dict keys, typically as returned by `getPhaseEntryList()`.
- **newvalue** – a new value for the phase setting. The type must be the same as the initial value, but if the value is a container (list, tuple, np.array,...) the elements inside are not checked.

See `setHAPentryValue()` for a related example.

setSampleProfile (*histname*, *parmType*, *mode*, *val1*, *val2*=None, *axis*=None, *LGmix*=None)

Sets sample broadening parameters for a histogram associated with the current phase. This currently supports isotropic and uniaxial broadening modes only.

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.
- **parmType** (*str*) – should be 'size' or 'microstrain' (can be abbreviated to 's' or 'm')
- **mode** (*str*) – should be 'isotropic' or 'uniaxial' (can be abbreviated to 'i' or 'u')
- **val1** (*float*) – value for isotropic size (in μm) or microstrain (unitless, $\Delta Q/Q \times 10^6$) or the equatorial value in the uniaxial case
- **val2** (*float*) – value for axial size (in μm) or axial microstrain (unitless, $\Delta Q/Q \times 10^6$) in uniaxial case; not used for isotropic
- **axis** (*list*) – tuple or list with three values indicating the preferred direction for uniaxial broadening; not used for isotropic
- **LGmix** (*float*) – value for broadening type (1=Lorentzian, 0=Gaussian or a value between 0 and 1. Default value (None) is ignored.

Examples:

```
phase0.setSampleProfile(0,'size','iso',1.2)
phase0.setSampleProfile(0,'micro','isotropic',1234)
phase0.setSampleProfile(0,'m','u',1234,4567,[1,1,1],.5)
phase0.setSampleProfile(0,'s','uni',1.2,2.3,[0,0,1])
```

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

set_refinements (*refs*)

Sets the phase refinement parameter ‘key’ to the specification ‘value’

Parameters

refs (*dict*) – A dictionary of the parameters to be set. See the [Phase parameters](#) table for a description of this dictionary.

Returns

None

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 file named by gpxfile and then when saved will be written to the file named by newgpx.
- **filename** (*str*) – To be deprecated. Serves the same function as newgpx, which has a somewhat more clear name. (Do not specify both newgpx and filename).

There are two ways to initialize this object:

```
>>> # Load an existing project file
>>> proj = G2Project('filename.gpx')
```

```
>>> # Create a new project
>>> proj = G2Project(newgpx='new_file.gpx')
```

Histograms can be accessed easily.

```
>>> # By name
>>> hist = proj.histogram('PWDR my-histogram-name')
```

```
>>> # Or by index
>>> hist = proj.histogram(0)
>>> assert hist.id == 0
```

```
>>> # Or by random id
>>> assert hist == proj.histogram(hist.ranId)
```

Phases can be accessed the same way.

```
>>> phase = proj.phase('name of phase')
```

New data can also be loaded via `add_phase()` and `add_powder_histogram()`.

```
>>> 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 at the project level as well as described in `set_refinement()`, `iter_refinements()` and `do_refinements()`.

ComputeWorstFit()

Computes the worst-fit parameters in a model.

Returns

(keys, derivCalcs, varyList) where:

- **keys** is a list of parameter names where the names are ordered such that first entry in the list will produce the largest change in the fit if refined and the last entry will have the smallest change;
- **derivCalcs** is a dict where the key is a variable name and the value is a list with three partial derivative values for $d(\text{Chi}^2)/d(\text{var})$ where the derivatives are computed for values $v-d$ to v ; $v-d$ to $v+d$; v to $v+d$ where v is the current value for the variable and d is a small delta value chosen for that variable type;
- **varyList** is a list of the parameters that are currently set to be varied.

add_EqnConstr (*total*, *varlist*, *multlist*=[], *reloadIdx*=True)

Set a constraint equation on a list of variables.

Note that this will cause the project to be saved if not already done so. It will always save the .gpx file before creating a constraint if `reloadIdx` is True.

Parameters

- **total** (*float*) – A value that the constraint must equal
- **varlist** (*list*) – A list of variables to use in the equation. Each value in the list may be one of the following three items: (A) a `GSASIIobj.G2VarObj` object, (B) a variable name (str), or (C) a list/tuple of arguments for `make_var_obj()`.

- **multlist** (*list*) – a list of multipliers for each variable in varlist. If there are fewer values than supplied for varlist then missing values will be set to 1. The default is [] which means that all multipliers are 1.
- **reloadIdx** (*bool*) – If True (default) the .gpx file will be saved and indexed prior to use. This is essential if atoms, phases or histograms have been added to the project.

Example:

```
gpx.add_EqnConstr(1.0, ('0::Ax:0', '0::Ax:1'), [1, 1])
```

add_EquivConstr (*varlist*, *multlist*=[], *reloadIdx*=True)

Set a equivalence on a list of variables.

Note that this will cause the project to be saved if not already done so. It will always save the .gpx file before creating a constraint if reloadIdx is True.

Parameters

- **varlist** (*list*) – A list of variables to make equivalent to the first item in the list. Each value in the list may be one of the following three items: (A) a *GSASIIobj.G2VarObj* object, (B) a variable name (str), or (C) a list/tuple of arguments for *make_var_obj()*.
- **multlist** (*list*) – a list of multipliers for each variable in varlist. If there are fewer values than supplied for varlist then missing values will be set to 1. The default is [] which means that all multipliers are 1.
- **reloadIdx** (*bool*) – If True (default) the .gpx file will be saved and indexed prior to use. This is essential if atoms, phases or histograms have been added to the project.

Examples:

```
gpx.add_EquivConstr(('0::AUiso:0', '0::AUiso:1', '0::AUiso:2'))
gpx.add_EquivConstr(('0::dAx:0', '0::dAx:1'), [1, -1])
```

add_HoldConstr (*varlist*, *reloadIdx*=True)

Set a hold constraint on a list of variables.

Note that this will cause the project to be saved if not already done so. It will always save the .gpx file before creating constraint(s) if reloadIdx is True.

Parameters

- **varlist** (*list*) – A list of variables to hold. Each value in the list may be one of the following three items: (A) a *GSASIIobj.G2VarObj* object, (B) a variable name (str), or (C) a list/tuple of arguments for *make_var_obj()*.
- **reloadIdx** (*bool*) – If True (default) the .gpx file will be saved and indexed prior to use. This is essential if atoms, phases or histograms have been added to the project.

Example:

```
gpx.add_HoldConstr(('0::A4', '0:1:D12', ':0:Lam'))
```

add_NewVarConstr (*varlist*, *multlist*=[], *name*=None, *vary*=False, *reloadIdx*=True)

Set a new-variable constraint from a list of variables to create a new parameter from two or more predefined parameters.

Note that this will cause the project to be saved, if not already done so. It will always save the .gpx file before creating a constraint if reloadIdx is True.

Parameters

- **varlist** (*list*) – A list of variables to use in the expression. Each value in the list may be one of the following three items: (A) a *GSASIIobj.G2VarObj* object, (B) a variable name (*str*), or (C) a list/tuple of arguments for *make_var_obj()*.
- **multlist** (*list*) – a list of multipliers for each variable in varlist. If there are fewer values than supplied for varlist then missing values will be set to 1. The default is [] which means that all multipliers are 1.
- **name** (*str*) – An optional string to be supplied as a name for this new parameter.
- **vary** (*bool*) – Determines if the new variable should be flagged to be refined.
- **reloadIdx** (*bool*) – If True (default) the .gpx file will be saved and indexed prior to use. This is essential if atoms, phases or histograms have been added to the project.

Examples:

```
gpx.add_NewVarConstr('0::AFrac:0', '0::AFrac:1'), [0.5, 0.5], 'avg', True)
gpx.add_NewVarConstr('0::AFrac:0', '0::AFrac:1'), [1, -1], 'diff', False, False)
```

The example above is a way to treat two variables that are closely correlated. The first variable, labeled as avg, allows the two variables to refine in tandem while the second variable (diff) tracks their difference. In the initial stages of refinement only avg would be refined, but in the final stages, it might be possible to refine diff. The second False value in the second example prevents the .gpx file from being saved.

add_PDF (*prnfile, 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 to the project.

Parameters

- **cons_scope** (*str*) – should be one of “Hist”, “Phase”, “HAP”, or “Global”.
- **constr** (*list*) – a constraint coded with *GSASIIobj.G2VarObj* objects as described in the *constraint definition descriptions*.

WARNING this function does not check the constraint is well-constructed. Please use *G2Project.add_HoldConstr()* or *G2Project.add_EquivConstr()* (etc.) instead, unless you are really certain you know what you are doing.

add_image (*imagefile, fmthint=None, defaultImage=None, indexList=None, cacheImage=False*)

Load an image into a project

Parameters

- **imagefile** (*str*) – The image file to read, a filename.
- **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).

- **defaultImage** (*str*) – The name of an image to use as a default for setting parameters for the image file to read.
- **indexList** (*list*) – specifies the image numbers (counting from zero) to be used from the file when a file has multiple images. A value of `[0, 2, 3]` will cause the only first, third and fourth images in the file to be included in the project.
- **cacheImage** (*bool*) – When True, the image is cached to save in rereading it later. Default is False (no caching).

Returns

a list of *G2Image* object(s) for the added image(s)

add_phase (*phasefile=None, phasename=None, histograms=[], fmthint=None, mag=False, spacegroup='P 1', cell=None*)

Loads a phase into the project, usually from a .cif file

Parameters

- **phasefile** (*str*) – The CIF file (or other file type, see *fmthint*) that the phase will be read from. May be left as None (the default) if the phase will be constructed a step at a time.
- **phasename** (*str*) – The name of the new phase, or None for the default. A phasename must be specified when a phasefile is not.
- **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).
- **mag** (*bool*) – Set to True to read a magCIF
- **spacegroup** (*str*) – The space group name as a string. The space group must follow the naming rules used in `GSASIIspc.SpcGroup()`. Defaults to 'P 1'. Note that this is only used when phasefile is None.
- **cell** (*list*) – a list with six unit cell constants (a, b, c, alpha, beta and gamma in Angstrom/degrees).

Returns

A *G2Phase* object representing the new phase.

add_powder_histogram (*datafile, iparams=None, phases=[], fmthint=None, databank=None, instbank=None, multiple=False*)

Loads a powder data histogram or multiple powder histograms into the project.

Note that the data type (x-ray/CW neutron/TOF) for the histogram will be set from the instrument parameter file. The instrument geometry is assumed to be Debye-Scherrer except for dual-wavelength x-ray, where Bragg-Brentano is assumed.

Parameters

- **datafile** (*str*) – A filename with the powder data file to read. Note that in unix fashion, “~” can be used to indicate the home directory (e.g. `~/G2data/data.fxyc`).

- **iparams** (*str*) – A filename for an instrument parameters file, or a pair of instrument parameter dicts from `load_iprms()`. This may be omitted for readers that provide the instrument parameters in the file. (Only a few importers do this.)
- **phases** (*list*) – A list of phases to link to the new histogram, phases can be references by object, name, rId 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 the first dataset in the file is read. When multiple is True, optionally a list of dataset numbers can be supplied here.
- **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.
- **multiple** (*bool*) – If False (default) only one dataset is read, but if specified as True, all selected banks of data (see databank) are read in.

Returns

A `G2PwdrData` object representing the histogram, or if multiple is True, a list of `G2PwdrData` objects is returned.

add_simulated_powder_histogram (*histname, iparams, Tmin, Tmax, Tstep=None, wavelength=None, scale=None, phases=[], ibank=None, Npoints=None*)

Create a simulated powder data histogram for 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 (millisec) for dataset to be simulated
- **Tmax** (*float*) – Maximum 2theta or TOF (millisec) for dataset to be simulated
- **Tstep** (*float*) – Step size in 2theta or deltaT/T (TOF) for simulated dataset. Default is to compute this from Npoints.
- **wavelength** (*float*) – Wavelength for CW instruments, overriding the value in the instrument parameters file if specified. For single-wavelength histograms, this should be a single float value, for K alpha 1,2 histograms, this should be a list or tuple with two values.
- **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 neutrons. The default, None, provides a scale of 1 for x-rays, 10,000 for CW neutrons and 100,000 for TOF.
- **phases** (*list*) – Phases to link to the new histogram. Use `proj.phases()` to link to all defined phases.

- **ibank** (*int*) – provides a bank number for the instrument parameter file. The default is None, corresponding to load the first bank.
- **Npoints** (*int*) – the number of data points to be used for computing the diffraction pattern. Defaults as None, which sets this to 2500. Do not specify both Npoints and Tstep. Due to roundoff the actual number of points used may differ by +-1 from Npoints. Must be below 25,000.

Returns

A *G2PwdrData* object representing the histogram

add_single_histogram (*datafile*, *phase=None*, *fmthint=None*)

Loads a powder data histogram or multiple powder histograms into the project.

Parameters

- **datafile** (*str*) – A filename with the single crystal data file to read. Note that in unix fashion, “~” can be used to indicate the home directory (e.g. ~/G2data/data.hkl).
- **phases** – A phase to link to the new histogram. A phase can be referenced by object, name, rId or number. If not specified, no phase will be linked.
- **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, an error will be generated, as the file format will not distinguish well between different data types.

Returns

A *G2Single* 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. If not specified, the default value is [{}], which performs a single refinement step is performed with the current refinement settings.
- **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:

```
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_Constraints (*ctype*)

Returns a list of constraints of the type selected.

Parameters

ctype (*str*) – one of the following keywords: ‘Hist’, ‘HAP’, ‘Phase’, ‘Global’

Returns

a list of constraints, see the [constraint definition descriptions](#). Note that if this list is changed (for example by deleting elements or by changing them) the constraints in the project are changed.

get_Controls (*control*, *variable=None*)

Return project controls settings

Parameters

- **control** (*str*) – the item to be returned. See below for allowed values.
- **variable** (*str*) – a variable name as a str or (as a [GSASIIObj.G2VarObj](#) object). Used only with control set to “parmMin” or “parmMax”.

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.
- **parmMin & parmMax**: retrieves a maximum or minimum value for a refined parameter. Note that variable will be a GSAS-II variable name, optionally with * specified for a histogram or atom number. Return value will be a float. (See [Parameter Limits](#) description.)
- 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

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:

```
sig = np.sqrt(np.diag(covMatrix))
xvar = np.outer(sig,np.ones_like(sig))
covArray = np.divide(np.divide(covMatrix,xvar),xvar.T)
```

get_Frozen (*histogram=None*)

Gets a list of Frozen variables. (See [Parameter Limits](#) description.) Note that use of this will cause the project to be saved if not already done so.

Parameters

histogram – A reference to a histogram, which can be reference by object, name, or number. Used for sequential fits only. If left as the default (None) for a sequential fit, all Frozen variables in all histograms are returned.

Returns

a list containing variable names, as str values

get_ParmList ()

Returns a list of all the parameters defined in the last refinement cycle

Returns

a list of parameters or None if no refinement has been performed.

get_Variable (var)

Returns the value and standard uncertainty (esd) for a variable parameters, as defined in the last refinement cycle

Parameters

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 ()

Returns a list of the refined variables in the last refinement cycle

Returns

a list of variables or None if no refinement has been performed.

histType (histname)

Returns the type for histogram object associated with histname, or None if it does not exist.

Parameters

histname – The name of the histogram (str), or ranId or (for powder) the histogram index.

Returns

‘PWDR’ for a Powder histogram, ‘HKLF’ for a single crystal histogram, or None if the histogram does not exist

See also:

histogram()

histogram (histname)

Returns the histogram object associated with histname, or None if it does not exist.

Parameters

histname – The name of the histogram (str), or ranId or (for powder) the histogram index.

Returns

A *G2PwdrData* object, or *G2Single* object, or None if the histogram does not exist

See also:

histograms() phase() phases()

histograms (typ=None)

Return a list of all histograms, as *G2PwdrData* objects

For now this only finds Powder/Single Xtal histograms, since that is all that is currently implemented in this module.

Parameters

typ (*ste*) – The prefix (type) the histogram such as ‘PWDR’ for powder or ‘HKLF’ for single crystal. If None (the default) all known histograms types are found.

Returns

a list of objects

See also:

histogram() *phase()* *phases()*

hold_many (*vars*, *ctype*)

Apply holds for all the variables in *vars*, for constraint of a given type. This routine has been superceeded by *add_Hold()*

Parameters

- **vars** (*list*) – A list of variables to hold. Each may be a *GSASIIobj.G2VarObj* object, a variable name (str), or a list/tuple of arguments for *make_var_obj()*.
- **ctype** (*str*) – A string constraint type specifier, passed directly to *add_constraint_raw()* as *consType*. Should be one of “Hist”, “Phase”, or “HAP” (“Global” not implemented).

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).

```
>>> 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.

Note that this will cause the project to be saved if not already done so.

pdf (*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:

pdfs() *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.

ReturnsA 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

ReturnsA *G2SeqRefRes* object or None if not present**set_Controls** (*control*, *value*, *variable=None*)

Set project controls.

Note that use of this with control set to parmMin or parmMax will cause the project to be saved if not already done so.

Parameters

- **control** (*str*) – the item to be set. See below for allowed values.
- **value** – the value to be set.
- **variable** (*str*) – used only with control set to “parmMin” or “parmMax”

Allowed values for *control* parameter:

- '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 ranId 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 set to False. Ignored for non-sequential fits.
- 'Reverse Seq': when True, sequential refinement is performed on the reversed list of histograms.

- 'parmMin' & 'parmMax': set a maximum or minimum value for a refined parameter. Note that variable will be a GSAS-II variable name, optionally with * specified for a histogram or atom number and value must be a float. (See [Parameter Limits](#) description.)

See also:

`get_Controls()`

set_Frozen (*variable=None, histogram=None, mode='remove'*)

Removes one or more Frozen variables (or adds one) (See [Parameter Limits](#) description.) Note that use of this will cause the project to be saved if not already done so.

Parameters

- **variable** (*str*) – a variable name as a str or (as a `GSASIIObj.G2VarObj` object). Should not contain wildcards. If None (default), all frozen variables are deleted from the project, unless a sequential fit and a histogram is specified.
- **histogram** – A reference to a histogram, which can be reference by object, name, or number. Used for sequential fits only.
- **mode** (*str*) – The default mode is to remove variables from the appropriate Frozen list, but if the mode is specified as 'add', the variable is added to the list.

Returns

True if the variable was added or removed, False otherwise. Exceptions are generated with invalid requests.

set_refinement (*refinement, histogram='all', phase='all'*)

Set refinement flags at the project level to specified 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:

```
G2PwdrData.set_refinements()          G2PwdrData.clear_refinements()
G2Phase.set_refinements()             G2Phase.clear_refinements()      G2Phase.
set_HAP_refinements()                 G2Phase.clear_HAP_refinements()    G2Single.
set_refinements()
```

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.

Scripts should not try to create a *G2PwdrData* object directly as *G2PwdrData.__init__()* should be invoked from inside *G2Project*.

property Background

Provides a list with with the Background parameters for this histogram.

Returns

list containing a list and dict with background values

EditSimulated (*Tmin, Tmax, Tstep=None, Npoints=None*)

Change the parameters for an existing simulated powder histogram. This will reset the previously computed "observed" pattern.

Parameters

- **Tmin** (*float*) – Minimum 2theta or TOF (microsec) for dataset to be simulated
- **Tmax** (*float*) – Maximum 2theta or TOF (usec) for dataset to be simulated
- **Tstep** (*float*) – Step size in 2theta or TOF (usec) for dataset to be simulated Default is to compute this from Npoints.
- **Npoints** (*int*) – the number of data points to be used for computing the diffraction pattern. Defaults as None, which sets this to 2500. Do not specify both Npoints and Tstep. Due to roundoff the actual number of points used may differ by +/-1 from Npoints. Must be below 25,000.

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

property 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

property 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)

property 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;#'

property 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 :param float area: peak area :param float dspace: peak position as d-space (A) :param float Q: peak position as Q (A-1) :param float ttheta: peak position as 2Theta (deg)

Note: only one of the parameters: dspace, Q or ttheta may be specified. See [Peak Fitting](#) for an example.

calc_autobkg (*opt=0*, *logLam=None*)

Sets fixed background points using the pybaselines Whittaker algorithm.

Parameters

- **opt** (*int*) – 0 for 'arpls' or 1 for 'iarpls'. Default is 0.

- **logLam** (*float*) – log₁₀ of the Lambda value used in the pybase-lines.whittaker.arpls/iarpls computation. If None (default) is provided, a guess is taken for an appropriate value based on the number of points.

Returns

the array of computed background points

clear_refinements (*refs*)

Clears the PWDR 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.

getHistEntryList (*keyname=""*)

Returns a dict with histogram setting values.

Parameters

keyname (*str*) – an optional string. When supplied only entries where at least one key contains the specified string are reported. Case is ignored, so 'sg' will find entries where one of the keys is 'SGdata', etc.

Returns

a set of histogram dict keys.

See [G2Phase.getHAPentryList\(\)](#) for a related example.

See also:

[getHistEntryValue\(\)](#) [setHistEntryValue\(\)](#)

getHistEntryValue (*keylist*)

Returns the histogram control value associated with a list of keys. Where the value returned is a list, it may be used as the target of an assignment (as in `getHistEntryValue(...)[...] = val`) to set a value inside a list.

Parameters

keylist (*list*) – a list of dict keys, typically as returned by [getHistEntryList\(\)](#).

Returns

a histogram setting; may be a int, float, bool, list,...

See [G2Phase.getHAPentryValue\(\)](#) for a related example.

get_wR ()

returns the overall weighted profile R factor for a histogram

Returns

a wR value as a percentage or None if not defined

getdata (*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*=[*l*])

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 (*mode*='useIP')

Causes a refinement of peak position, background and instrument parameters

Parameters

mode (*str*) – this determines how peak widths are determined. If the value is 'useIP' (the default) then the width parameter values (sigma, gamma, alpha,...) are computed from the histogram's instrument parameters. If the value is 'hold', then peak width parameters are not overridden. In this case, it is not possible to refine the instrument parameters associated with the peak widths and an attempt to do so will result in an error.

Returns

a list of dicts with refinement results. Element 0 has uncertainties on refined values (also placed in self.data['Peak List']['sigDict']) element 1 has the peak fit result, element 2 has the peak fit uncertainties and element 3 has r-factors from the fit. (These are generated in GSASIIpwd.DoPeakFit()).

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).

property residuals

Provides a dictionary 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).

setHistEntryValue (*keylist*, *newvalue*)

Sets a histogram control value associated with a list of keys.

See [G2Phase.setHAPentryValue\(\)](#) for a related example.

Parameters**keylist** (*list*) –

a list of dict keys, typically as returned by

`getHistEntryList()`.**param newvalue**

a new value for the hist setting. The type must be the same as the initial value, but if the value is a container (list, tuple, np.array,...) the elements inside are not checked.

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.

| key name | type of value | | meaning of value |
|-----------------|---------------|--------------------|---|
| fixed-Hist | int, None | str, or G2PwdrData | reference to a histogram in the current project or None to remove the reference. |
| fixed-File-Mult | float | | multiplier applied to intensities in the background histogram where a value of -1.0 means full subtraction of the background histogram. |

- **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, alp=None, bet=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 sigma (Gaussian width) value. If None (the default), no change is made.
- **gam** (*bool*) – Sets or clears the refinement flag for the peak gamma (Lorentzian width) value. If None (the default), no change is made.
- **alp** (*bool*) – Sets or clears the refinement flag for the peak alpha (TOF width) value. If None (the default), no change is made.
- **bet** (*bool*) – Sets or clears the refinement flag for the peak beta (TOF 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 PWDR 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

Scripts should not try to create a [G2SeqRefRes](#) object directly as this object will be created when a .gpx project file is read.

As an example:

```
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:

```
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 indices 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

class GSASIIscriptable.**G2Single** (data, proj, name)

Wrapper for a HKLF tree entry, containing a single crystal histogram Note that in a GSASIIscriptable script, instances of G2Single will be created by calls to `G2Project.histogram()`, `G2Project.histograms()`, or `G2Project.add_single_histogram()`. Scripts should not try to create a `G2Single` object directly.

This object contains these class variables:

- `G2Single.proj`: contains a reference to the `G2Project` object that contains this histogram
- `G2Single.name`: contains the name of the histogram
- `G2Single.data`: contains the histogram's associated data in a dict, as documented for the [Single Crystal Tree Item](#). This contains the actual histogram values, as documented for Data.

Example use of `G2Single`:

```
gpx0 = G2sc.G2Project('HTO_base.gpx')
gpx0.add_single_histogram('HTO_xray/xtal1/xs2555a.hkl', 0, fmthint='Shelx HKLF 4')
gpx0.save('HTO_scripted.gpx')
```

This opens an existing GSAS-II project file and adds a single crystal dataset that is linked to the first phase and saves it under a new name.

See also:

`add_single_histogram()` `histogram()` `histograms()` `link_histogram_phase()`

clear_refinements (refs)

Clears the HKLF 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.

Example:

```
hist.clear_refinements(['Scale', 'Es', 'Flack'])
hist.clear_refinements({'Scale': True, 'Es': False, 'Flack': True})
```

Note that the two above commands are equivalent: the values specified in the dict in the second command are ignored.

set_refinements (*refs*)

Sets the HKLF 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.

Example:

```
hist.set_refinements({'Scale':True, 'Es':False, 'Flack':True})
```

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.GenHLaue()`).

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:

```
>>> 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]
```

(continues on next page)

(continued from previous page)

```
[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:

```
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','Restrains',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]},
  'Restrains':{'data':{'restraint data dict'}},
  'Constraints':{'data':{'constraint data dict'}}
}
nameList = [                                     #NB: reproduces tree order
  ['Notebook'],
  ['Controls'],
  ['Covariance'],
  ['Constraints'],
  ['Restrains'],
  ['Rigid bodies'],
  ['PWDR FAP.XRA Bank 1',
    'Comments',
    'Limits',
    'Background',
```

(continues on next page)

(continued from previous page)

```

    '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 described 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.SetDefaultDDData(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 documented 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)

Initialize phase data.

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
-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 = 128`

Integration block size; 128 or 256 seems to be optimal for CPU use, but 128 uses less memory, must be <=1024 (for polymask/histogram3d)

`GSASIIscriptable.calcMaskMap (imgprms, mskprms)`

Computes a set of blocked mask arrays for a set of image controls and mask parameters. This capability is also provided with `G2Image.IntMaskMap()`.

`GSASIIscriptable.calcThetaAzimMap (imgprms)`

Computes the set of blocked arrays for theta-azimuth mapping from a set of image controls, which can be cached and reused for integration of multiple images with the same calibration parameters. This capability is also provided with `G2Image.IntThetaAzMap()`.

`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
```

(continues on next page)

(continued from previous page)

```
-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.**dictDive** (*d*, *search*="", *keylist*=[], *firstcall*=True, *l*=None)

Recursive routine to scan a nested dict. Reports a list of keys and the associated type and value for that key.

Parameters

- **d** (*dict*) – a dict that will be scanned
- **search** (*str*) – an optional search string. If non-blank, only entries where one of the keys contains search (case ignored)
- **keylist** (*list*) – a list of keys to apply to the dict.
- **firstcall** (*bool*) – do not specify
- **l** (*list*) – do not specify

Returns

a list of keys located by this routine in form [(*keylist*, type, value),...] where if *keylist* is ['a','b','c'] then d[['a']['b']['c']] will have the value.

This routine can be called in a number of ways, as are shown in a few examples:

```
>>> for i in G2sc.dictDive(p.data['General'],'paw'): print(i)
...
(['Pawley dmin'], <class 'float'>, 1.0)
(['doPawley'], <class 'bool'>, False)
(['Pawley dmax'], <class 'float'>, 100.0)
(['Pawley neg wt'], <class 'float'>, 0.0)
>>>
>>> for i in G2sc.dictDive(p.data,'paw',['General']): print(i)
...
(['General', 'Pawley dmin'], <class 'float'>, 1.0)
(['General', 'doPawley'], <class 'bool'>, False)
(['General', 'Pawley dmax'], <class 'float'>, 100.0)
(['General', 'Pawley neg wt'], <class 'float'>, 0.0)
>>>
>>> for i in G2sc.dictDive(p.data,'',['General','doPawley']): print(i)
...
(['General', 'doPawley'], <class 'bool'>, False)
```

GSASIIscriptable.**dump** (*args*)

Implements the dump command-line subcommand, which shows the contents of a GSAS-II project:

```
usage: GSASIIscriptable.py dump [-h] [-d] [-p] [-r] files [files ...]
```

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.**installScriptingShortcut** ()

Creates a file named G2script in the current Python site-packages directory. This is equivalent to the “Install GSASIIscriptable shortcut” command in the GUI’s File menu. Once this is done, a shortcut for calling GSASIIscriptable is created, where the command:

```
>>> import G2script as G2sc
```

will provide access to GSASIIscriptable without changing the sys.path; also see [Shortcut for Scripting Access](#).

Note that this only affects the current Python installation. If more than one Python installation will be used with GSAS-II (for example because different conda environments are used), this command should be called from within each Python environment.

If more than one GSAS-II installation will be used with a Python installation, this shortcut can only be used with one of them.

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, see `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.**patchControls**(*Controls*)

patch routine to convert variable names used in parameter limits to G2VarObj objects (See [Parameter Limits](#) description.)

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
```

REFERENCES TO THE GSAS-II DEVELOPER'S DOCUMENTATION

Constraints_processing: See the [Constraints Processing](#) from GSASIImapvars.

REFERENCES TO THE GSAS-II DEVELOPER'S DOCUMENTATION

Imports: See the [Import models documentation](#) for information on reading data files.

PYTHON MODULE INDEX

g

GSASIIobj, [36](#)

GSASIIscriptable, [82](#)

Symbols

__eq__ () (GSASIIobj.G2VarObj method), 41
 __hash__ () (GSASIIobj.G2VarObj method), 42
 __init__ () (GSASIIobj.ExpressionCalcObj method), 38
 __init__ () (GSASIIobj.ExpressionObj method), 40
 __init__ () (GSASIIobj.G2Exception method), 40
 __init__ () (GSASIIobj.G2RefineCancel method), 41
 __init__ () (GSASIIobj.G2VarObj method), 42
 __init__ () (GSASIIobj.ImportBaseclass method), 44
 __init__ () (GSASIIobj.ImportImage method), 45
 __init__ () (GSASIIobj.ImportPDFData method), 45
 __init__ () (GSASIIobj.ImportPhase method), 45
 __init__ () (GSASIIobj.ImportPowderData method), 45
 __init__ () (GSASIIobj.ImportReflectometryData method), 45
 __init__ () (GSASIIobj.ImportSmallAngleData method), 46
 __init__ () (GSASIIobj.ImportStructFactor method), 46
 __init__ () (GSASIIobj.ShowTiming method), 49
 __repr__ () (GSASIIobj.G2VarObj method), 42
 __str__ () (GSASIIobj.G2Exception method), 40
 __str__ () (GSASIIobj.G2RefineCancel method), 41
 __str__ () (GSASIIobj.G2VarObj method), 42
 __weakref__ (GSASIIobj.ExpressionCalcObj attribute), 38
 __weakref__ (GSASIIobj.ExpressionObj attribute), 40
 __weakref__ (GSASIIobj.G2Exception attribute), 41
 __weakref__ (GSASIIobj.G2RefineCancel attribute), 41
 __weakref__ (GSASIIobj.G2VarObj attribute), 42
 __weakref__ (GSASIIobj.ImportBaseclass attribute), 44
 __weakref__ (GSASIIobj.ImportBaseclass.ImportException attribute), 44
 __weakref__ (GSASIIobj.ShowTiming attribute), 49
 _lookup () (in module GSASIIobj), 50
 _show () (GSASIIobj.G2VarObj method), 42

A

add () (in module GSASIIscriptable), 122
 add_atom () (GSASIIscriptable.G2Phase method), 92
 add_back_peak () (GSASIIscriptable.G2PwdrData method), 113

add_constraint_raw () (GSASIIscriptable.G2Project method), 101
 add_EqnConstr () (GSASIIscriptable.G2Project method), 99
 add_EquivConstr () (GSASIIscriptable.G2Project method), 100
 add_HoldConstr () (GSASIIscriptable.G2Project method), 100
 add_image () (GSASIIscriptable.G2Project method), 101
 add_NewVarConstr () (GSASIIscriptable.G2Project method), 100
 add_PDF () (GSASIIscriptable.G2Project method), 101
 add_peak () (GSASIIscriptable.G2PwdrData method), 113
 add_phase () (GSASIIscriptable.G2Project method), 102
 add_powder_histogram () (GSASIIscriptable.G2Project method), 102
 add_simulated_powder_histogram () (GSASIIscriptable.G2Project method), 103
 add_single_histogram () (GSASIIscriptable.G2Project method), 104
 addDistRestraint () (GSASIIscriptable.G2Phase method), 91
 AddPhase2Index () (in module GSASIIobj), 36
 ADP (GSASIIscriptable.G2AtomRecord property), 82
 adp_flag (GSASIIscriptable.G2AtomRecord property), 83
 assgnVars (GSASIIobj.ExpressionObj attribute), 40
 atom () (GSASIIscriptable.G2Phase method), 92
 AtomIdLookup (in module GSASIIobj), 36
 AtomRanIdLookup (in module GSASIIobj), 36
 Atoms record description, 23
 atoms () (GSASIIscriptable.G2Phase method), 92

B

Background (GSASIIscriptable.G2PwdrData property), 112
 Banks (GSASIIobj.ImportStructFactor attribute), 46
 blkSize (in module GSASIIscriptable), 123

C

calc_autobkg () (GSASIIscriptable.G2PwdrData

method), 113
 calcMaskMap() (in module *GSASIIscriptable*), 123
 calcThetaAzimMap() (in module *GSASIIscriptable*), 123
 calculate() (*GSASIIscriptable.G2PDF* method), 89
 CheckVars() (*GSASIIobj.ExpressionObj* method), 39
 CIFValidator() (*GSASIIobj.ImportBaseclass* method), 43
 clear_HAP_refinements() (*GSASIIscriptable.G2Phase* method), 93
 clear_refinements() (*GSASIIscriptable.G2Phase* method), 93
 clear_refinements() (*GSASIIscriptable.G2PwdrData* method), 114
 clear_refinements() (*GSASIIscriptable.G2Single* method), 119
 clearDistRestraint() (*GSASIIscriptable.G2Phase* method), 92
 clearImageCache() (*GSASIIscriptable.G2Image* method), 86
 clearPixelMask() (*GSASIIscriptable.G2Image* method), 86
 clone_powder_histogram() (*GSASIIscriptable.G2Project* method), 104
 compiledExpr (*GSASIIobj.ExpressionCalcObj* attribute), 38
 CompileVarDesc() (in module *GSASIIobj*), 37
 composition (*GSASIIscriptable.G2Phase* property), 93
 ComputeWorstFit() (*GSASIIscriptable.G2Project* method), 99
 Constraint definition object description, 16
 Constraints object description, 15
 ContentsValidator() (*GSASIIobj.ImportBaseclass* method), 43
 Controllist (*GSASIIscriptable.G2Image* attribute), 84
 coordinates (*GSASIIscriptable.G2AtomRecord* property), 83
 copy_PDF() (*GSASIIscriptable.G2Project* method), 105
 copyHAPvalues() (*GSASIIscriptable.G2Phase* method), 93
 copyHistParms() (*GSASIIscriptable.G2Project* method), 104
 Covariance description, 16
 create() (in module *GSASIIscriptable*), 123
 CreatePDFitems() (in module *GSASIIobj*), 37

D

Data object descriptions
 Atoms record, 23
 Constraint Definition, 16
 Constraints, 15
 Covariance, 16
 Drawing atoms record, 23

Phase, 17
 Powder Data, 24
 Powder Reflections, 28
 Rigid Body Data, 20
 Single crystal data, 28
 Single Crystal Reflections, 29
 Space Group Data, 21
 Superspace Group Data, 22
 DefaultControls (in module *GSASIIobj*), 37
 del_back_peak() (*GSASIIscriptable.G2PwdrData* method), 114
 density (*GSASIIscriptable.G2Phase* property), 94
 dictDive() (in module *GSASIIscriptable*), 124
 do_refinements() (*GSASIIscriptable.G2Project* method), 105
 Drawing atoms record description, 23
 dump() (in module *GSASIIscriptable*), 124

E

EditExpression() (*GSASIIobj.ExpressionObj* method), 39
 EditSimulated() (*GSASIIscriptable.G2PwdrData* method), 112
 element (*GSASIIscriptable.G2AtomRecord* property), 83
 eObj (*GSASIIobj.ExpressionCalcObj* attribute), 38
 EvalExpression() (*GSASIIobj.ExpressionCalcObj* method), 37
 export() (*GSASIIscriptable.G2PDF* method), 89
 Export() (*GSASIIscriptable.G2PwdrData* method), 112
 export() (in module *GSASIIscriptable*), 125
 export_CIF() (*GSASIIscriptable.G2Phase* method), 94
 Export_peaks() (*GSASIIscriptable.G2PwdrData* method), 112
 exportersByExtension (in module *GSASIIscriptable*), 125
 exprDict (*GSASIIobj.ExpressionCalcObj* attribute), 38
 expression (*GSASIIobj.ExpressionObj* attribute), 40
 ExpressionCalcObj (class in *GSASIIobj*), 37
 ExpressionObj (class in *GSASIIobj*), 38
 ExtensionValidator() (*GSASIIobj.ImportBaseclass* method), 43

F

findControl() (*GSASIIscriptable.G2Image* method), 86
 FindFunction() (in module *GSASIIobj*), 40
 fit_fixed_points() (*GSASIIscriptable.G2PwdrData* method), 114
 fmtVarByMode() (*GSASIIobj.G2VarObj* method), 42
 fmtVarDescr() (in module *GSASIIobj*), 50
 freeVars (*GSASIIobj.ExpressionObj* attribute), 40
 from_dict_and_names() (*GSASIIscriptable.G2Project* class method), 105
 fxnpkgdict (*GSASIIobj.ExpressionCalcObj* attribute), 38

G

G2AtomRecord (class in GSASIIscriptable), 82
 G2Exception, 40
 G2Image (class in GSASIIscriptable), 83
 G2ImportException, 88
 G2ObjectWrapper (class in GSASIIscriptable), 88
 G2PDF (class in GSASIIscriptable), 88
 G2Phase (class in GSASIIscriptable), 90
 G2Project (class in GSASIIscriptable), 98
 G2PwdrData (class in GSASIIscriptable), 111
 G2RefineCancel, 41
 G2ScriptException, 117
 G2SeqRefRes (class in GSASIIscriptable), 117
 G2Single (class in GSASIIscriptable), 119
 G2VarObj (class in GSASIIobj), 41
 GeneratePixelMask() (GSASIIscriptable.G2Image method), 84
 GenerateReflections() (in module GSASIIscriptable), 120
 GenWildCard() (in module GSASIIobj), 42
 get_cell() (GSASIIscriptable.G2Phase method), 95
 get_cell_and_esd() (GSASIIscriptable.G2Phase method), 95
 get_cell_and_esd() (GSASIIscriptable.G2SeqRefRes method), 119
 get_Constraints() (GSASIIscriptable.G2Project method), 105
 get_Controls() (GSASIIscriptable.G2Project method), 106
 get_Covariance() (GSASIIscriptable.G2Project method), 106
 get_Covariance() (GSASIIscriptable.G2SeqRefRes method), 118
 get_Frozen() (GSASIIscriptable.G2Project method), 106
 get_ParmList() (GSASIIscriptable.G2Project method), 107
 get_ParmList() (GSASIIscriptable.G2SeqRefRes method), 118
 get_Variable() (GSASIIscriptable.G2Project method), 107
 get_Variable() (GSASIIscriptable.G2SeqRefRes method), 118
 get_VaryList() (GSASIIscriptable.G2Project method), 107
 get_VaryList() (GSASIIscriptable.G2SeqRefRes method), 118
 get_wR() (GSASIIscriptable.G2PwdrData method), 114
 getControl() (GSASIIscriptable.G2Image method), 86
 getControls() (GSASIIscriptable.G2Image method), 86
 getdata() (GSASIIscriptable.G2PwdrData method), 114
 GetDepVar() (GSASIIobj.ExpressionObj method), 39

getDescr() (in module GSASIIobj), 50
 getHAPentryList() (GSASIIscriptable.G2Phase method), 94
 getHAPentryValue() (GSASIIscriptable.G2Phase method), 94
 getHAPvalues() (GSASIIscriptable.G2Phase method), 95
 getHistEntryList() (GSASIIscriptable.G2PwdrData method), 114
 getHistEntryValue() (GSASIIscriptable.G2PwdrData method), 114
 GetIndependentVars() (GSASIIobj.ExpressionObj method), 39
 getMasks() (GSASIIscriptable.G2Image method), 87
 getPhaseEntryList() (GSASIIscriptable.G2Phase method), 95
 getPhaseEntryValue() (GSASIIscriptable.G2Phase method), 95
 GetPhaseNames() (in module GSASIIobj), 43
 getVarDescr() (in module GSASIIobj), 50
 GetVaried() (GSASIIobj.ExpressionObj method), 39
 GetVariedVarVal() (GSASIIobj.ExpressionObj method), 39
 getVarStep() (in module GSASIIobj), 50
 getVary() (GSASIIscriptable.G2Image method), 87
 GSAS-II variable naming, 12
 GSASIIobj
 module, 36
 GSASIIscriptable
 module, 82

H

HAPvalue() (GSASIIscriptable.G2Phase method), 90
 HistIdLookup (in module GSASIIobj), 43
 histogram() (GSASIIscriptable.G2Project method), 107
 histograms() (GSASIIscriptable.G2Phase method), 96
 histograms() (GSASIIscriptable.G2Project method), 107
 histograms() (GSASIIscriptable.G2SeqRefRes method), 119
 HistRanIdLookup (in module GSASIIobj), 43
 histType() (GSASIIscriptable.G2Project method), 107
 hold_many() (GSASIIscriptable.G2Project method), 108
 HowDidIgetHere() (in module GSASIIobj), 43

I

image() (GSASIIscriptable.G2Project method), 108
 image: Image data object description, 30
 image: Image object descriptions, 30
 imageMultiDistCalib() (GSASIIscriptable.G2Project method), 108
 images() (GSASIIscriptable.G2Project method), 108
 import_generic() (in module GSASIIscriptable), 125
 ImportBaseclass (class in GSASIIobj), 43

ImportBaseclass.ImportException, 44
 ImportImage (class in GSASIIobj), 44
 ImportPDFData (class in GSASIIobj), 45
 ImportPhase (class in GSASIIobj), 45
 ImportPowderData (class in GSASIIobj), 45
 ImportReflectometryData (class in GSASIIobj), 45
 ImportSmallAngleData (class in GSASIIobj), 45
 ImportStructFactor (class in GSASIIobj), 46
 IndexAllIds() (in module GSASIIobj), 46
 initMasks() (GSASIIscriptable.G2Image method), 87
 InitParameters() (GSASIIobj.ImportImage method), 45
 InitParameters() (GSASIIobj.ImportStructFactor method), 46
 installScriptingShortcut() (in module GSASIIscriptable), 125
 InstrumentParameters (GSASIIscriptable.G2PwdrData property), 112
 Integrate() (GSASIIscriptable.G2Image method), 85
 IntMaskMap() (GSASIIscriptable.G2Image method), 85
 IntThetaAzMap() (GSASIIscriptable.G2Image method), 85
 IPyBrowse() (in module GSASIIscriptable), 121
 iter_refinements() (GSASIIscriptable.G2Project method), 109

L

label (GSASIIscriptable.G2AtomRecord property), 83
 lastError (GSASIIobj.ExpressionObj attribute), 40
 lblLookup (GSASIIobj.ExpressionCalcObj attribute), 38
 link_histogram_phase() (GSASIIscriptable.G2Project method), 109
 load_iprms() (in module GSASIIscriptable), 125
 load_pwd_from_reader() (in module GSASIIscriptable), 125
 loadControls() (GSASIIscriptable.G2Image method), 87
 LoadDictFromProjFile() (in module GSASIIscriptable), 121
 LoadExpression() (GSASIIobj.ExpressionObj method), 39
 LoadG2fil() (in module GSASIIscriptable), 122
 LoadImage() (GSASIIobj.ImportImage method), 45
 loadMasks() (GSASIIscriptable.G2Image method), 87
 LoadProfile() (GSASIIscriptable.G2PwdrData method), 113
 LookupAtomId() (in module GSASIIobj), 47
 LookupAtomLabel() (in module GSASIIobj), 47
 LookupHistId() (in module GSASIIobj), 47
 LookupHistName() (in module GSASIIobj), 47
 LookupPhaseId() (in module GSASIIobj), 47
 LookupPhaseName() (in module GSASIIobj), 47
 LookupWildcard() (in module GSASIIobj), 48

M

main() (in module GSASIIscriptable), 125
 make_empty_project() (in module GSASIIscriptable), 126
 make_var_obj() (GSASIIscriptable.G2Project method), 109
 MakeUniqueLabel() (in module GSASIIobj), 48
 MaskFrameMask() (GSASIIscriptable.G2Image method), 85
 MaskThetaMap() (GSASIIscriptable.G2Image method), 86
 module
 GSASIIobj, 36
 GSASIIscriptable, 82
 mu() (GSASIIscriptable.G2Phase method), 96
 mult (GSASIIscriptable.G2AtomRecord property), 83

O

occupancy (GSASIIscriptable.G2AtomRecord property), 83
 optimize() (GSASIIscriptable.G2PDF method), 89

P

Parameter dictionary, 32
 Parameter limits, 35
 Parameter names, 12
 Parameters (GSASIIobj.ImportStructFactor attribute), 46
 parmDict (GSASIIobj.ExpressionCalcObj attribute), 38
 ParseExpression() (GSASIIobj.ExpressionObj method), 40
 patchControls() (in module GSASIIscriptable), 126
 pdf() (GSASIIscriptable.G2Project method), 109
 pdfs() (GSASIIscriptable.G2Project method), 109
 PeakList (GSASIIscriptable.G2PwdrData property), 113
 Peaks (GSASIIscriptable.G2PwdrData property), 113
 Phase information record description, 23
 Phase object description, 17
 phase() (GSASIIscriptable.G2Project method), 109
 PhaseIdLookup (in module GSASIIobj), 48
 PhaseRanIdLookup (in module GSASIIobj), 48
 phases() (GSASIIscriptable.G2Project method), 110
 Powder data CW Instrument Parameters, 27
 Powder data object description, 24
 Powder data TOF Instrument Parameters, 27
 Powder reflection object description, 28
 PreSetup() (in module GSASIIscriptable), 122
 prmlLookup() (in module GSASIIobj), 50

R

ranId (GSASIIscriptable.G2AtomRecord property), 83
 ReadCIF() (in module GSASIIobj), 48

Readers (in module *GSASIIscriptable*), 122
 Recalibrate() (*GSASIIscriptable.G2Image* method), 86
 ref_back_peak() (*GSASIIscriptable.G2PwdrData* method), 115
 RefData() (*GSASIIscriptable.G2SeqRefRes* method), 117
 refine() (*GSASIIscriptable.G2Project* method), 110
 refine() (in module *GSASIIscriptable*), 126
 refine_peaks() (*GSASIIscriptable.G2PwdrData* method), 115
 refinement_flags (*GSASIIscriptable.G2AtomRecord* property), 83
 reflections() (*GSASIIscriptable.G2PwdrData* method), 115
 ReInitialize() (*GSASIIobj.ImportBaseclass* method), 44
 ReInitialize() (*GSASIIobj.ImportImage* method), 45
 ReInitialize() (*GSASIIobj.ImportPDFData* method), 45
 ReInitialize() (*GSASIIobj.ImportPowderData* method), 45
 ReInitialize() (*GSASIIobj.ImportReflectometryData* method), 45
 ReInitialize() (*GSASIIobj.ImportSmallAngleData* method), 46
 ReInitialize() (*GSASIIobj.ImportStructFactor* method), 46
 reload() (*GSASIIscriptable.G2Project* method), 110
 removeNonRefined() (in module *GSASIIobj*), 53
 residuals (*GSASIIscriptable.G2PwdrData* property), 115
 reVarDesc (in module *GSASIIobj*), 51
 reVarStep (in module *GSASIIobj*), 53
 Rigid Body Data description, 20

S

SampleParameters (*GSASIIscriptable.G2PwdrData* property), 113
 save() (*GSASIIscriptable.G2Project* method), 110
 saveControls() (*GSASIIscriptable.G2Image* method), 87
 SaveDictToProjFile() (in module *GSASIIscriptable*), 122
 SaveProfile() (*GSASIIscriptable.G2PwdrData* method), 113
 seqref() (*GSASIIscriptable.G2Project* method), 110
 set_background() (*GSASIIscriptable.G2PDF* method), 90
 set_background() (*GSASIIscriptable.G2PwdrData* method), 116
 set_Controls() (*GSASIIscriptable.G2Project* method), 110
 set_formula() (*GSASIIscriptable.G2PDF* method), 90

set_Frozen() (*GSASIIscriptable.G2Project* method), 111
 set_HAP_refinements() (*GSASIIscriptable.G2Phase* method), 97
 set_peakFlags() (*GSASIIscriptable.G2PwdrData* method), 116
 set_refinement() (*GSASIIscriptable.G2Project* method), 111
 set_refinements() (*GSASIIscriptable.G2Phase* method), 98
 set_refinements() (*GSASIIscriptable.G2PwdrData* method), 117
 set_refinements() (*GSASIIscriptable.G2Single* method), 120
 setCalibrant() (*GSASIIscriptable.G2Image* method), 87
 setControl() (*GSASIIscriptable.G2Image* method), 87
 setControlFile() (*GSASIIscriptable.G2Image* method), 88
 setControls() (*GSASIIscriptable.G2Image* method), 88
 SetDefaultDDData() (in module *GSASIIscriptable*), 122
 SetDefaultSample() (in module *GSASIIobj*), 48
 SetDepVar() (*GSASIIobj.ExpressionObj* method), 40
 setDistRestraintWeight() (*GSASIIscriptable.G2Phase* method), 96
 setHAPentryValue() (*GSASIIscriptable.G2Phase* method), 96
 setHAPvalues() (*GSASIIscriptable.G2Phase* method), 96
 setHistEntryValue() (*GSASIIscriptable.G2PwdrData* method), 115
 setMasks() (*GSASIIscriptable.G2Image* method), 88
 SetNewPhase() (in module *GSASIIobj*), 48
 setPhaseEntryValue() (*GSASIIscriptable.G2Phase* method), 97
 SetPrintLevel() (in module *GSASIIscriptable*), 122
 setSampleProfile() (*GSASIIscriptable.G2Phase* method), 97
 SetupCalc() (*GSASIIobj.ExpressionCalcObj* method), 38
 SetupGeneral() (in module *GSASIIscriptable*), 122
 setVary() (*GSASIIscriptable.G2Image* method), 88
 ShortHistNames (in module *GSASIIobj*), 48
 ShortPhaseNames (in module *GSASIIobj*), 49
 ShowTiming (class in *GSASIIobj*), 49
 Single Crystal data object description, 28
 Single Crystal reflection object description, 29
 SortVariables() (in module *GSASIIobj*), 49
 Space Group Data description, 21
 StripUnicode() (in module *GSASIIobj*), 49

`su` (*GSASIIobj.ExpressionCalcObj attribute*), 38
`Superspace Group Data description`, 22

T

`TestFastPixelMask()` (*GSASIIscriptable.G2Image method*), 86
`TestIndexAll()` (*in module GSASIIobj*), 49
`type` (*GSASIIscriptable.G2AtomRecord property*), 83

U

`uiso` (*GSASIIscriptable.G2AtomRecord property*), 83
`update_ids()` (*GSASIIscriptable.G2Project method*), 111
`UpdateDict()` (*GSASIIobj.ExpressionCalcObj method*), 38
`UpdateParameters()` (*GSASIIobj.ImportStructFactor method*), 46
`UpdateVariedVars()` (*GSASIIobj.ExpressionObj method*), 40
`UpdateVars()` (*GSASIIobj.ExpressionCalcObj method*), 38

V

`validateAtomDrawType()` (*in module GSASIIobj*), 53
`VarDescr()` (*in module GSASIIobj*), 49
`varLookup` (*GSASIIobj.ExpressionCalcObj attribute*), 38
`varname()` (*GSASIIobj.G2VarObj method*), 42

Y

`y_calc()` (*GSASIIscriptable.G2PwdrData method*), 117