GSAS-II Developers Documentation

Release 65073c 18-Jun-2026 09:50 (#5836)

Robert B. Von Dreele and Brian H. Toby

Jun 18, 2026

CONTENTS

The documentation here is intended for those wishing to extend the capabilities within the GSAS-II framework, for scientists/students working to understand how GSAS-II works, or for people wishing to develop scripting applications using the GSAS-II Python API (module `GSASIIscriptable`). Note that most data structures used in GSAS-II are defined in module `GSASIIobj`.

For information on downloading/installing GSAS-II, please see the GSAS-II home page: https://advancedphotonsource.github.io/GSAS-II-tutorials. To learn how to use GSAS-II, please see the tutorials, which can be accessed from the GSAS-II Help menu or from this web page: https://advancedphotonsource.github.io/GSAS-II-tutorials/tutorials.html.

For scripting use of GSAS-II, see *GSASIIscriptable: Scripting Interface*.

Invention Information: Argonne has assigned Software Invention #SF-22-147 to GSAS-II.

CHAPTER

ONE

GSAS-II EXECUTION ENVIRONMENT

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 need to modify the meson files.

  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 some older compiled binaries are provided at present for both, but 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 GSAS2MAIN 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 wxpython (quite slow). Please let Brian know if you are intending to use GSAS-II on a Raspberry Pi for a classroom, etc and would need help with this.

1.2 Source Code Management

The master version of the source code for GSAS-II resides on GitHub at URL (in branch main) 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 and associated packages. 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.

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, 3.12 or 3.13 is recommended. No testing has yet been done with Python 3.14, but no problems are expected. GSAS-II should run with any Python version from 3.7 or later, but we do not create binaries for all versions of Python and numpy. You will need to locate (from the old subversion server) older binaries to match older Python versions or compile them yourself.
- wxPython 4.2 or later is recommended, but with Python <=3.9 any wx4.x version should be OK. Problems with newer sections of the GUI are expected for wx <4.0.
- NumPy 1.26 recommended with Python 3.11 and 2.2 with 3.12 or 3.13, but anything from 1.17 on is likely fine, but if you do not match the supplied GSAS-II binaries you will need to build them yourself.
- matplotlib-base Note that matplotlib-base is preferred over matplotlib unless matplotlib will be used outside GSAS-II. 3.10 is recommended, but anything later than 3.4 should be fine.
- 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.
- PyCifRW: no version issues are known. We had been using an older version for a long time, but in 2025 switched to the latest version and did not see any problems.
- pybaselines: no version issues are known.

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 through 3.13. 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.

1.3.1 GUI Requirements

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

- wxPython (http://wxpython.org/docs/api/). Note that GSAS-II has been tested with 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).
- PyCifRW: (https://github.com/jamesrhester/pycifrw)

GSAS-II will not start or will start but will not be able to do much if the above packages are not available.

1.3.2 Recommended Packages for GUI Use

In addition to the previous required packages, several Python packages are utilized in limited 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 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 and hdf5: 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 importers 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 utilized for plotting superspace (modulated) structures.
- seekpath is used for magnetic lattice (k-vector) searches (https://seekpath.readthedocs.io)
- 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.

- pybaselines: Determines a background for a powder pattern in the "autobackground" option. See https://pybaselines.readthedocs.io and https://github.com/derb12/pybaselines for more information.
- xmltodict: Needed to read Bruker BRML files. The BRML importer will not appear in the importer menu if this package is not installed.
- 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.
- zarr: The zarr package is used to read and write compressed hierarchical files. It is used by the APS MIDAS program to produce files of integrated powder diffraction patterns.
- sympy: This package performs symbolic computations and is used for k-vector searching with ISODISTORT.

***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=3.13  numpy=2.2 wxpython scipy matplotlib-base pyopengl␣
↪pillow h5py imageio requests git gitpython pycifrw pybaselines -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-base scipy wxpython ␣
↪pyopengl imageio h5py hdf5 pillow requests pycifrw pybaselines ipython conda␣
↪spyder-kernels meson 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>
```

Note that binaries for Python 3.12 and 3.13 using numpy 2.2 are also now supplied.

1.3.3 Scripting Requirements

The GSAS-II scripting interface (`GSASIIscriptable`) will not run without the NumPy Python extension package:

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

In theory, GSAS-II should start without access to the CIF read/write library, PyCifRW, but in practice, almost everything one wants to do with GSAS-II needs CIF access at some point and I have never tested without this package, so I will consider this also as mandatory for scripting:

- PyCifRW: (https://github.com/jamesrhester/pycifrw)

While not required, and not used very much in GSAS-II scripting, installing the SciPy is recommended:

- SciPy (http://docs.scipy.org/doc/scipy/reference/).

These packages fortunately are common and are easy to install.

1.3.4 Recommended Packages for Scripting

There are some relatively minor scripting capabilities that will only run when a few additional packages are installed:

- requests: for web access
- matplotlib (http://matplotlib.org/contents.html),
- Pillow (https://pillow.readthedocs.org) and/or
- h5py (requires hdf5). Used to read HDF5 files.
- pybaselines: for auto-background (https://github.com/derb12/pybaselines)
- xmltodict: for reading Bruker BRML files.
- zarr: reading powder data files produced MIDAS (APS)
- seekpath: for k-vector searching

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

1.3.5 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.
- The sphinx-rtd-theme is required to build the documentation in standard the format (though this can be changed with minor editing.)

1.3.6 Compilation Requirements

Most users on Windows and Mac will not need to compile GSAS-II. Binaries are supplied as part of the gsas2main self-installer. Linux users may need to install the software in a manner that allows for local compilation. Developers may wish to perform all installation steps for themselves. These are the requirements:

- The gfortran complier is required. There has been some work done with glang, and I think this passes the self-tests but it is unknown if there are other problems. This can be installed in a number of ways. For Windows and Mac, conda-forge is a good choice. (For MacOS, Apple's XCode must also be installed). For Linux, dist-supplied versions are probably a better choice.
- gcc or other c compiler is required to build one binary for image processing. For Windows use Microsoft Visual C/C++. On Mac, use of conda-forge to install gcc is a good installation choice (again XCode is required). For Linux, dist-supplied versions are probably a better choice.
- meson (https://mesonbuild.com/meson-python/) is used to compile the relatively small amount of Fortran, C and Cython code that is included with GSAS-II. This is a Python package typically installed with conda or pip. On Linux, a dist-supplied version (Debian, RedHat, etc.) is likely available too.
- Cython is needed to build one binary used for magnetism (k-vector searching). Install this typically with conda or pip.

1.3.7 Installation Notes for 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 pycifrw matplotlib-base 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 note that matplotlib, pillow, h5py 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.

1.4 Required Binary Files

As noted before, GSAS-II also requires that some code be compiled. For the following platforms:

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

Some binaries are also supplied for Raspberry Pi, but may not be up-to-date. Please ask for newer if needed:

- Linux: ARM processors (64-bit and 32-bit Raspberry Pi OS and Ubuntu for Raspberry Pi).

Binary images are provided at https://github.com/AdvancedPhotonSource/GSAS-II-buildtools/releases/latest. At present binaries are supplied for the following versions:

- Python 3.11 and NumPy 1.26
- Python 3.12 and NumPy 2.2
- Python 3.13 and NumPy 2.2

Note that these binaries must match the major and minor version of both Python. Usually if the minor version is close to the numpy version (1.25.x and 1.27.x for 1.26) the binaries will still work.

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. The build process was recently updated to use meson (in place of scons).

1.5 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

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.

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 (http://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://iso.byu.edu/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. These are listed below. See the web installation instructions for details on how they are installed.

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

RMCProfile
: Provides large-box PDF & S(Q) fitting. The GSAS-II interface was originally written for use with release 6.7.7 of RMCProfile, but updates have been made for compatible with 6.7.9 as well. RMCProfile 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.

PDFfit2
: For small-box fitting of PDFs; see https://github.com/diffpy/diffpy.pdffit2?tab=readme-ov-file#-diffpypdffit2. This software is no longer developed, but it is being maintained with respect to new Python versions.

CHAPTER

TWO

VERSION NUMBERING USED IN GSAS-II

2.1 Git commits

GSAS-II versions are numbered in several ways. Every time a change to one or more GSAS-II source files is committed to a repo by the git source code management system, git assigns the set of changes a 40 character hexadecimal "hash" code for that version. Usually the hash code is abbreviated to the first 6-8 characters, which is still likely to be unique. We refer to this label as the git commit number.

2.2 Integer Tag numbers

When a significant change is made to GSAS-II, a developer can assign an integer tag number. These numbers are intended to be assigned consecutively and thus can be used to establish when a GSAS-II version has been superseded. As of January 2025, the current tag number is 5802. Tag numbers can be looked up in GitHub.

2.3 Release version numbers

Software packaging systems require version numbers of the form X.Y.Z, where the numbers X, Y and Z are labeled as major, minor and micro release version numbers, such as 5.1.3. X and Y are integers. Z is allowed to have a decimal place (such as 5.1.3.1) or may contain letters (such as 5.4.0rc1), but at present GSAS-II will use integers for Z as well. The software numbering system that GSAS-II uses is intended to follow – more or less – the "effective versioning" numbering concept, as implemented below:

- The major release (starting at 5 in 2015) will be changed in GSAS-II when there are very significant changes in how the software is installed or functions that affect large portions of how the program operates. Users are advised to look carefully at the GSAS-II home page to see what has been changed.
- When relatively limited new capabilities are added to GSAS-II or a change is made in how a small section of the program functions, the minor release number is incremented, such as changing from 5.1.3 to 5.2.0. (Note that when the minor release is incremented, the micro release is set to 0.) This will be accompanied with a change to the integer tag number (see above). A change in the minor release indicates that there is some new functionality in GSAS-II that might be useful for users to learn about or a significant change due to a bug.
- Changes to the micro version are made when a minor bug is fixed or a cosmetic change has been applied and where users are encouraged to update. Users will normally not need to consider the details of an update when only the micro version number is incremented and may not want to update every time there is a new micro version released. However, users should always test GSAS-II with the latest version before reporting a bug. The developers may become unhappy after tracking down something that has been reported as not working, only to find that problem was already fixed.

Not every change (git commit) for GSAS-II will be assigned a new micro version. If the change is a development that is not completed or does not introduce any new functionality (for example improvements to documentation) or is otherwise not of general interest, while a new git commit tag will be always generated, the mini version number will not be incremented.

2.4 Version number location

When GSAS-II is installed via git, the version number can be obtained from a git command, but this is relatively slow, so the file *git_verinfo.py* is used to find this information quickly. When GSAS-II is installed directlywithout git, (for example with pixi), this becomes the only way to determine what version of GSAS-II is being run.

2.5 Tools

At present two scripts are provided for developers to use to create or work with integer tag and release version numbers, to save having to review the git logs for previously assigned numbers. All of these scripts update the *git_verinfo.py* version file which is kept so that non-git installs of GSAS-II can report the currect version number.

To assign a new integer tag number and increment the mini version number, use script:

> *…/GSASII/install/incr-version.py* or *…/GSASII/install/incr-version.py mini*
>
> This command is used when there have been minor changes made in GSAS-II and the changes warrent a version number update (for example, due to a bug fix).

To increment the minor version number, use the script with argument “minor”:

> *…/GSASII/install/incr-version.py minor*
>
> This is used when there are significant changes (possibly cumulative) to the code and a new release should be tagged both with a new integer tag number and the minor version number should be increased (e.g. 5.8.2 to 5.9.0).

For non-git installs, there is also this routine:

> *…/GSASII/install/save-versions.py*
>
> This script is used to save the latest version info in the *git_verinfo.py* file. This file is created when a non-git install is made for GSAS-II so that the hash number for the current git version (which cannot be placed into the *git_verinfo.py* file by the *incr-version.py* script) is recorded. The *save-versions.py* script does not change any tags or release version numbers.

2.6 History

When the GSAS-II project was originally started, the subversion source code management system was used to track releases of code. Subversion (svn) assigns a consecutive number every time the code was updated and this version number was used to track GSAS-II software versions. Versions up to ~5700 (early 2024) were maintained with the Argonne svn server. For much of 2024, parallel versions of GSAS-II were kept in both svn and GitHub, but updates were applied with decreasing frequency in svn, so version numbers advanced more slowly. Finally, in November, 2024 GSAS-II version 5799 was the last update made in the Argonne subversion server and update incrementing became discretionary rather than automatic each time an update was applied to the svn server.

Release version numbering starts with version 5.0.0 in early 2025.

CHAPTER

THREE

GSASII DATA OBJECT & VARIABLE ORGANIZATION

This chapter documents how data is organized within the data structures used in GSAS-II.

3.1 *Summary/Contents*

Section Contents

3.2 Parameter names in GSAS-II

Parameters in GSAS-II contain values that are used in diffraction computations. These include atom positions, but also factors that affect peak shapes or compensate for physical effects such as absoption. Many, but not all, can be optimized. The term variables is intended to refer to parameters that are being optimized in GSAS-II, but this usage is not always applied consistently within GSAS-II and in places the term variables may be applied to unvaried parameters.

Parameter in GSAS-II are uniquely 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 Ai and Djk; where *K* is one of the characters a, b or c.
α	Lattice parameter, α, computed from both Ai and Djk.
β	Lattice parameter, β, computed from both Ai and Djk.
γ	Lattice parameter, γ, computed from both Ai and Djk.
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.
dA*M* (example: `dAx`)	Refined change to atomic coordinate, *M*; where *M* is one of the characters x, y or z.
A*M* (example: `Ax`)	Fractional atomic coordinate, *M*; where *M* is one of the characters x, y or z.
AUiso	Atomic isotropic displacement parameter.
AUN_0N_1 (example: `AU11`)	Atomic anisotropic displacement parameter UN_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 valence radial fxn, *O*; where *O* is one of the characters 0, - or 6.
O“ (example: `0"`)	Atomic orbital softness for deformation radial fxn, 1; where *O* is one of the characters 0, - or 6.
ANe*P* (example: `ANe0`)	Atomic <j0> orbital population for orbital, *P*; where *P* is one of the characters 0 or 1.

continues on next page

Table 1 – continued from previous page

<var>	usage
ADO_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.
ADO_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.
AUVmat	Atomic orbital orientation matrix, 1.
ARadial	Atomic radial function, 1.
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.
BkPksig;J (example: `BkPksig;11`)	Background peak #J Gaussian width; where J is the background peak number.
BkPkgam;J (example: `BkPkgam;11`)	Background peak #J Cauchy width; where J 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.
DN_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;J (example: `Mustrain;11`)	Microstrain coefficient (delta Q/Q x 10**6); where J 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;J (example: `Size;11`)	Crystallite size value (in microns); where J 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.
Ma	microED dynamic scattering coeff A.
Mb	microED dynamic scattering coeff B.
Mc	microED dynamic scattering coeff C.
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.

continues on next page

Table 1 – continued from previous page

<var>	usage
Zero	Debye-Scherrer zero correction.
Shift	Bragg-Brentano sample displ.
SurfRoughA	Bragg-Brentano surface roughness A.
SurfRoughB	Bragg-Brentano surface roughness B.
Transparency	Bragg-Brentano sample tranparency.
DebyeA	Debye model amplitude.
DebyeR	Debye model radius.
DebyeU	Debye model Uiso.
RBVJ (example: `RBV11`)	Vector rigid body parameter.
RBVOU (example: `RBVOa`)	Vector rigid body orientation parameter U; where U is one of the characters a, i, j or k.
RBVPM (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.
RBV$V_0W_0W_1$ (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.
RBROU (example: `RBROa`)	Residue rigid body orientation parameter U; where U is one of the characters a, i, j or k.
RBRPM (example: `RBRPx`)	Residue rigid body M position parameter; where M is one of the characters x, y or z.
RBRTr;J (example: `RBRTr;11`)	Residue rigid body torsion parameter.
RBRf	Residue rigid body site fraction.
RBR$V_0W_0W_1$ (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.
RBSOU (example: `RBSOa`)	Spinning rigid body orientation parameter U; where U is one of the characters a, i, j or k.
RBSPM (example: `RBSPx`)	Spinning rigid body M position parameter; where M is one of the characters x, y or z.
RBSSh;[0-9];RJ (example: `RBSSh;[0-9];R11`)	Spinning rigid body shell radius.
RBSSh;[0-9];CJ (example: `RBSSh;[0-9];C11`)	Spinning rigid body sph. harmonics term.
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.
mVH (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.
Tmax (example: `Xmax`)	ZigZag/Block max value for T; where T is one of the characters X, Y or Z.
Tsin (example: `Xsin`)	Sin position wave for T; where T is one of the characters X, Y or Z.
Tcos (example: `Xcos`)	Cos position wave for T; where T is one of the characters X, Y or Z.

continues on next page

Table 1 – continued from previous page

<var>	usage
$U_{N_0N_1}$sin (example: `U11sin`)	Sin thermal wave for $U_{N_0N_1}$; where N_0 is one of the characters 1, 2 or 3 and N_1 is one of the characters 1, 2 or 3.
$U_{N_0N_1}$cos (example: `U11cos`)	Cos thermal wave for $U_{N_0N_1}$; where N_0 is one of the characters 1, 2 or 3 and N_1 is one of the characters 1, 2 or 3.
MTsin (example: `MXsin`)	Sin mag. moment wave for T; where T is one of the characters X, Y or Z.
MTcos (example: `MXcos`)	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.
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.
eX_0X_1 (example: `e11`)	strain tensor eX_0X_1; where X_0 is one of the characters 1 or 2 and X_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-Y (example: `beta-0`)	TOF/Pink profile term; where Y is one of the characters 0, 1 or q.
sig-Z (example: `sig-0`)	TOF profile term; where Z is one of the characters 0, 1, 2 or q.
dif$a$ (example: `difA`)	TOF to d-space calibration; where a is one of the characters A, B or C.
CG_0,G_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.

continues on next page

Table 1 – continued from previous page

<var>	usage
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.

3.3 GSAS-II Data Tree

A GSAS-II project is stored in a data file and is loaded into a wxPython data tree (wx.TreeCtrl) defined by GSASIIctrlGUI.G2TreeCtrl. Each entry in the tree has a text label and a data object associated with it. Note that all information used in a GSAS-II project is stored in the data tree, with the exception of images (which are too large). For images, a reference to the file location is saved and images are loaded from the file when needed.

To save a GSAS-II project, routine GSASIImiscGUI.ProjFileSave() is used to convert the tree contents to a "flat" format and write it to a file. The tree is transversed, and for each first-level tree item, a list is created, where the first item in that list is a two-element list containing the label of the tree item and the data object associated with the label. If there are second-level tree items that are children of that first-level tree item, additional items are added to the outermost list with pairs of text labels and data objects. Finally the outermost list is converted to a binary representation and written to disk with the Python pickle function. Note that GSAS-II does not use any data tree items other than first-level and second-level. Routine GSASIImiscGUI.ProjFileOpen() is used to read a GSAS-II project file and populate the data tree. GSAS-II project files are written with the .gpx extension.

Two pointers are kept for a selected tree entry in the GSAS-II data tree, saved as class variables in GSASIIdataGUI.GSASII (often referenced as G2frame). These are G2frame.PickId, which points to the selected data tree item, and G2frame.PatternId, which points to the parent of the data tree item, when G2frame.PickId points to a histogram. The two pointer may be the same when the first-level tree item for a histogram is selected.

3.3.1 Constraints Tree Item

Constraints are stored in a dict, separated into groups. Note that parameter are named in the following pattern, p:h:<var>:n, where p is the phase number, h is the histogram number <var> is a variable name and n is the parameter number. If a parameter does not depend on a histogram or phase or is unnumbered, that number is omitted. Note that the contents of each dict item is a List where each element in the list is a *constraint definition objects*. The constraints in this form are converted in 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 :h:<var>:n.
HAP	This specifies a list of constraints on parameters that are defined for every histogram in each phase and are of form p:h:<var>:n.
Phase	This specifies a list of constraints on phase parameters, which will be of form p::<var>:n.
Global	This specifies a list of constraints on parameters that are not tied to a histogram or phase and are of form ::<var>:n

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 and all other equivalent variables in the list are generated from that variable, using the appropriate multipliers.
  - 'c' defines a constraint equation of form, $m_1 \times var_1 + m_2 \times var_2 + ... = c$
  - 'h' defines a variable to hold (not vary). Any variable on this list is not varied, even if its refinement flag is set. Only one [mult,var] pair is allowed in a hold constraint and the mult value is ignored. This is of particular value when needing to hold one or more variables where a single flag controls a set of variables such as, coordinates, the reciprocal metric tensor or anisotropic displacement parameter.
  - 'f' defines a new variable (function) according to relationship $newvar = m_1 \times var_1 + m_2 \times var_2 + ...$

3.3.2 Covariance Tree Item

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

key	sub-key	explanation
newCellDict		(dict) lattice parameters computed by `GSASIIstrMath.GetNewCellParms()`
title		(str) Name of gpx file
variables		(list) Values for refined variables (list of float values, length N, ordered to match varyList)
sig		(list) Standard uncertainty values for refined variables (list of float values, length N, ordered to match varyList)
varyList		(list of str values, length N) List of directly refined variables
varyListStart		(list) initial refined variables before dependent vars are removed
newAtomDict		(dict) atom position values computed in `GSASIIstrMath.ApplyXYZshifts()`
Lastshft		(list) The shifts applied to each variable in the last refinement run. (list of float values, length N, ordered to match varyList)
depSigDict		(dict) Values along with standard uncertainty values for dependent variables
covMatrix		(np.array) The (NxN) covVariance matrix
freshCOV		(bool) indicates if the covMatrix has been freshly computed
msg		Warning/error messages from the last refinement run
Rvals		(dict) R-factors, GOF, Marquardt value for last refinement cycle
	Nobs	(int) Number of observed data points
	Nvars	(int) Number of refined parameters
	Rwp	(float) overall weighted profile R-factor (%)
	chisq	(float) $\sum w * (I_{obs} - I_{calc})^2$ for all data. Note: this is what GSAS-II calls χ^2, which is not the same thing as the reduced χ^2.
	lamMax	(float) Marquardt value applied to Hessian diagonal
	GOF	(float) The goodness-of-fit, aka square root of the reduced χ^2 squared, after refinement.
	GOF0	(float) The goodness-of-fit, aka square root of the reduced χ^2 square, before refinement.
	lastShifts	(dict) values of the shifts applied in the last refinement cycle (note: differs from *Lastshft*, which has values from the last run).
	SVD0	(int) number of singular value decomposition (SVD) singularities
	converged	(bool) True if last refinement run converged
	DelChi2	(float) change in χ^2 in last refinement cycle
	RestraintSum	(float) sum of restraints
	RestraintTerms	(float) total number of restraints
	Max shft/sig	(float) maximum shift/s.u. for shifts applied in last refinement run

3.3.3 Phase Tree Items

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

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

continues on next page

Table 2 – continued from previous page

key	sub-key	explanation
	AtomTypes	(llist 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
	DisAglCtls	(dict) with distance/angle search controls, see *Controls used for Distance/Angle computation*.
	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 *space group (SGData)* object, as defined in `GSASIIspc.SpcGroup()`.
	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 *Atom Records* description.
Drawing		(dict) Display parameters
	Atoms	(list of lists) with an entry for each atom that is drawn

continues on next page

Table 2 – continued from previous page

key	sub-key	explanation
	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 $e/Å^3$
	contourMax	(float) map contour maximum
	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 transofmation 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 dsplacement

continues on next page

Table 2 – continued from previous page

key	sub-key	explanation
	G2parentCoords	(list) full set of parent structure coordinates transformed to child structure; starting basis for mode displacements
	G2VarList	(list)
	IsoVarList	(list)
	G2coordOffset	(list) only adjustible 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
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: 0. Type, one of 'isotropic', 'uniaxial', '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. 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

continues on next page

Table 2 – continued from previous page

key	sub-key	explanation
	Size	List of crystallite size parameters, in order: 0. (str) Type, one of 'isotropic', 'uniaxial', '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

3.3.4 Rigid Body Objects

Rigid body descriptions are available for two types of rigid bodies: 'Vector' and 'Residue'. Vector rigid bodies are developed by a sequence of translations each with a refinable magnitude and Residue rigid bodies are described as Cartesian coordinates with defined refinable torsion angles.

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
	useCount	(int)Number of times rigid body is used in any structure
RBIds		(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

3.3.5 Space Group Objects

Space groups are interpreted by `GSASIIspc.SpcGroup()` and the information is placed in a SGdata object which is a dict with these keys. 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: `np.array([[0, 0, 0]])`
SGFixed	(bool) Only True if phase mported 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
SGLatt	(str)Lattice centering type. Will be one of P, A, B, C, I, F, R
SGLaue	(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 `[[M1,T1], [M2,T2],...]` 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 operatiors (+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

3.3.6 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 *Atom Records* description.
Drawing	Parameters that determine how the phase is displayed, including a list of atoms to be included, as described in the *Drawing Atom Records* 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 *Rigid Body Insertions*. Note that the rigid bodies are defined as *Rigid Body Objects*
RMC	PDF modeling parameters
Pawley ref	Pawley refinement parameters

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
cia+8, -1	(int) unique atom identifier
-1	(dict) wave info (modulated structures only)

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 (RBG 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

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 *Rigid Body Objects*
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 *Rigid Body Object*

3.3.7 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 [[Ld,Hd],[L,H]] 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, *J. Appl. Cryst*, **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-postions (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

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 *degrees*
	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)

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
	X,Y,Z	(float) Lorentzian profile coefficients
Instrument Parameters[1]	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.

3.3.8 Powder Reflection Data Structure

The data tree entry for powder diffraction histograms contains an entry labeled `Reflection Lists` containing a dict keyed by phase name, for every phase linked to the histogram. Each entry is itself a dict with four entries, with keys:

key	explanation
RefList	This contains the reflection list, as described below.
FF	Contains a dict with two entries, `El` which contains a list of `n` element types and `FF` which contains a 55 x `n` np.array of of form factor values.
Type	Contains a string specifying the type of histogram, such as 'PXC'
Super	Contains a bool value, which is True when the phase has a superspace spacegroup (3+1 dimension).

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
3	multiplicity
4	d-space, Å
5	pos, two-theta
6	sig, Gaussian width
7	gam, Lorenzian width
8	F^2_{obs}
9	F^2_{calc}
10	reflection phase, in degrees
11	intensity correction for reflection, this times F^2_{obs} or F^2_{calc} gives Iobs or Icalc
12	Preferred orientation correction
13	Transmission (absorption correction)
14	Extinction correction

Note that when the `Super` entry in the phase's main dict is True, indicating that the phase is a 3+1 super-space group, the columns are:

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

3.3.9 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

3.3.10 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^2_{obs}
6	7	$\sigma(F^2_{obs})$
7	8	F^2_{calc}
8	9	$F^2_{obs}(T)$
9	10	$F^2_{calc}(T)$
10	11	reflection phase, in degrees
11	12	intensity correction for reflection, this times F^2_{obs} or F^2_{calc} 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.

3.3.11 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. Accomodates 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 `ImageCalibrants()` and UserCalibrants.py (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/Ib for a point selected in a diffraction ring for calibration calculations. See pixLimit 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.
	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.
	IOtth	(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.

continues on next page

Table 3 – continued from previous page

key	sub-key	explanation
	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 `GSASIIimage.ImageLocalMax()` and subject to cutoff in `GSASIIimage.makeRing()`. Locations are used to construct rings of points for calibration calcualtions.
	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 experimnents, 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.
	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.

continues on next page

Table 3 – continued from previous page

key	sub-key	explanation
	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.

3.3.12 Controls used for Distance/Angle computation

Two arrays are used as input to `GSASIIstrMain.RetDistAngle()` and `GSASIIstrMain.PrintDistAngle()`, DisAglCtls and DisAglData.

- DisAglCtls is a dict with has keys `Name`, `AtomTypes`, `BondRadii`, `AngleRadii` which are atomic radii to be used in computation of distances. Also contains `Factors`, which is a 2 element list with a multiplier for bond and angle search range [typically (0.85,0.85)]. The maximum search distance is the product of the two radii and the multiplier, so raising the multiplier increases the number of distances or angles that are located. Example:

```
{'Name': 'Example',
'Factors': [0.85, 0.85],
'AtomTypes': ['Co', 'C', 'N', 'O', 'H'],
'BondRadii': [2.2, 1.12, 1.08, 1.09, 0.5],
'AngleRadii': [1.25, 0.92, 0.88, 0.89, 0.98]}
```

- DisAglData is a dict containing phase & refinement data:
  - 'OrigAtoms' and 'TargAtoms' contain the atoms to be used for distance/angle origins and atoms to be used as targets.
  - 'OrigIndx' contains the index numbers for the Origin atoms.
  - 'SGData' has the space group information (see *Space Group object*)
  - 'pId' has the phase id
  - 'Cell' has the unit cell parameters and cell volume
  - 'covData' has the contents of Covariance data tree item

  Added for use with rigid bodies:
  - 'RBlist' has the index numbers for atoms in a rigid body
  - 'rigidbodyDict' the contents of the main Rigid Body data tree item

- 'Phases' has the phase information for all used phases in the data tree. Only the current phase is needed, but this is easy.
- 'parmDict' is the GSAS-II parameter dict

3.4 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 `GSASIIfiles.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).

3.5 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 suymmetry 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 specifed 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.

3.6 ISODISTORT implementation

CIFs prepared with the ISODISTORT web site https://iso.byu.edu/isotropy.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.

3.6.1 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 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.

3.6.2 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 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.

3.6.3 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.

3.7 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.
3. 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.
4. 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.SaveUsedHistogramsAndPhases()` `GSASIIstrIO.SaveUpdatedHistogramsAndPhases()` `GSASIIstrIO.SetSeqResult()` `GSASIIstrMain.dropOOBvars()` `GSASIIscriptable.G2Project.set_Controls()` `GSASIIscriptable.G2Project.get_Frozen()` `GSASIIscriptable.G2Project.set_Frozen()`

CHAPTER

FOUR

GSASII: GSAS-II GUI

4.1 Script G2.py

File *G2.py* can be used to start the GSAS-II graphical user interface (GUI), particularly when GSAS-II has been installed in a location outside of Python and thus requires changing the Python path. When GSAS-II is installed in a location that is on the default Python path, this command is sufficient to start the GSAS-II GUI:

python -c "from GSASII.GSASIIGUI import main; main()"

The *G2.py* script checks to see if GSAS-II is on the path. If not, the directory where the *G2.py* file is located is placed into the Python path. At this point the func:*GSASIIGUI.main* routine is called to start the GSAS-II GUI.

4.2 Module GSASIIGUI.py

The *GSASIIGUI.py* script imports GSASIIpath, which does some minor initialization and then (before any wxPython calls can be made) creates a wx.App application. At this point `GSASIIpath.SetBinaryPath()` is called to establish the directory where GSAS-II binaries are found. If the binaries are not installed or are incompatible with the OS/Python packages, the user is asked if they should be updated from the subversion site. The wxPython app is then passed to `GSASIIdataGUI.GSASIImain()`, which creates the GSAS-II GUI and finally the event loop is started. GSASIIGUI provides a short file that is used to start the GSAS-II GUI. It is usually called from *G2.py* but this routine can also be invoked directly when GSAS-II has been installed into Python. On MacOS, a class, `G2App`, is defined inside *`main()`* that creates a wxPython application. For other platforms `wx.App()` is called directly.

`GSASII.GSASIIGUI.`**`main`**`()`

This routine is called to start the GSAS-II GUI

4.3 Keyboard Menu Shortcuts

Shortcuts for commonly-used menu commands are created by adding a menu command with a "\tCtrl+" addition such as:

```
item = parent.Append(wx.ID_ANY,'&Refine\tCtrl+R','Perform a refinement')
```

This will allow the above menu command to be executed with a "Control-R" keyboard command (on MacOS this will be "Command+R" rather than "Control-R") as well as using the menu to access that action. The following table lists the keyboard letters/numbers that have GSAS-II assigned actions. are system assigned. Note that there are also plotting keyboard commands that are implemented in `GSASIIplot`. These can be discovered from the "K" button on the plot menu bar, as they vary depending on the type of plot.

key	explanation
O	Open project (File menu)
E	Reopen recent (File menu)
S	Save project (File menu)
B	Project browser (File menu)
Q	Quit (File menu). This is system assigned on MacOS
F4	Quit (File menu). This is system-assigned on Windows
L	View LS parms (Calculate menu)
R	Refine/Sequential Refine (Calculate menu)
I	Parameter Impact (Calculate menu)
U	Check for updates (Help menu)
T	Tutorials (Help menu)
F1	Help on current tree item (Help menu). This is system-assigned
P	Peakfit (Peak Fitting menu, requires selection of Histogram Peak)
M	Minimize GSAS-II windows (MacOS Windows menu). This is system-assigned

CHAPTER

FIVE

GSASIIOBJ: DATA OBJECTS & DOCS

This module defines many data structures used in GSAS-II, as well as provides misc. support routines for accessing them.

5.1 GSASIIobj Classes and routines

Classes and routines defined in `GSASIIobj` follow.

`GSASII.GSASIIobj.`**`AddPhase2Index`**(*rdObj*, *filename*)

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

`GSASII.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()*

`GSASII.GSASIIobj.`**`AtomRanIdLookup = {}`**

dict listing for each phase the atom sequential index keyed by atom random Id; best to access this using *LookupAtomId()*

`GSASII.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 (*Parameter 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.

`GSASII.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

GSASII.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 GSASII.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

Parameters
parmDict (*dict*) – a dict of values, items not in use are ignored

UpdateVars(*varList*, *valList*)

Update the dict for the expression with a set of values

Parameters

- **varList** (*list*) – a list of variable names
- **valList** (*list*) – 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 GSASII.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)

GSASII.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 GSASII.GSASIIobj.**G2Exception**(*msg*)

A generic GSAS-II exception class

__init__(*msg*)

__str__()

Return str(self).

__weakref__

list of weak references to the object

exception GSASII.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 GSASII.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

GSASII.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.

GSASII.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

GSASII.GSASIIobj.**HistIdLookup = {}**

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

GSASII.GSASIIobj.**HistRanIdLookup = {}**

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

GSASII.GSASIIobj.**HowDidIgetHere**(*wherecalledonly=False*)

Show a traceback with calls that brought us to the current location. Used for debugging.

Parameters

wherecalledonly (*bool*) – When True, the entire calling stack is shown. When False (default), only the 2nd to last stack entry (the routine that called the calling routine is shown.

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

Defines a base class for the reading of input files (diffraction data, coordinates,…). See *Writing a Import Routine* for an explanation on how to use a subclass of this class.

CIFValidator(*filepointer*)

A *ContentsValidator()* for use to validate CIF files.

ContentsValidator(*filename*)

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

ExtensionValidator(*filename*)

This methods checks if the file has the correct extension

Returns

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

exception ImportException

Defines an Exception that is used when an import routine hits an expected error, usually in .Reader.

Good practice is that the Reader should define a value in self.errors that tells the user some information about what is wrong with their file.

__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 GSASII.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 `GSASIImiscGUI.ReadImages()`, which puts image info directly into the data tree.
- Unlike all other data types read by GSAS-II, images are only kept in memory as they are used and function `GSASIIfiles.GetImageData()` or `GSASIIfiles.RereadImageData()` is used to reread images if they are reloaded. For quick retrieval of previously read images, it may be useful to save sums of images or save a keyword (see `ImageTag`, below

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 used:
  - `pixelSize`: size of each pixel (x,y) 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]`, in mm measured from top left corner of the detector
  - `size`: size of image in pixels (x,y) (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
  - `PolaVal`: has two values, the polarization fraction (typically 0.95-0.99 for synchrotrons, 0.5 for lab instruments) and a refinement flag (such as `[0.99, False]`).
  - `setdist`: nominal distance from sample to detector. Note that `distance` may be changed during calibration, but `setdist` will not be, so that calibration may be repeated.

optional data items:

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

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

InitParameters()

initialize the instrument parameters structure

LoadImage(*ParentFrame*, *imagefile*, *imagetag=None*)

Optionally, call this after reading in an image to load it into the tree. This saves time by preventing a reread of the same information.

ReInitialize()

Reinitialize the Reader to initial settings – not used at present

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

class GSASII.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 GSASII.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 GSASII.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 GSASII.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 GSASII.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 GSASII.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*)

GSASII.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?

GSASII.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)

GSASII.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)

GSASII.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)

GSASII.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)

GSASII.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)

GSASII.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)

GSASII.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)

GSASII.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

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

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

GSASII.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.

Used for CIF imports and for reading CIF templates for project CIF exports

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 or an empty string if PyCifRW is not accessible

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

GSASII.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.]

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

GSASII.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 GSASII.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

GSASII.GSASIIobj.**SortVariables**(*varlist*)

Sorts variable names in a sensible manner

GSASII.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

GSASII.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.

GSASII.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.

GSASII.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*

GSASII.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

GSASII.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

GSASII.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.

GSASII.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.

GSASII.GSASIIobj.**patchControls**(*Controls*)

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

GSASII.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.

GSASII.GSASIIobj.**reVarDesc = {}**

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

GSASII.GSASIIobj.**reVarStep = {}**

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

GSASII.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

GSASII.GSASIIobj.**restraintNames = [['Bond', 'Bonds'], ['Angle', 'Angles'], ['Plane', 'Planes'], ['Chiral', 'Volumes'], ['Torsion', 'Torsions'], ['Rama', 'Ramas'], ['ChemComp', 'Sites'], ['Texture', 'HKLs'], ['Moments', 'Moments'], ['General', 'General'], ['SpinRB', 'SpinRBs']]**

Names of restraint keys for the restraint dict and the location of the restraints in each dict

GSASII.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.

CHAPTER

SIX

GSAS-II UTILITY MODULES

6.1 GSASIIpath: locations & updates

Routines for dealing with file locations, etc.

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

Interfaces with git and gets version number.

Accesses configuration options, as defined in config.py

6.1.1 GSASIIpath Classes & Routines

`GSASIIpath` Classes & routines follow

`GSASII.GSASIIpath.`**`AddConfigValue`**(*valsdict*)

Set configuration variables.

Parameters
`valsdict` (*dict*) – a dictionary of values that are added directly to configDict.

`GSASII.GSASIIpath.`**`DoNothing`**()

A routine that does nothing. This is called in place of IPyBreak and pdbBreak except when the debug option is set True in the configuration settings

`GSASII.GSASIIpath.`**`GetBinaryPrefix`**(*pyver=None*)

Creates the first part of the binary directory name such as linux_64_p3.9 (where the full name will be linux_64_p3.9_n1.21).

Note that any change made here is also needed in GetBinaryDir in fsource/SConstruct or GSASII-buildtools/compile/nameTar.py

`GSASII.GSASIIpath.`**`GetConfigDefault`**(*key*)

Return the default value for a config value

Parameters
`key` (*str*) – a value to be found in the configuration (config_example.py) file

Returns
the default value or None

`GSASII.GSASIIpath.`**`GetConfigValue`**(*key*, *default=None*, *getDefault=False*)

Return the configuration file value for key or a default value if not specified.

Parameters

- **`key`** (*str*) – a value to be found in the configuration settings

- **default** (*any*) – a value to be supplied if a value for key is not specified in the config file or the config file is not found. Defaults to None.
- **getDefault** (*bool*) – If True looks up the default value from the config_example.py file (default value is False). Do not specify a getDefault=True if a value is provided for default.

Returns
the value found or the default.

GSASII.GSASIIpath.**GetRepoUpdatesInBackground**()

Get the latest GSAS-II version info. This serves to make sure that *gitGetUpdate()* is called only if git has been used to install GSAS-II.

Returns
returns a Popen object (see subprocess)

GSASII.GSASIIpath.**GetVersionNumber**()

Obtain a numeric (sequential) version number for GSAS-II from version files, or directly from git if no other choice.

Returns
an int value normally, but unexpected error conditions could result in a value of 'unknown' or '?'.

GSASII.GSASIIpath.**GetVersionTag**()

Obtain a release (X.Y.Z) version number for GSAS-II from version files, or directly from git if no other choice.

Returns
a string of form <Major>.<Minor>.<mini> normally, but unexpected error conditions could result in a value of '?.?.?' or '?'.

GSASII.GSASIIpath.**HowIsG2Installed**()

Determines if GSAS-II was installed with git. Result is cached to avoid time needed for multiple calls of this.

Returns

- a string starting with 'git' from git, if installed from the GSAS-II GitHub repository (defined in g2URL), the string is 'github', if the post-3/2024 directory structure is in use '-rev' is appended.
- or 'noVCS' if installed without a connection to a version control system

GSASII.GSASIIpath.**IPyBreak**()

A routine that does nothing. This is called in place of IPyBreak and pdbBreak except when the debug option is set True in the configuration settings

GSASII.GSASIIpath.**IPyBreak_base**(*userMsg=None*)

A routine that invokes an IPython session at the calling location. Called by a call to *IPyBreak()* or command `breakpoint()`. This routine is only used when debug=True is set in the configuration settings.

GSASII.GSASIIpath.**InstallGitBinary**(*tarURL*, *instDir*, *nameByVersion=False*, *verbose=True*)

Install the GSAS-II binary files into the location specified.

Parameters

- **tarURL** (*str*) – a URL for the tar file.
- **instDir** (*str*) – location directory to install files. This directory may not exist and will be created if needed.
- **nameByVersion** (*bool*) – if True, files are put into a subdirectory of *instDir*, named to match the tar file (with plaform, Python & numpy versions). Default is False, where the binary files are put directly into *instDir*.
- **verbose** (*bool*) – if True (default), status messages are printed.

Returns
None

GSASII.GSASIIpath.**InvokeDebugOpts**()

Called to set up debug options

GSASII.GSASIIpath.**LoadConfig**(*printInfo=True*)

Read configuration settings from ~/.GSASII/config.ini, if present. Place the values into global dict configDict.

Parameters
printInfo (*bool*) – if printInfo is True (default) then a message is shown with the number of settings read (upon startup).

GSASII.GSASIIpath.**LoadConfigFile**(*filename*)

Read a GSAS-II configuration file. Comments (starting with "%") are removed, as are empty lines

Parameters
filename (*str*) – base file name (such as 'file.dat'). Files with this name are located from the path and the contents of each are concatenated.

Returns
a list containing each non-empty (after removal of comments) line found in every matching config file.

GSASII.GSASIIpath.**MacRunScript**(*script*)

Start a bash script in a new terminal window. Used on Mac OS X only.

Parameters
script (*str*) – file name for a bash script

GSASII.GSASIIpath.**SetBinaryPath**(*showConfigMsg=False*)

Add location of GSAS-II shared libraries (binaries: .so or .pyd files) to path (when needed). When GSAS-II is installed by pixi, no change in the path is needed.

This routine must be executed after GSASIIpath is imported and before any other GSAS-II imports are done, since they may assume binary files are in path

Parameters
showConfigMsg (*bool*) – When True, config info is shown (default is False)

GSASII.GSASIIpath.**SetConfigValue**(*parmdict*)

Set configuration variables. Note that parmdict is a dictionary from `GSASIIctrlGUI.GetConfigValsDocs()` where each element is a list. The first item in list is the default value, the second is the value to use for that configuration variable. Most of the information gathered in GetConfigValsDocs is no longer used.

GSASII.GSASIIpath.**TestSPG**()

Test if pyspg.[so,.pyd] can be run from a location in the existing path Do not modify the path if not.

GSASII.GSASIIpath.**WriteConfig**(*configDict*)

Write the configDict information to the GSAS-II ini settings into file ~/.GSASII/config.ini. Called from `GSASIIctrlGUI.SaveConfigVars()`.

GSASII.GSASIIpath.**addCondaPkg**()

Install the conda API into the current conda environment using the command line, so that the API can be used in the current Python interpreter

Attempts to do this without a shell failed on the Mac because it seems that the environment was inherited; seems to work w/o shell on Windows.

GSASII.GSASIIpath.**addPrevGPX**(*fil*, *cDict*)

Add a GPX file to the list of previous files. Move previous names to start of list. Keep most recent five files

GSASII.GSASIIpath.**commonPath**(*dir1*, *dir2*)

Check if two directories share a path. Note that paths are considered the same if either directory is a subdirectory of the other, but not if they are in different subdirectories /a/b/c shares a path with /a/b/c/d but /a/b/c/d and /a/b/c/e do not.

Returns
True if the paths are common

GSASII.GSASIIpath.**condaEnvCreate**(*envname*, *packageList*, *force=False*)

Create a Python interpreter in a new conda environment. Use this when there is a potential conflict between packages and it would be better to keep the packages separate (which is one of the reasons conda supports environments). Note that conda should be run from the base environment; this attempts to deal with issues if it is not.

Currently, this is used only to install diffpy.PDFfit2.

Parameters

- **envname** (*str*) – the name of the environment to be created. If the environment exists, it will be overwritten only if force is True.
- **packageList** (*list*) – a list of conda install create command options, such as:

```
['python=3.7', 'conda', 'gsl', 'diffpy.pdffit2',
    '-c', 'conda-forge', '-c', 'diffpy']
```

- **force** (*bool*) – if False (default) an error will be generated if an environment exists

Returns
(status,msg) where status is True if an error occurs and msg is a string with error information if status is True or the location of the newly-created Python interpreter.

GSASII.GSASIIpath.**condaInstall**(*packageList*)

Installs one or more packages using the anaconda conda package manager. Can be used to install multiple packages and optionally use channels.

Parameters
packageList (*list*) – a list of strings with name(s) of packages and optionally conda options. Examples:

```
packageList=['gsl']
packageList=['-c','conda-forge','wxpython']
packageList=['numpy','scipy','matplotlib']
```

Returns
None if the the command ran normally, or an error message if it did not.

GSASII.GSASIIpath.**condaTest**(*requireAPI=False*)

Returns True if it appears that Python is being run under Anaconda Python with conda present. Tests for conda environment vars and that the conda package is installed in the current environment.

Returns
True, if running under Conda

GSASII.GSASIIpath.**countDetachedCommits**(*g2repo=None*)

Count the number of commits that have been made since a commit that is containined in the main branch

returns the count and the commit object for the parent commit that connects the current stranded branch to the main branch.

None is returned if no connection is found

GSASII.GSASIIpath.**dirGitHub**(*dirlist*, *orgName='AdvancedPhotonSource'*, *repoName='GSAS-II-Tutorials'*)

Obtain a the contents of a GitHub repository directory using the GitHub REST API.

Parameters

- **dirlist** (*str*) – a list of sub-directories *['parent','child',sub']* for *parent/child/sub* or *[]* for a file in the top-level directory.
- **orgName** (*str*) – the name of the GitHub organization
- **repoName** (*str*) – the name of the GitHub repository

Returns

a list of file names or None if the dirlist info does not reference a directory

examples:

```
dirGitHub([], 'GSASII', 'TutorialTest')
dirGitHub(['TOF Sequential Single Peak Fit', 'data'])
```

The first example will get the contents of the top-level directory for the specified repository

The second example will provide the contents of the "TOF Sequential Single Peak Fit"/data directory.

GSASII.GSASIIpath.**downloadDirContents**(*dirlist*, *targetDir*, *orgName='AdvancedPhotonSource'*, *repoName='GSAS-II-Tutorials'*)

Download the entire contents of a directory from a repository on GitHub. Used to download data for a tutorial.

GSASII.GSASIIpath.**exceptHook**(**args*)

A routine to be called when an exception occurs. It prints the traceback with fancy formatting and then calls an IPython shell with the environment of the exception location.

This routine is only used when debug=True is set in the configuration settings.

GSASII.GSASIIpath.**findConda**()

Determines if GSAS-II has been installed as g2conda or gsas2full with conda located relative to this file. We could also look for conda relative to the python (sys.executable) image, but I don't want to muck around with python that someone else installed.

GSASII.GSASIIpath.**fullsplit**(*fil*, *prev=None*)

recursive routine to split all levels of directory names

GSASII.GSASIIpath.**getG2Branch**()

Get name of current branch, as named on local computer

GSASII.GSASIIpath.**getG2VersionInfo**()

Get the git version information. This can be a bit slow, so reading …/GSASII/saved_version.py may be faster

GSASII.GSASIIpath.**getGitBinaryLoc**(*npver=None*, *pyver=None*, *verbose=True*, *debug=False*)

Identify the best GSAS-II binary download location from the distributions in the latest release section of the github repository on the CPU platform, and Python & numpy versions. The CPU & Python versions must match, but the numpy version may only be close.

Parameters

- **npver** (*str*) – Version number to use for numpy, if None (default) the version is taken from numpy in the current Python interpreter.

- **pyver** (*str*) – Version number to use for Python, if None (default) the version is taken from the current Python interpreter.
- **verbose** (*bool*) – if True (default), status messages are printed

Returns
a URL for the tar file (success) or None (failure)

GSASII.GSASIIpath.**getGitBinaryReleases**(*cache=False*)

Retrieves the binaries and download urls of the latest release

Parameters
cache (*bool*) – when cache is True and the binaries file names are retrieved (does not always succeed when done via GitHub Actions), the results are saved in a file for reuse should the retrieval fail. Default is False so the file is not changed.

Returns

a URL dict for GSAS-II binary distributions found in the newest release in a GitHub repository. The repo location is defined in global *G2binURL*.

The dict keys are references to binary distributions, which are named as f"{platform}_p{pver}_n{npver}" where platform is determined in `GSASIIpath.GetBinaryPrefix()` (linux_64, mac_arm, win_64,…) and where *pver* is the Python version (such as "3.10") and *npver* is the numpy version (such as "1.26").

The value associated with each key contains the full URL to download a tar containing that binary distribution.

GSASII.GSASIIpath.**getGitHubVersion**()

Get the latest git version info when not accessible from git as saved by saveGitHubVersion

GSASII.GSASIIpath.**getIconFile**(*imgfile*)

Looks in either the main GSAS-II install location (old) or subdirectory icons (after reorg) for an icon

Returns
the full path for the icon file

GSASII.GSASIIpath.**getSavedVersionInfo**()

Get version number information from a file written by install routines. This is faster than getting the information from git. Also, when GSAS-II is installed into Python, files are no longer in a git repository, so querying git is not possible.

The git_verinfo.py file is written by install/incr-version.py or by install/save_versions.py The saved_version.py file was previousl written by install/save_versions.py but this file is no longer in use.

Returns
a reference to the version variables or None if no version info file is found.

GSASII.GSASIIpath.**gitCheckForUpdates**(*fetch=True*, *g2repo=None*)

Provides a list of the commits made locally and those in the local copy of the repo that have not been applied. Does not provide useful information in the case of a detached Head (see *countDetachedCommits()* for that.)

Parameters

- **fetch** (*bool*) – if True (default), updates are downloaded from the remote repository (git fetch), before checking for changes. At present, this feature is not used anywhere in GSAS-II (fetch=False in all calls).
- **g2repo** (*str*) – git.Rwpo connecton to GSAS-II installation. If None (default) it will be opened.

Returns

a list containing (remotecommits, localcommits, fetched) where

- remotecommits is a list of hex hash numbers of remote commits and
- localcommits is a list of hex hash numbers of local commits and
- fetched is a bool that will be True if the update (fetch) step ran successfully

Note that if the head is detached (GSAS-II has been reverted to an older version) or the branch has been changed, the values for each of the three items above will be None.

GSASII.GSASIIpath.**gitCountRegressions**(*g2repo=None*)

Count the number of new check ins on the main branch since the head was detached as well as any checkins made on the detached head.

Returns

maincount,detachedcount, where

- maincount is the number of check ins made on the main branch remote repository since the reverted check in was first made.
- detachedcount is the number of check ins made locally starting from the detached head (hopefully 0)

If the connection between the current head and the main branch cannot be established, None is returned for both. If the connection from the reverted check in to the newest version (I don't see how this could happen) then only maincount will be None.

GSASII.GSASIIpath.**gitGetUpdate**(*mode='Background'*)

Download the latest updates into the local copy of the GSAS-II repository from the remote main, but don't actually update the GSAS-II files being used. This can be done immediately or in background.

In 'Background' mode, a background process is launched. The results from the process are recorded in file in ~/GSASII_bkgUpdate.log (located in %HOME% on Windows). A pointer to the created process is returned.

In 'immediate' mode, the update is performed immediately. The function does not return until after the update is downloaded.

Returns

In 'Background' mode, returns a Popen object (see subprocess). In 'immediate' mode nothing is returned.

GSASII.GSASIIpath.**gitHash2Tags**(*githash=None*, *g2repo=None*)

Find tags associated with a particular git commit. Note that if *githash* cannot be located because it does not exist or is not unique, a *git.BadName* exception is raised.

Parameters

- **githash** (*str*) – hex hash code (abbreviated to as few characters as needed to keep it unique). If None (default), the HEAD is used.
- **g2repo** (*str*) – git.Rwpo connecton to GSAS-II installation. If None (default) it will be opened.

Returns

a list of tags (each a string)

GSASII.GSASIIpath.**gitHistory**(*values='tag'*, *g2repo=None*, *maxdepth=100*)

Provides the history of commits to the main, either as tags or hash values

Parameters

- **values** (*str*) – specifies what type of values are returned. If values=='hash', then hash values or for values=='tag', a list of list of tag(s).
- **g2repo** (*str*) – git.Rwpo connecton to GSAS-II installation. If None (default) it will be opened.

Returns

a list of str values where each value is a hash for a commit (values=='hash'), for values=='tag', a list of lists, where a list of tags is provided for each commit. When tags are provided, for any commit that does not have any associated tag(s), that entry is omitted from the list. for values=='both', a list of lists, where a hash is followed by a list of tags (if any) is provided

GSASII.GSASIIpath.**gitLookup**(*repo_path*, *gittag=None*, *githash=None*)

Return information on a particular checked-in version of GSAS-II.

Parameters

- **repo_path** (*str*) – location where GSAS-II has been installed
- **gittag** (*str*) – a tag value.
- **githash** (*str*) – hex hash code (abbreviated to as few characters as needed to keep it unique). If None (default), a tag must be supplied.

Returns

either None if the tag/hash is not found or a tuple with four values (hash, tag-list, message,date_time) where

- hash (str) is the git checking hash code;
- tag-list is a list of tags (typically there will be one or two);
- message is the check-in message (str)
- date_time is the check-in date as a datetime object

GSASII.GSASIIpath.**gitStartUpdate**(*cmdopts*)

Update GSAS-II in a separate process, by running this script with the options supplied in the call to this function and then exiting GSAS-II.

GSASII.GSASIIpath.**gitTag2Hash**(*gittag*, *g2repo=None*)

Provides the hash number for a git tag. Note that if *gittag* cannot be located because it does not exist or is too old and is beyond the *depth* of the local repository, a *ValueError* exception is raised.

Parameters

- **repo_path** (*str*) – location where GSAS-II has been installed.
- **gittag** (*str*) – a tag value.
- **g2repo** (*str*) – git.Rwpo connecton to GSAS-II installation. If None (default) it will be opened.

Returns

a str value with the hex hash for the commit.

GSASII.GSASIIpath.**gitTestGSASII**(*verbose=True*, *g2repo=None*)

Test a the status of a GSAS-II installation

Parameters

- **verbose** (*bool*) – if True (default), status messages are printed

- **g2repo** (*str*) – git.Rwpo connecton to GSAS-II installation. If None (default) it will be opened.

Returns

istat, with the status of the repository, with one of the following values:

- -1: path is not found
- -2: no git repository at path
- -3: unable to access repository
- value&1==1: repository has local changes (uncommitted/stashed)
- value&2==2: repository has been regressed (detached head)
- value&4==4: repository has staged files
- value&8==8: repository has has been switched to other than main branch
- value==0: no problems noted

GSASII.GSASIIpath.**makeScriptShortcut**()

Creates a shortcut to GSAS-II in the current Python installation so that "import G2script" (or "import G2script as GSASIIscripting") can be used without having to add GSASII to the path.

The new shortcut is then tested.

Returns

returns the name of the created file if successful. None indicates an error.

GSASII.GSASIIpath.**openInNewTerm**(*project=None*, *g2script=None*, *pythonapp='/home/docs/checkouts/readthedocs.org/user_builds/gsas-ii/conda/latest/bin/python'*)

Open a new and independent GSAS-II session in separate terminal or console window and as a separate process that will continue even if the calling process exits. Intended to work on all platforms.

This could be used to run other scripts inside python other than GSAS-II

Parameters

- **project** (*str*) – the name of an optional parameter to be passed to the script (usually a .gpx file to be opened in a new GSAS-II session)
- **g2script** (*str*) – the script to be run. If None (default) the G2.py file in the same directory as this file will be used.
- **pythonapp** (*str*) – the Python interpreter to be used. Defaults to sys.executable which is usually what is wanted.
- **terminal** (*str*) – a name for a preferred terminal emulator

GSASII.GSASIIpath.**pathhack_TestSPG**(*fpth*)

Test if pyspg.[so,.pyd] can be run from a specified location. If so modify the path to include it.

GSASII.GSASIIpath.**pdbBreak**()

A routine that does nothing. This is called in place of IPyBreak and pdbBreak except when the debug option is set True in the configuration settings

GSASII.GSASIIpath.**pipInstall**(*packageList*)

Installs one or more packages using the pip package installer. Use of this should be avoided if conda can be used (see *condaTest()* to test for conda). Can be used to install multiple packages together. One can use pip options, but this is probably not needed.

Parameters

packageList (*list*) – a list of strings with name(s) of packages Examples:

```
packageList=['gsl']
packageList=['wxpython','matplotlib','scipy']
packageList=[r'\Mac\Home\Scratch\wheels\pygsl-2.3.3-py3-none-any.whl']
packageList=['z:/Scratch/wheels/pygsl-2.3.3-py3-none-any.whl']
```

Returns

None if the the command ran normally, or an error message if it did not.

GSASII.GSASIIpath.**postURL**(*URL*, *postdict*, *getcookie=None*, *usecookie=None*, *timeout=None*, *retry=2*, *mode='get'*)

Posts a set of values as from a web form using the "get" or "post" protocols. If access fails to an https site, the access is retried with http.

Parameters

- **URL** (*str*) – the URL to post; typically something like 'http://www…/dir/page?'
- **postdict** (*dict*) – contains keywords and values, such as {'centrosymmetry': '0', 'crystal-system': '0', …}
- **getcookie** (*dict*) – dict to save cookies created in call, or None (default) if not needed.
- **usecookie** (*dict*) – dict containing cookies to be used in call, or None (default) if not needed.
- **timeout** (*int*) – specifies a timeout period for the get or post (default is None, which means the timeout period is set by the server). The value when specified is the time in seconds to wait before giving up on the request.
- **retry** (*int*) – the number of times to retry the request, if it times out. This is only used if timeout is specified. The default is 2. Note that if retry is left at the default value (2), The timeout is increased by 25% for the second try.
- **mode** (*str*) – either 'get' (default) or 'post'. Determines how the request will be submitted.

Returns

a string with the response from the web server or None if access fails.

GSASII.GSASIIpath.**rawGitHubURL**(*dirlist*, *filename*, *orgName='AdvancedPhotonSource'*, *repoName='GSAS-II-Tutorials'*, *branchname='main'*)

Create a URL that can be used to view/downlaod the raw version of file in a GitHub repository.

Parameters

- **dirlist** (*str*) – a list of sub-directories *['parent','child',sub']* for *parent/child/sub* or *[]* for a file in the top-level directory.
- **filename** (*str*) – the name of the file
- **orgName** (*str*) – the name of the GitHub organization
- **repoName** (*str*) – the name of the GitHub repository
- **branchname** (*str*) – the name of the GitHub branch. Defaults to "main".

Returns

a URL-encoded URL

GSASII.GSASIIpath.**runScript**(*cmds=[]*, *wait=False*, *G2frame=None*)

run a shell script of commands in an external process

Parameters

- **cmds** (*list*) – a list of str's, each ietm containing a shell (cmd.exe or bash) command
- **wait** (*bool*) – if True indicates the commands should be run and then the script should return. If False, then the currently running Python will exit. Default is False
- **G2frame** (*wx.Frame*) – provides the location of the current .gpx file to be used to restart GSAS-II after running the commands, if wait is False. Default is None which prevents restarting GSAS-II regardless of the value of wait.

GSASII.GSASIIpath.**saveGitHubVersion**()

Get the latest GSAS-II version tags from the GitHub site and place them into the config.ini file. This is always done in background so that app startup time is minimally delayed.

Returns

Returns a Popen object (see subprocess).

6.2 *config_example.py: Configuration options*

6.2.1 Configuration variables

This file contains optional configuration options for GSAS-II. The values for the variables named here will be set in file ~/.GSASII/config.ini which is read on startup by `GSASIIpath.LoadConfig()`. To check if a configuration variable has been set use `GSASIIpath.GetConfigValue()`, which returns None if the variable is not set. Values are typically changed using `GSASIIctrlGUI.SelectConfigSetting()` which uses `GSASIIctrlGUI.SaveConfigVars()` to write the ~/.GSASII/config.ini file.

To define new config variables for GSAS-II, define them here with a default value: use None or a string for strings. If an integer or real values is used as a default, the routines will ensure that this type is preserved for any user setting. Always include a doc string after defining each variable. This definition will be shown in the GUI to explain what the variable does.

If a name ends with a particular keyword, then specialized edit routines are used.

- Names ending in _location or _directory are for path items
- Names ending in _exec for executable files (.exe on windows).
- Names ending in _color for colors, to be specified as RGBA values (note that Contour_color is restricted to color maps).
- Names ending in _pos or _Size are integer tuples for wx sizes or positions.

For example:

```
int_config = 0
float_config = 0.0
string_config = None (or)
string_config = 'value'
```

Below are the configuration settings that can be used in GSAS-II.

GSASII.config_example.**Arc_mask_azimuth = 10.0**

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

GSASII.config_example.**Autoint_PollTime = 30.0**

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

GSASII.config_example.**Autoscale_ParmNames = ['userComment2', 'extraInputs\\1\\extraInputs', 'Ion_Chamber_I0']**

Gives the possible selection of incident monitor names as found in an image metadata file. Used in AutoIntegration

GSASII.config_example.**BCS_API_KEY = ''**

This contains a key that is used to access the Bilbao Crystallographic web Server. It will be generated individually for each user who wants to access the server based on specific information (likely including an e-mail address) that BCS chooses.

The mechanism for obtaining that key and getting it into the GSAS-II configuration will be established later.

GSASII.config_example.**BRML_T_col = -1**

When zero or positive, this indicates a column where the temperature is stored in a Bruker BRML file. With the default value (-1), the columns 8, 6, and 5 are tried, in that order.

GSASII.config_example.**Bkg_color = 'ff0000ff'**

The color for plotting the background powder diffraction pattern. Colors are specified as hex RGBA values, as used in Matplotlib (without preceding #). The default is ff0000ff, which sets the color to red.

GSASII.config_example.**CINEMA_DS_directory = None**

Location where CINEMA: Debye-Scherrer is installed. This is the location where the index.html file is located. If not set, if the export for this is used, you will be asked to supply the directory, which will be saved here.

GSASII.config_example.**Calc_color = '008000ff'**

The color for plotting the computed powder diffraction pattern. Colors are specified as hex RGBA values, as used in Matplotlib (without preceding #). The default is 00ff00ff, which sets the color to green.

GSASII.config_example.**Clip_on = True**

if True then line plots willl be clipped at plot border; if False line plots extend nto white space around plot frme

GSASII.config_example.**Column_Metadata_directory = None**

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

GSASII.config_example.**Contour_color = 'GSPaired'**

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

GSASII.config_example.**DefaultAutoScale = 'userComment2'**

DefaultAutoScale selects one of the AutoScale_ParmNames. Used in AutoIntegration

GSASII.config_example.**Diff_color = '00bfbfff'**

The color for plotting the obs-calc powder diffraction pattern. Colors are specified as hex RGBA values, as used in Matplotlib (without preceding #). The default is 00ffffff, which sets the color to cyan.

GSASII.config_example.**DrawAtoms_default = ''**

Allows selection of the default plotting mode for structures in Draw Atoms. The only valid values are: 'lines', 'vdW balls', 'sticks', 'balls & sticks', 'ellipsoids'. %% If a non-valid choice is used (the default) 'vdW balls' is used.

GSASII.config_example.**FontSize_incr = 0**

Specifies a point size to increase (or decrease if negative) the default font size for the GSAS-II windows. Default is 0. An increment much larger than ~4 will likely cause some places where text no longer fits, but might be useful on high resolution monitors. %% Restart GSAS-II for this setting to take effect.

GSASII.config_example.**G2FileBrowser = False**

When set to True, the GSAS-II provided file browser is used to find files when files are imported. For Linux the default is True, but for Windows and Mac, the default is False

GSASII.config_example.**G2RefinementWindow = False**

When True a custom progress window is displayed to track the progress of refinements. When False a generic wxpython supplied progress dialog is used.

GSASII.config_example.**HDF5selection = 20**

When an HDF5 file contains more than this number of images, a selection window is offered to determine which images will be read. If negative, the selection window will never be used. If zero, it will always be used.

GSASII.config_example.**Help_mode = 'browser'**

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

GSASII.config_example.**Image_2theta_max = 50.0**

Specifies a default 2-theta maximum used for calibration and integration as the Outer 2-theta value. Will be applied for newly-read images, but if changed the new value will be saved.

GSASII.config_example.**Image_2theta_min = 5.0**

Specifies a default 2-theta minimum used for calibration and integration as the Inner 2-theta value. Will be applied for newly-read images, but if changed the new value will be saved.

GSASII.config_example.**Image_calibrant = ''**

Specifies a default calibrant material for images. Will be applied for newly-read images, but if changed the specified material will be saved.

GSASII.config_example.**Import_directory = None**

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

GSASII.config_example.**Instprm_default = False**

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

GSASII.config_example.**LogAllVars = False**

When True, up to 50 refined parameters will be saved after each refinement run in the GSAS-II Notebook tree entry. This setting will be ignored if individual variables are set to be logged in the Calculate/View LS Parameters window.

GSASII.config_example.**Main_Pos = (100, 100)**

Main window location - will be updated & saved when user moves it. If position is outside screen then it will be repositioned to default. This is used internally by GSAS-II and would not normally be changed by a user.

GSASII.config_example.**Main_Size = (700, 450)**

Main window size (width, height) - initially uses wx.DefaultSize but will updated and saved as the user changes the window. This is used internally by GSAS-II and would not normally be changed by a user.

GSASII.config_example.**Movie_fps = 10**

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

GSASII.config_example.**Movie_time = 5**

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

GSASII.config_example.**Multiprocessing_cores = 0**

Specifies the number of cores to use when performing multicore computing. A number less than zero causes the recommended number of cores [using multiprocessing.cpu_count()/2] to be used. Setting this number to 0 or 1 avoids use of the multiprocessing module: all computations are performed in-line.

GSASII.config_example.**Obs_color = '0000ffff'**

The color for plotting the observed powder diffraction pattern. Colors are specified as hex RGBA values, as used in Matplotlib (without preceding #). The default is 0000ffff, which sets the color to blue.

GSASII.config_example.**PDF_Rmax = 100.0**

Maximum radius for G(r) calculations: range is from 10-200A; default is 100A

GSASII.config_example.**Plot_Pos = (200, 200)**

Plot window location - will be updated & saved when user moves it these widows. If position is outside screen then it will be repositioned to default. This is used internally by GSAS-II and would not normally be changed by a user.

GSASII.config_example.**Plot_Size = (700, 600)**

Plot window size (width, height) - initially uses wx.DefaultSize but will updated and saved as the user changes the window. This is used internally by GSAS-II and would not normally be changed by a user.

GSASII.config_example.**Ref_Colors = 'b r c g m k'**

The colors for reflection tick marks by phase. Use one of 'k'-black, 'r'-red, 'b'-blue, 'g'-green, 'm'-magenta, 'c'-cyan for the line colors, or any other valid matplotlib color name or hex code.

GSASII.config_example.**Ring_mask_thickness = 0.1**

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

GSASII.config_example.**Save_paths = False**

When set to True, the last-used path for saving of .gpx and for importing of input files is saved in the configuration file.

GSASII.config_example.**SeparateHistPhaseTreeItem = False**

When this is set to True, the parameters specific to each histogram and phase together (such as peak shapes & phase fractions) are shown as a 1st-level tree item rather than inside each Phase's Data tab. If this is changed, the current GSAS-II project will be saved and will be reloaded for the change to take effect. Default is False.

GSASII.config_example.**Show_timing = False**

If True, shows various timing results.

GSASII.config_example.**SortExports = False**

When True, data tree entries (Histograms, Phases,…) are sorted alphanumerically prior to to export, when multiple entries are selected. This will matter for exporters that place multiple entries in a single file. The default is to use the data tree entries in the order they appear in the data tree.

GSASII.config_example.**Split_Loc = 250**

Location of the splitter bar between the data tree and the data window. Can be dragged by user. Is saved automatically with the window size and position

GSASII.config_example.**Spot_mask_diameter = 1.0**

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

GSASII.config_example.**Starting_directory = None**

Specifies a default location for starting GSAS-II and where .gpx files should be read from. Will be updated if Save_paths is True. Note that os.path.expanduser is run on this before it is used, so the user's home directory can be specified with a '~'.

GSASII.config_example.**Tick_length = 8.0**

Specifies the length of phase tick marks in pixels. Default is 8.

GSASII.config_example.**Tick_width = 1.0**

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

GSASII.config_example.**Transpose = False**

Set to True to cause images to be Transposed when read (for code development)

GSASII.config_example.**Tutorial_location = None**

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

```
Tutorial_location = GSASIIpath.path2GSAS2
```

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

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

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

GSASII.config_example.**debug = False**

Set to True to turn on debugging mode. This enables use of IPython on exceptions and on calls to `GSASIIpath.IPyBreak()` or breakpoint(). Calls to `GSASIIpath.pdbBreak()` will invoke pdb at that location. %% If debug is False, calls to `GSASIIpath.IPyBreak()`, breakpoint() and `GSASIIpath.pdbBreak()` are ignored. %% From inside Spyder, calls to breakpoint() invoke the Spyder debugger, independent of the setting of debug. %% Restart GSAS-II for the setting of debug to take effect.

GSASII.config_example.**enum_DrawAtoms_default = ['', 'lines', 'vdW balls', 'sticks', 'balls & sticks', 'ellipsoids']**

choices for DrawAtoms_default

GSASII.config_example.**fullIntegrate = True**

If True then full image integration is default; False otherwise

GSASII.config_example.**fullrmc_exec = None**

Defines the full path to a Python executable that has been configured with the fullrmc package. If None (the default), GSAS-II will see if fullrmc can be imported into the current Python (which is unlikely to ever work). If that does not work, GSAS-II will search for an executable named fullrmc* (or fullrmc*.exe on Windows) in the Python `sys.path` search path, which includes the GSAS-II binary directory.

GSASII.config_example.**lastUpdateNotice = 0**

Defines the version number for the last update notice that has been shown. This should not need to be changed manually.

GSASII.config_example.**pdffit2_exec = None**

Defines the full path to a Python executable that has been configured with the PDFfit2 (diffpy) package. If None (the default), GSAS-II will see if PDFfit2 can be imported into the current Python.

GSASII.config_example.**previous_GPX_files = []**

A list of previously used .gpx files. This is used internally by GSAS-II and would not normally be changed by a user.

GSASII.config_example.**rmcprofile_exec = None**

Defines the full path to a compiled executable for RMCProfile (downloaded from https://rmcprofile.ornl.gov/download) If None (the default), GSAS-II will see if RMCProfile is found in the path, as well as, where python, the GSAS-II Python files & binaries are located and the current working directory. For MacOS, RMCProfile will run only from /Applications/ so that is the only location checked if rmcprofile_exec is not specified.

GSASII.config_example.**show_gpxSize = False**

When True, the sizes of the sections of the GPX file are listed when the GPX file is opened. Default is False.

6.3 *GSASIIElem: functions for element types*

6.3.1 GSASIIElem Routines

Routines used to define element settings follow.

GSASII.GSASIIElem.**AddWave2atm**(*atom*)

For modulated structures, make sure that the wave info dict is the last atom entry

GSASII.GSASIIElem.**BlenResCW**(*Els*, *BLtables*, *wave*)

Computes resonant scattering lengths - single wavelength version (CW) returns bo+b' and b'"

GSASII.GSASIIElem.**BlenResTOF**(*Els*, *BLtables*, *wave*)

Computes resonant scattering lengths - multiple wavelength version (TOF) returns bo+b' and b'"

GSASII.GSASIIElem.**CheckElement**(*El*)

Check if element El is in the periodic table

Parameters

El (*str*) – One or two letter element symbol, capitaliztion ignored

Returns

True if the element is found

GSASII.GSASIIElem.**ClosedFormFF**(*Z*, *SQ*, *k*, *N*)

Closed form expressions for FT Slater fxns. IT B Table 1.2.7.4 (not used at present - doesn't make sense yet)

Parameters

- **Z** – element zeta factor
- **SQ** – (sin-theta/lambda)**2
- **k** – int principal Bessel fxn order as in <jk>
- **N** – int power

return: form factor

GSASII.GSASIIElem.**ComptonFac**(*El*, *SQ*)

compute Compton scattering factor

Parameters

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

Returns

compton scattering factor

GSASII.GSASIIElem.**FPcalc**(*Orbs*, *KEv*)

Compute real & imaginary resonant X-ray scattering factors

Parameters

- **Orbs** – list of orbital dictionaries as defined in GetXsectionCoeff
- **KEv** – x-ray energy in keV

Returns

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

GSASII.GSASIIElem.**FixValence**(*El*)

Returns the element symbol, even when a valence is present

GSASII.GSASIIElem.**GetAtomInfo**(*El*, *ifMag=False*)

reads element information from atmdata.py

GSASII.GSASIIElem.**GetBLtable**(*General*)

returns a dictionary of neutron scattering length data for atom types & isotopes found in General

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

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

GSASII.GSASIIElem.**GetEFFtable**(*atomTypes*)

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

Parameters
atomTypes (*list*) – list of atom types

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

GSASII.GSASIIElem.**GetEFormFactorCoeff**(*El*)

Read electron form factor coefficients from *atomdata.py* file

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

Returns
FormFactors: list of form factor dictionaries

Each electrn form factor dictionary is:

- *Symbol*: 4 character element symbol (no valence)
- *Z*: atomic number
- *fa*: 5 A coefficients
- *fb*: 5 B coefficients

GSASII.GSASIIElem.**GetFFC5**(*ElSym*)

Get 5 term form factor and Compton scattering data

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

Return El
dictionary with 5 term form factor & compton coefficients

GSASII.GSASIIElem.**GetFFtable**(*atomTypes*)

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

Parameters
atomTypes (*list*) – list of atom types

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

GSASII.GSASIIElem.**GetFormFactorCoeff**(*El*)

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

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

Returns
FormFactors: list of form factor dictionaries

Each X-ray form factor dictionary is:

- *Symbol*: 4 character element symbol with valence (e.g. 'NI+2')
- *Z*: atomic number
- *fa*: 4 A coefficients
- *fb*: 4 B coefficients
- *fc*: C coefficient

GSASII.GSASIIElem.**GetMFtable**(*atomTypes*, *Landeg*)

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

Parameters

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

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

GSASII.GSASIIElem.**GetMagFormFacCoeff**(*El*)

Read magnetic form factor data from atmdata.py

Parameters
El – 2 character element symbol

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

each dictionary contains:

- 'Symbol':Symbol
- 'Z':Z
- 'mfa': 4 MA coefficients
- 'nfa': 4 NA coefficients
- 'mfb': 4 MB coefficients
- 'nfb': 4 NB coefficients
- 'mfc': MC coefficient
- 'nfc': NC coefficient

GSASII.GSASIIElem.**GetORBtable**(*atomTypes*)

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

Parameters
atomTypes (*list*) – list of atom types

Returns
ORBtable, dictionary of orbital form factor data; key is atom type

GSASII.GSASIIElem.**GetXsectionCoeff**(*El*)

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

Parameters
El – 2 character element symbol

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

each dictionary is:

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

GSASII.GSASIIElem.**MagScatFac**(*El*, *SQ*)

compute value of form factor

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

Returns
real part of form factor

GSASII.GSASIIElem.**ScatFac**(*El*, *SQ*)

compute value of form factor

Parameters
- **El** – element dictionary defined in GetFormFactorCoeff
- **SQ** – (sin-theta/lambda)**2

Returns
real part of form factor

GSASII.GSASIIElem.**ScatFacDer**(*El*, *SQ*)

compute derivative of form factor wrt SQ

Parameters
- **El** – element dictionary defined in GetFormFactorCoeff

- **SQ** – (sin-theta/lambda)**2

Returns
real part of form factor

GSASII.GSASIIElem.**SetAtomColor**(*El*, *RGB*)

Overrides the default color in the atoms table; not saved

GSASII.GSASIIElem.**SetupGeneral**(*data*, *dirname*)

Initialize the General sections of the Phase tree contents. Should be done after changes to the Atoms array.

Called by routine SetupGeneral (in `GSASIIphsGUI.UpdatePhaseData()`), `GSASIIphsGUI.makeIsoNewPhase()`, `GSASIImiscGUI.saveNewPhase()`, and in `GSASIIscriptable.SetupGeneral()`.

GSASII.GSASIIElem.**StripValence**(*El*)

Returns element symbol without valence

GSASII.GSASIIElem.**getBLvalues**(*BLtables*, *ifList=False*)

Needs a doc string

GSASII.GSASIIElem.**getFFvalues**(*FFtables*, *SQ*, *ifList=False*)

Needs a doc string

GSASII.GSASIIElem.**getMFvalues**(*MFtables*, *SQ*, *ifList=False*)

Needs a doc string

6.4 *GSASIIlattice: Unit Cell Computations*

Performs lattice-related computations

Note that as used here *G* is the reciprocal lattice tensor, and *g* is its inverse, $G = g^{-1}$, where

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

The "*A* tensor" terms are defined as $A = (G_{11} \quad G_{22} \quad G_{33} \quad 2G_{12} \quad 2G_{13} \quad 2G_{23})$ and *A* can be used in this fashion: $d^* = \sqrt{A_0h^2 + A_1k^2 + A_2l^2 + A_3hk + A_4hl + A_5kl}$, where *d* is the d-spacing, and d^* is the reciprocal lattice spacing, $Q = 2\pi d^* = 2\pi/d$. Note that GSAS-II variables `p::Ai` (i = 0, 1,... 5) and `p` is a phase number are used for the *Ai* values. See `A2cell()`, `cell2A()` for interconversion between A and unit cell parameters; `cell2Gmat()` `Gmat2cell()` for G and cell parameters.

When the hydrostatic/elastic strain coefficients (*Dij*, D_{ij}) are used, they are added to the *A* tensor terms (Ai, A_i) so that A is redefined $A = (A_0 + D_{11} \quad A_1 + D_{22} \quad A_2 + D_{33} \quad A_3 + D_{12} \quad A_4 + D_{13} \quad A_5 + D_{23})$. See `cellDijFill()`. Note that GSAS-II variables `p:h:Dij` (i,j = 1, 2, 3) and `p` is a phase number and `h` a histogram number are used for the *Dij* values.

6.4.1 GSASIIlattice Classes & Routines

`GSASIIlattice` Classes & routines follow

GSASII.GSASIIlattice.**A2Gmat**(*A*, *inverse=True*)

Fill real & reciprocal metric tensor (G) from A.

Parameters

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

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

GSASII.GSASIIlattice.**A2cell**(*A*)

Compute unit cell constants from A

Parameters
A – [G11,G22,G33,2*G12,2*G13,2*G23] G - reciprocal metric tensor

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

GSASII.GSASIIlattice.**A2invcell**(*A*)

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

GSASII.GSASIIlattice.**AplusDij**(*A*, *Dij*, *SGData*)

returns the A corrected by Dij

Parameters

- **A** (*list*) – reciprocal metric terms A0-A5
- **Dij** (*array*) – unique Dij values as stored in Hstrain
- **SGdata** (*dict*) – a symmetry object

Returns list newA
A corrected by Dij

GSASII.GSASIIlattice.**CellAbsorption**(*ElList*, *Volume*)

Compute unit cell absorption

Parameters

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

Returns
mu-total/Volume

GSASII.GSASIIlattice.**CellBlock**(*nCells*)

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

GSASII.GSASIIlattice.**CellDijCorr**(*Cell*, *SGData*, *Data*, *hist*)

Returns the cell corrected for Dij values.

Parameters

- **Cell** (*list*) – lattice parameters
- **SGdata** (*dict*) – a symmetry object
- **Data** (*dict*) – phase data structure; contains set of Dij values
- **hist** (*str*) – histogram name

Returns
cell corrected for Dij values

GSASII.GSASIIlattice.**CentCheck**(*Cent*, *H*)

checks individual hkl for centering extinction; returns True for allowed, False otherwise - slow

GSASII.GSASIIlattice.**CosAngle**(*U*, *V*, *G*)

calculate cos of angle between U & V in generalized coordinates defined by metric tensor G

Parameters

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

Returns
cos(phi)

GSASII.GSASIIlattice.**CosSinAngle**(*U*, *V*, *G*)

calculate sin & cos of angle between U & V in generalized coordinates defined by metric tensor G

Parameters

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

Returns
cos(phi) & sin(phi)

GSASII.GSASIIlattice.**CrsAng**(*H*, *cell*, *SGData*)

Convert HKL to polar coordinates with proper orientation WRT space group point group

Parameters

- **H** (*array*) – hkls
- **cell** (*list*) – lattice parameters
- **SGData** (*dict*) – space group data

Returns arrays phi,beta
polar, azimuthal angles for HKL

GSASII.GSASIIlattice.**CubicSHarm**(*L*, *M*, *Th*, *Ph*)

Calculation of the cubic harmonics given in Table 3 in M.Kara & K. Kurki-Suonio, Acta Cryt. A37, 201 (1981). For L = 14,20 only for m3m from F.M. Mueller and M.G. Priestley, Phys Rev 148, 638 (1966)

Parameters

- **L** (*int*) – degree of the harmonic (L >= 0)
- **M** (*int*) – order number [|M| <= L]
- **Th** (*float/array*) – Azimuthal coordinate 0 <= Th <= 360
- **Ph** (*float/array*) – Polar coordinate 0<= Ph <= 180

Returns klm value/array
cubic harmonics

GSASII.GSASIIlattice.**Dsp2pos**(*Inst*, *dsp*)

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

GSASII.GSASIIlattice.**ExpandCell**(*Atoms*, *atCodes*, *cx*, *Trans*)

Expand a list of atoms to an new unit cell.

Parameters

Atoms (*list*) – atoms array as found in a Phase object

:param list atCodes:list of codes that describe symmetry applied to

generate atom from original position

Parameters

- **cx** (*int*) – pointer to location of x coord in Atoms
- **Trans** (*np.array*) – 3x3 transformation matrix

Returns

newAtoms,Codes where newAtoms is the expanded list of atoms and Codes is an expanded and updated version of atCodes

GSASII.GSASIIlattice.**FindNonstandard**(*controls*, *Phase*)

Find nonstandard setting of magnetic cell that aligns with parent nuclear cell

Parameters

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

Returns

None

GSASII.GSASIIlattice.**Flnh**(*SHCoef*, *phi*, *beta*, *SGData*)

needs doc string

GSASII.GSASIIlattice.**GenAtomConstraints**(*oldPhase*, *newPhase*, *atCodes*, *Trans*)

Generate the constraints on generated atoms between oldPhase and newPhase where atCodes links new atoms to the original atoms. uUsed for a magnetic phase transformed generated from oldPhase.

Parameters

- **oldPhase** – the phase object for the original phase
- **newPhase** – the phase object for the newly created phase
- **atCodes** (*list*) – a list strings where each element has the original atom id and the operator for each new atom (I think: at#:opr#+x,y,z)
- **Trans** (*np.array*) – the transformation matrix

Returns

(constraints, message) where * constraints is a list of constraints that should be added to the

Phase section of the constraints dict.

- message is a message to display or None

GSASII.GSASIIlattice.**GenCellConstraints**(*Trans*, *oRanId*, *nRanId*, *origA*, *oSGLaue*, *nSGLaue*, *debug=False*)

Generate the constraints between two unit cells constants for a magnetic phase transformed by matrix Trans.

Parameters

- **Trans** (*np.array*) – a 3x3 direct cell transformation matrix where, Trans = np.array([[2/3, 4/3, 1/3], [-1, 0, 0], [-1/3, -2/3, 1/3]]) (for a' = 2/3a + 4/3b + 1/3c; b' = -a; c' = -1/3, -2/3, 1/3)
- **oRanId** (*int*) – Random id for the original phase
- **nRanId** (*int*) – Random id for the transformed phase to be constrained from original phase
- **origA** (*list*) – reciprocal cell ("A*") tensor (used for debug only)
- **oSGLaue** (*dict*) – space group info for original phase
- **nSGLaue** (*dict*) – space group info for transformed phase
- **debug** (*bool*) – If true, the constraint input is used to compute and print A* and from that the direct cell for the transformed phase.

Returns
a list of generated constraints

GSASII.GSASIIlattice.**GenHAPConstraints**(*Vratio*, *oRanId*, *nRanId*, *hRanId*)

Generate the HAP constraints between two phases: the original chemical (nuclear) phase and the new, magnetic phase.

Parameters

- **Vratio** (*float*) – the ratio of the volume of the magnetic phase divided by the original phase. This should be 1 or >1 [often 2 or sqrt(2) etc.]
- **oRanId** (*int*) – Random id for the original phase
- **nRanId** (*int*) – Random id for the transformed (magnetic) phase to be constrained to match the original phase
- **hRanId** (*str*) – histogram Random Id

Returns
a list of generated constraints

GSASII.GSASIIlattice.**GenHBravais**(*dmin*, *Bravais*, *A*, *cctbx_args=None*, *ifList=False*)

Generate the positionally unique powder diffraction reflections

Parameters

- **dmin** – minimum d-spacing in A
- **Bravais** – lattice type (see GetBraviasNum). Bravais is one of:
  - 0 F cubic
  - 1 I cubic
  - 2 P cubic
  - 3 R hexagonal (trigonal not rhombohedral)
  - 4 P hexagonal
  - 5 I tetragonal
  - 6 P tetragonal
  - 7 F orthorhombic
  - 8 I orthorhombic
  - 9 A orthorhombic
  - 10 B orthorhombic

  - 11 C orthorhombic
  - 12 P orthorhombic
  - 13 I monoclinic
  - 14 A monoclinic
  - 15 C monoclinic
  - 16 P monoclinic
  - 17 P triclinic
- **A** – reciprocal metric tensor elements as [G11,G22,G33,2*G12,2*G13,2*G23]
- **cctbx_args** (*dict*) – items defined in CCTBX:
  - 'sg_type': value from cctbx.sgtbx.space_group_type(symmorphic_sgs[ibrav])
  - 'uctbx_unit_cell': pointer to `cctbx.uctbx.unit_cell()`
  - 'miller_index_generator': pointer to `cctbx.miller.index_generator()`
- **ifList** – if True output is 2D list of HKL; if False (default) output is 2D nd.array of HKLs * ifList=False is fast & suitable for indexing routines; ifList=True is needed for graphics. * Currently applies only to Triclinic, Monoclinic & Orthorhombic; otherwise output is 2D list.

Returns

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

GSASII.GSASIIlattice.**GenHLaue**(*dmin*, *SGData*, *A*)

Generate the crystallographically unique powder diffraction reflections for a lattice and Bravais type

Parameters

- **dmin** – minimum d-spacing
- **SGData** – space group dictionary with at least
  - 'SGLaue': Laue group symbol: one of '-1','2/m','mmm','4/m','6/m','4/mmm','6/mmm', '3m1', '31m', '3', '3R', '3mR', 'm3', 'm3m'
  - 'SGLatt': lattice centering: one of 'P','A','B','C','I','F'
  - 'SGUniq': code for unique monoclinic axis one of 'a','b','c' (only if 'SGLaue' is '2/m') otherwise an empty string
- **A** – reciprocal metric tensor elements as [G11,G22,G33,2*G12,2*G13,2*G23]

Returns

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

GSASII.GSASIIlattice.**GenPfHKLs**(*nMax*, *SGData*, *A*)

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

Parameters

- **nMax** – maximum number of hkls returned
- **SGData** – space group dictionary with at least
  - 'SGLaue': Laue group symbol: one of '-1','2/m','mmm','4/m','6/m','4/mmm','6/mmm', '3m1', '31m', '3', '3R', '3mR', 'm3', 'm3m'

- 'SGLatt': lattice centering: one of 'P','A','B','C','I','F'
- 'SGUniq': code for unique monoclinic axis one of 'a','b','c' (only if 'SGLaue' is '2/m') otherwise an empty string

- **A** – reciprocal metric tensor elements as [G11,G22,G33,2*G12,2*G13,2*G23]

Returns
HKL = list of 'h k l' strings sorted with largest d first; no duplicate zones

GSASII.GSASIIlattice.**GenRBCoeff**(*sytsym*, *RBsym*, *L*)

imposes rigid body symmetry on spherical harmonics terms

Parameters

- **sytsym** (*str*) – atom position site symmetry symbol
- **RBsym** (*str*) – molecular point symmetry symbol
- **L** (*int*) – spherical harmonic order no.

Returns
list newNames: spherical harmonic term of order L as either C(L,M) or C(L,M)c for cubic terms, list newSgns: matching coefficient signs as +/- 1.0

GSASII.GSASIIlattice.**GenSHCoeffT**(*SGLaue*, *SamSym*, *L*, *IfLMN=True*)

Generate spherical harmonics coefficient names for texture

Note there are functions GenShCoeff and GenSHCoeffT

Parameters

- **SGLaue** (*str*) – Laue symbol
- **SamSym** (*str*) – sample symmetry symbol
- **L** (*int*) – spherical harmonic order no.
- **IfLMN** (*bool*) – if TRUE returns sp.harm. name as C(L,M,N), otherwise returns C(L,N)

Returns
coefficient name as C(L,M,N) or C(L,N)

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

needs a doc string

GSASII.GSASIIlattice.**GenShCoeff**(*sytsym*, *L*)

Generate spherical harmonic coefficient names for atom site symmetry

Note there are functions GenShCoeff and GenSHCoeffT

Parameters

- **sytsym** (*str*) – site symmetry or perhaps molecular symmetry
- **L** (*int*) – spherical harmonic order no.

Returns
newNames, newSgns list newNames: spherical harmonic term of order L as either C(L,M) or C(L,M)c for cubic terms list newSgns: matching coefficient signs as +/- 1.0

GSASII.GSASIIlattice.**GenerateCellConstraints**()

Generate unit cell constraints for transforming one set of A tensor values to another using symbolic math (requires the sympy package)

Note that this is only used to do the symbolic math needed to generate cell relationships. It is not used normally in GSAS-II.

GSASII.GSASIIlattice.**GetBraviasNum**(*center*, *system*)

Determine the Bravais lattice number, as used in GenHBravais

Parameters

- **center** – one of: 'P', 'C', 'I', 'F', 'R' (see SGLatt from GSASIIspc.SpcGroup)
- **system** – one of 'cubic', 'hexagonal', 'tetragonal', 'orthorhombic', 'trigonal' (for R) 'monoclinic', 'triclinic' (see SGSys from GSASIIspc.SpcGroup)

Returns

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

GSASII.GSASIIlattice.**GetKcl**(*L*, *N*, *SGLaue*, *phi*, *beta*)

needs doc string

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

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

GSASII.GSASIIlattice.**GetKsl**(*L*, *M*, *SamSym*, *psi*, *gam*)

needs doc string

GSASII.GSASIIlattice.**Glnh**(*SHCoef*, *psi*, *gam*, *SamSym*)

needs doc string

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

Extract A from reciprocal metric tensor (G)

Parameters

G – reciprocal metric tensor (3x3 numpy array)

Returns

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

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

Computes orthogonalization matrix from reciprocal metric tensor G

Returns

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

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

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

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

Parameters

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

Returns

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

GSASII.GSASIIlattice.**H2ThPh2**(*H*, *Bmat*)

Convert HKL to spherical polar & azimuth angles

Parameters

- **H** (*array*) – array of hkl as [n,3]
- **Bmat** (*[3,3] array*) – inv crystal to Cartesian transformation

Returns array Th
HKL azimuth angles

Returns array Ph
HKL polar angles

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

Computes spherical coords for hkls; view along 001

Parameters

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

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

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

needs doc string

GSASII.GSASIIlattice.**KslCalc**(*trm*, *psi*, *gam*, *ifDerv=False*)

Compute one angular part term in spherical harmonics

:param str trm:sp. harm term name in the form of 'C(l,m)' or 'C(l,m)c' for cubic :param float/array psi: Azimuthal coordinate 0 <= psi <= 360 :param float/array gam: Polar coordinate 0<= gam <= 180

Returns array Ksl
spherical harmonics angular part for psi,gam pairs

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

Impose Laue symmetry on hkl

Parameters

- **Laue** (*str*) – Laue symbol, as below

  centrosymmetric Laue groups:

```
['-1','2/m','112/m','2/m11','mmm','-42m','-4m2','4/mmm','-3','-3m',
'-31m','-3m1','6/m','6/mmm','m3','m3m']
```

  noncentrosymmetric Laue groups:

```
['1','2','211','112','m','m11','11m','222','mm2','m2m','2mm',
'4','-4','422','4mm','3','312','321','3m','31m','3m1','6','-6',
'622','6mm','-62m','-6m2','23','432','-43m']
```

- **HKLF** – np.array([[h,k,l,…]]) reflection set to be converted

Returns
HKLF new reflection array with imposed Laue symmetry

GSASII.GSASIIlattice.**LaueUnique2**(*SGData*, *refList*)

Impose Laue symmetry on hkl

Parameters

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

Returns

HKLF new reflection array with imposed Laue symmetry

GSASII.GSASIIlattice.**MatchGenAtomConstraints**(*oldPhase*, *newPhase*, *Trans*, *U*, *V*)

Generate the constraints on generated atoms in newPhase similar to *GenAtomConstraints()* but without pointers to relate the atoms. The atom positions are regenerated from the original phase and then are matched to the magnetic positions in newPhase. The positions generated here are what are then used for the new atom positions.

Parameters

- **oldPhase** – the phase object for the original phase
- **newPhase** – the phase object for the newly created phase
- **Trans** (*np.array*) – the transformation matrix
- **U** (*np.array*) – the offset vector, subtracted before transformation matrix
- **V** (*np.array*) – the offset vector, added after transformation matrix

Returns

(constraints, message) where * constraints is a list of constraints that should be added to the

Phase section of the constraints dict.

- message is a message to display or None

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

needs doc string

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

finds symmetry rules for spherical harmonic coefficients for Laue groups

Parameters

- **SGLaue** (*str*) – Laue symbol
- **L** (*int*) – principal harmonic term; only evens are used
- **M** (*int*) – second harmonic term; can be -L <= M <= L

Returns

True if allowed

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

find unit cell intercepts for a stack of hkl planes

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

convert powder pattern position (2-theta or TOF, musec) to d-spacing is currently only approximate for EDX data; accurate for others.

GSASII.GSASIIlattice.**RBChk**(*sytsym*, *L*, *M*)

finds symmetry rules for spherical harmonic coefficients for site symmetries :param str sytsym: atom site symmetry symbol :param int L: principal harmonic term L>0 none beyond L=5; not considered :param int M: second harmonic term; can be -L <= M <= L :returns True if allowed and sign for term NB: not complete for all possible site symmetries! Many are missing Based on Tables 2 & 4 of M. Kara & K. Kurki-Suonio, Acta Cryst. A37, 201-210 (1981).

GSASII.GSASIIlattice.**RBsymCheck**(*Atoms*, *ct*, *cx*, *cs*, *AtLookUp*, *Amat*, *RBObjIds*, *SGData*)

Checks members of a rigid body to see if one is a symmetry equivalent of another. If so the atom site frac is set to zero.

Parameters

- **Atoms** – atom array as defined in GSAS-II; modified here
- **ct** – int location of atom type in Atoms item
- **cx** – int location of x,y,z,frac in Atoms item
- **AtLookUp** (*dict*) – atom lookup by Id table
- **Amat** (*np.array*) – crystal-to-Cartesian transformation matrix
- **RBObjIds** (*list*) – atom Id belonging to rigid body being tested
- **SGData** (*dict*) – GSAS-II space group info.

Returns

Atoms with modified atom frac entries

GSASII.GSASIIlattice.**RBsymChk**(*RBsym*, *cubic*, *coefNames*, *L=18*)

imposes rigid body symmetry on spherical harmonics terms Key problem is noncubic RB symmetries in cubic site symmetries & vice versa. :param str RBsym: molecular point symmetry symbol :param bool cubic: True if atom site symmetry is cubic :param list coefNames: sp. harm coefficient names to be checked/converted :param int L: maximum spherical harmonic order no. for cubic generation if needed

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

needs doc string

GSASII.GSASIIlattice.**SHarmcal**(*SytSym*, *SHFln*, *psi*, *gam*)

Perform a surface spherical harmonics computation & return sum of squares. Only used for plotting. Note that the the number of gam values must either be 1 or must match psi

Parameters

- **SytSym** (*str*) – site symmetry - only looking for cubics
- **SHFln** (*dict*) – spherical harmonics coefficients; key has L & M
- **psi** (*float/array*) – Azimuthal coordinate 0 <= psi <= 360
- **gam** (*float/array*) – Polar coordinate 0<= gam <= 180

Returns array SHVal

spherical harmonics array for psi,gam values

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

Compute sample orientation angles vs laboratory coord. system

Parameters

- **Tth** – Signed theta
- **Gangls** – Sample goniometer angles phi,chi,omega,azmuth

- **Sangl** – Sample angle zeros om-0, chi-0, phi-0
- **IFCoup** – True if omega & 2-theta coupled in CW scan

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

GSASII.GSASIIlattice.**SetUVvec**(*Neigh*)

Set deformation coordinate choices from neighbors; called in G2phsGUI/UpdateDeformation

Parameters
Neigh (*list*) – list of neighboring atoms; each with name, dist & cartesian vector

Returns list UVvec
list of normalized vectors

Returns list UVchoice
list of names for each

GSASII.GSASIIlattice.**SphHarmAng**(*L*, *M*, *P*, *Th*, *Ph*)

Compute spherical harmonics values using scipy.special.sph_harm

Parameters

- **L** (*int*) – degree of the harmonic (L >= 0)
- **M** (*int*) – order number (|M| <= L)
- **P** (*int*) – sign flag = -1 or 1
- **Th** (*float/array*) – Azimuthal coordinate 0 <= Th <= 360
- **Ph** (*float/array*) – Polar coordinate 0<= Ph <= 180

Returns ylmp value/array
as reals

GSASII.GSASIIlattice.**SwapIndx**(*Axis*, *H*)

needs doc string

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

exchange 2 items in a list

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

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

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

Transform lattice parameters by matrix

Parameters

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

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

GSASII.GSASIIlattice.**TransformPhase**(*oldPhase*, *newPhase*, *Trans*, *Uvec*, *Vvec*, *ifMag*, *Force=True*)

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

Parameters

- **oldPhase** – dict G2 phase info for old phase

- **newPhase** – dict G2 phase info for new phase; with new cell & space group atoms are from oldPhase & will be transformed
- **Trans** (*np.array*) – lattice transformation matrix M
- **Uvec** – array parent coordinates transformation vector U
- **Vvec** – array child coordinate transformation vector V
- **ifMag** – bool True if convert to magnetic phase; if True all nonmagnetic atoms will be removed

Returns
newPhase dict modified G2 phase info

Returns
atCodes list atom transformation codes

GSASII.GSASIIlattice.**U6toUij**(*U6*)

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

Parameters
U6 (*list*) – 6 terms of u11,u22,…

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

GSASII.GSASIIlattice.**Uij2Ueqv**(*Uij*, *GS*, *Amat*)

returns 1/3 trace of diagonalized U matrix :param Uij: numpy array [Uij] :param GS: Uij too betaij conversion matrix :param Amat: crystal to Cartesian transformation matrix :returns: 1/3 trace of diagonalized U matrix :returns: True if nonpositive-definite; False otherwise

GSASII.GSASIIlattice.**Uij2betaij**(*Uij*, *G*)

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

Parameters

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

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

GSASII.GSASIIlattice.**UijtoU6**(*U*)

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

GSASII.GSASIIlattice.**UniqueCellByLaue = [[['m3', 'm3m'], (0,)], [['3R', '3mR'], (0, 3)], [['3', '3m1', '31m', '6/m', '6/mmm', '4/m', '4/mmm'], (0, 2)], [['mmm'], (0, 1, 2)], [['2/ma'], (0, 1, 2, 3)], [['2/mb'], (0, 1, 2, 4)], [['2/mc'], (0, 1, 2, 5)], [['-1'], (0, 1, 2, 3, 4, 5)]]**

List the unique cell terms by index for each Laue class

GSASII.GSASIIlattice.**betaij2Uij**(*betaij*, *G*)

Convert beta-ij to Uij tensors

:param beta-ij - numpy array [beta-ij] :param G: reciprocal metric tensor :returns: Uij: numpy array [Uij]

GSASII.GSASIIlattice.**calc_V**(*A*)

Compute the real lattice volume (V) from A

GSASII.GSASIIlattice.**calc_rDsq**(*H*, *A*)

calc 1/d^2 from individual hkl and A-terms

GSASII.GSASIIlattice.**calc_rDsq2**(*H*, *G*)

computes 1/d^2 from one hkl & reciprocal metric tensor G

GSASII.GSASIIlattice.**calc_rDsqA**(*H*, *A*)

calc array of 1/d^2 from array of hkl & A-terms

GSASII.GSASIIlattice.**calc_rDsqSS**(*H*, *A*, *vec*)

computes 1/d^2 from one hklm, reciprocal metric tensor A & k-vector

GSASII.GSASIIlattice.**calc_rDsqT**(*H*, *A*, *Z*, *tof*, *difC*)

computes 1/d^2 from hkl array & reciprocal metric tensor A with TOF ZERO shift

GSASII.GSASIIlattice.**calc_rDsqTSS**(*H*, *A*, *vec*, *Z*, *tof*, *difC*)

computes 1/d^2 from hklm array, reciprocal metric tensor A & k-vector with TOF Z shift

GSASII.GSASIIlattice.**calc_rDsqZ**(*H*, *A*, *Z*, *tth*, *lam*)

computes 1/d^2 from hkl array & reciprocal metric tensor A with CW ZERO shift

GSASII.GSASIIlattice.**calc_rDsqZSS**(*H*, *A*, *vec*, *Z*, *tth*, *lam*)

computes 1/d^2 from hklm array, reciprocal metric tensor A & k-vector with CW Z shift

GSASII.GSASIIlattice.**calc_rV**(*A*)

Compute the reciprocal lattice volume (V*) from A

GSASII.GSASIIlattice.**calc_rVsq**(*A*)

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

GSASII.GSASIIlattice.**cell2A**(*cell*)

Obtain A = [G11,G22,G33,2*G12,2*G13,2*G23] from lattice parameters

Parameters

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

Returns

G reciprocal metric tensor as 3x3 numpy array

GSASII.GSASIIlattice.**cell2AB**(*cell*, *alt=False*)

Computes orthogonalization matrix from unit cell constants

Parameters

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

Returns

tuple of two 3x3 numpy arrays (A,B) A for crystal to Cartesian transformations A*x = np.inner(A,x) = X B (= inverse of A) for Cartesian to crystal transformation B*X = np.inner(B,X) = x both rounded to 12 places (typically zero terms = +/-10e-6 otherwise)

GSASII.GSASIIlattice.**cell2GS**(*cell*)

returns Uij to betaij conversion matrix

GSASII.GSASIIlattice.**cell2Gmat**(*cell*)

Compute real and reciprocal lattice metric tensor from unit cell constants

Parameters

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

Returns

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

GSASII.GSASIIlattice.**cellAlbl = ('a', 'b', 'c', 'alpha', 'beta', 'gamma')**

ASCII labels for a, b, c, alpha, beta, gamma

GSASII.GSASIIlattice.**cellDijFill**(*pfx*, *phfx*, *SGData*, *parmDict*)

Returns the filled-out reciprocal cell (A) terms from the parameter dictionaries corrected for Dij.

Parameters

- **pfx** (*str*) – parameter prefix ("n::", where n is a phase index)
- **phfx** (*str*) – parameter prefix ("n:h:", where n is a phase index and h is a histogram index)
- **SGdata** (*dict*) – a symmetry object
- **parmDict** (*dict*) – a dictionary of parameters

Returns

A, a list of six terms

GSASII.GSASIIlattice.**cellUlbl = ('a', 'b', 'c', 'α', 'β', 'γ')**

unicode labels for a, b, c, alpha, beta, gamma

GSASII.GSASIIlattice.**cellUnique**(*SGData*)

Returns the indices for the unique A tensor terms based on the Laue class. Any terms that are determined from others or are zero are not included.

Parameters

SGdata (*dict*) – a symmetry object

Returns

a list of 0 to 6 terms with indices of the unique A terms

GSASII.GSASIIlattice.**cellXformRelations = {0: ['1.0*A0*T[0,0]**2', '1.0*A1*T[0,1]**2', '1.0*A2*T[0,2]**2', '1.0*A3*T[0,0]*T[0,1]', '1.0*A4*T[0,0]*T[0,2]', '1.0*A5*T[0,1]*T[0,2]'], 1: ['1.0*A0*T[1,0]**2', '1.0*A1*T[1,1]**2', '1.0*A2*T[1,2]**2', '1.0*A3*T[1,0]*T[1,1]', '1.0*A4*T[1,0]*T[1,2]', '1.0*A5*T[1,1]*T[1,2]'], 2: ['1.0*A0*T[2,0]**2', '1.0*A1*T[2,1]**2', '1.0*A2*T[2,2]**2', '1.0*A3*T[2,0]*T[2,1]', '1.0*A4*T[2,0]*T[2,2]', '1.0*A5*T[2,1]*T[2,2]'], 3: ['2.0*A0*T[0,0]*T[1,0]', '2.0*A1*T[0,1]*T[1,1]', '2.0*A2*T[0,2]*T[1,2]', '1.0*A3*(T[0,0]*T[1,1] + T[1,0]*T[0,1])', '1.0*A4*(T[0,0]*T[1,2] + T[1,0]*T[0,2])', '1.0*A5*(T[0,1]*T[1,2] + T[1,1]*T[0,2])'], 4: ['2.0*A0*T[0,0]*T[2,0]', '2.0*A1*T[0,1]*T[2,1]', '2.0*A2*T[0,2]*T[2,2]', '1.0*A3*(T[0,0]*T[2,1] + T[2,0]*T[0,1])', '1.0*A4*(T[0,0]*T[2,2] + T[2,0]*T[0,2])', '1.0*A5*(T[0,1]*T[2,2] + T[2,1]*T[0,2])'], 5: ['2.0*A0*T[1,0]*T[2,0]', '2.0*A1*T[1,1]*T[2,1]', '2.0*A2*T[1,2]*T[2,2]', '1.0*A3*(T[1,0]*T[2,1] + T[2,0]*T[1,1])', '1.0*A4*(T[1,0]*T[2,2] + T[2,0]*T[1,2])', '1.0*A5*(T[1,1]*T[2,2] + T[2,1]*T[1,2])']}**

cellXformRelations provide the constraints on newA[i] values for a new cell generated from oldA[i] values.

GSASII.GSASIIlattice.**cellZeros**(*SGData*)

Returns a list with the A terms required to be zero based on Laue symmetry

Parameters

SGdata (*dict*) – a symmetry object

Returns

A list of six terms where the values are True if the A term must be zero, False otherwise.

GSASII.GSASIIlattice.**combinations**(*items*, *n*)

take n distinct items, order matters

GSASII.GSASIIlattice.**criticalEllipse**(*prob*)

Calculate critical values for probability ellipsoids from probability

GSASII.GSASIIlattice.**fillgmat**(*cell*)

Compute lattice metric tensor from unit cell constants

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

Returns
3x3 numpy array

GSASII.GSASIIlattice.**fmtCellConstraints**(*cellConstr*)

Format the cell relationships created in *GenerateCellConstraints()* in a format that can be used to generate constraints.

Use:

```
cXforms = G2lat.fmtCellConstraints(G2lat.GenerateCellConstraints())
```

Note that this is only used to do the symbolic math needed to generate cell relationships. It is not used normally in GSAS-II.

GSASII.GSASIIlattice.**getCellEsd**(*pfx*, *SGData*, *A*, *covData*, *unique=False*)

Compute the standard uncertainty on cell parameters

Parameters

- **pfx** (*str*) – prefix of form "p::"
- **SGdata** – space group information
- **A** (*list*) – Reciprocal cell Ai terms
- **covData** (*dict*) – covariance tree item
- **unique** (*bool*) – when True, only directly refined parameters (a in cubic, a & alpha in rhombohedral cells) are assigned positive s.u. values. Used for CIF generation.

GSASII.GSASIIlattice.**getCellSU**(*pId*, *hId*, *SGData*, *parmDict*, *covData*)

Compute the unit cell parameters and standard uncertainties where lattice parameters and Hstrain (Dij) may be refined.

Parameters

- **pId** – phase index
- **hId** – histogram index
- **SGdata** – space group information for current phase
- **parmDict** (*dict*) – parameter dict, must have all non-zero Dij and Ai terms
- **covData** (*dict*) – covariance tree item

Returns
cellList,cellSig where each term is a list of 7 items, a, b,… Vol and sig(a), sig(b),… sig(Vol), respectively

GSASII.GSASIIlattice.**getHKLmax**(*dmin*, *SGData*, *A*)

finds maximum allowed hkl for given A within dmin

GSASII.GSASIIlattice.**getPeakPos**(*dataType*, *parmdict*, *dsp*)

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

GSASII.GSASIIlattice.**invcell2Gmat**(*invcell*)

Compute real and reciprocal lattice metric tensor from reciprocal
unit cell constants

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

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

GSASII.GSASIIlattice.**invpolfcal**(*ODFln*, *SGData*, *phi*, *beta*)

needs doc string

GSASII.GSASIIlattice.**newCentCheck**(*Cent*, *H*)

checks np.array of HKls for centering extinction; returns allowed HKLs - fast

GSASII.GSASIIlattice.**permutations**(*items*)

take all items, order matters

GSASII.GSASIIlattice.**polfcal**(*ODFln*, *SamSym*, *psi*, *gam*)

Perform a pole figure computation. Note that the the number of gam values must either be 1 or must match psi. Updated for numpy 1.8.0

GSASII.GSASIIlattice.**prodMGMT**(*G*, *Mat*)

Transform metric tensor by matrix

Parameters

- **G** – array metric tensor
- **Mat** – array transformation matrix

Returns
array new metric tensor

GSASII.GSASIIlattice.**rotdMat**(*angle*, *axis=0*)

Prepare rotation matrix for angle in degrees about axis(=0,1,2)

Parameters

- **angle** – angle in degrees
- **axis** – axis (0,1,2 = x,y,z) about which for the rotation

Returns
rotation matrix - 3x3 numpy array

GSASII.GSASIIlattice.**rotdMat4**(*angle*, *axis=0*)

Prepare rotation matrix for angle in degrees about axis(=0,1,2) with scaling for OpenGL

Parameters

- **angle** – angle in degrees
- **axis** – axis (0,1,2 = x,y,z) about which for the rotation

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

GSASII.GSASIIlattice.**sec2HMS**(*sec*)

Convert time in sec to H:M:S string

Parameters
sec – time in seconds

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

GSASII.GSASIIlattice.**selections**(*items*, *n*)

take n (not necessarily distinct) items, order matters

GSASII.GSASIIlattice.**showCellSU**(*cellList*, *cellSig*, *SGData*, *cellNames=None*)

Produce the cell parameters from *getCellSU()* as a nicely formatted string

Parameters

- **cellList** (*list*) – a list of 7 items, a, b,… Vol, from getCellSU
- **cellSig** (*list*) – a list of 7 items, sig(a), sig(b),… sig(Vol), from getCellSU
- **SGdata** – space group information for current phase
- **cellNames** (*list*) – if specified, should be the labels to be used for a, b,… volume. Defaults to cellAlbl & 'vol', but for on-screen use, cellUlbl might be better than cellAlbl

GSASII.GSASIIlattice.**sortHKLd**(*HKLd*, *ifreverse*, *ifdup*, *ifSS=False*)

sort reflection list on d-spacing; can sort in either order

Parameters

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

Returns
sorted reflection list

GSASII.GSASIIlattice.**subVals**(*expr*, *A*, *T*)

Evaluate the symbolic expressions by substituting for A0-A5 & Tij

This can be used on the cell relationships created in *GenerateCellConstraints()* like this:

```
Trans = np.array([ [2/3, 4/3, 1/3], [-1, 0, 0], [-1/3, -2/3, 1/3] ])
T = np.linalg.inv(Trans).T
print([subVals(i,Aold,T) for i in GenerateCellConstraints()])
```

Parameters

- **expr** (*list*) – a list of sympy expressions.
- **A** (*list*) – This is the A* tensor as defined above.
- **T** (*np.array*) – a 3x3 transformation matrix where, Trans = np.array([[2/3, 4/3, 1/3], [-1, 0, 0], [-1/3, -2/3, 1/3]]) (for a' = 2/3a + 4/3b + 1/3c; b' = -a; c' = -1/3, -2/3, 1/3) then T = np.linalg.inv(Trans).T

Note that this is only used to do the symbolic math needed to generate cell relationships. It is not used normally in GSAS-II.

GSASII.GSASIIlattice.**symInner**(*M1*, *M2*)

Compute inner product of two square matrices with symbolic processing Use dot product because sympy does not define an inner product primitive

This requires that M1 & M2 be two sympy objects, as created in GenerateCellConstraints().

Note that this is only used to do the symbolic math needed to generate cell relationships. It is not used normally in GSAS-II.

GSASII.GSASIIlattice.**textureIndex**(*SHCoef*)

needs doc string

GSASII.GSASIIlattice.**transposeHKLF**(*transMat*, *Super*, *refList*)

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

GSASII.GSASIIlattice.**uniqueCombinations**(*items*, *n*)

take n distinct items, order is irrelevant

6.5 *GSASIIspc: Space Group Computations*

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

6.5.1 GSASIIspc Classes & Routines

GSASIIspc Classes & routines follow

GSASII.GSASIIspc.**AllOps**(*SGData*)

Returns a list of all operators for a space group, including those for centering and a center of symmetry

Parameters

SGData – from *SpcGroup()*

Returns

(SGTextList,offsetList,symOpList,G2oprList) where

- SGTextList: a list of strings with formatted and normalized symmetry operators.
- offsetList: a tuple of (dx,dy,dz) offsets that relate the GSAS-II symmetry operation to the operator in SGTextList and symOpList. these dx (etc.) values are added to the GSAS-II generated positions to provide the positions that are generated by the normalized symmetry operators.
- symOpList: a list of tuples with the normalized symmetry operations as (M,T) values (see SGOps in the *Space Group object*)
- G2oprList: a list with the GSAS-II operations for each symmetry operation as a tuple with (center,mult,opnum,opcode), where center is (0,0,0), (0.5,0,0), (0.5,0.5,0.5),…; where mult is 1 or -1 for the center of symmetry where opnum is the number for the symmetry operation, in SGOps (starting with 0) and opcode is mult*(100*icen+j+1).
- G2opcodes: a list with the name that GSAS-II uses for each symmetry operation (same as opcode, above)

GSASII.GSASIIspc.**ApplyStringOps**(*A*, *SGData*, *X*, *Uij=[]*)

Needs a doc string

GSASII.GSASIIspc.**ApplyStringOpsMom**(*A*, *SGData*, *SSGData*, *Mom*)

Applies string operations to modulated magnetic moment components used in drawing Drawing matches Bilbao MVISUALIZE

GSASII.GSASIIspc.**AtomDxSymFix**(*Dx*, *SytSym*, *CSIX*)

Applies site symmetry restrictions to atom position shifts. 1st parameter value of each kind encountered is assumed to be the independent one. Needed for ISODISTORT mode shifts.

GSASII.GSASIIspc.**CheckSpin**(*isym*, *SGData*)

Check for exceptions in spin rules

GSASII.GSASIIspc.**CompareSym**(*symList*, *sgName=None*, *SGData=None*)

Compare symmetry generated by GSAS-II from a space group name with a list of operators from some other source.

Parameters

- **symList** (*list*) – a list of symmetry operations (such as 'x,y,z' or '-Y,X,Z'). The code is fairly forgiving in that 1/2-X or X-1/2 or even -X+0.5 can all be accepted. This is typically read from a CIF.
- **sgName** (*str*) – a space group name. Need not follow GSAS-II convention for spaces, etc (see `GSASIIspc.StandardizeSpcName()`)
- **SGData** – a GSAS-II symmetry objectspace group name. Need not follow GSAS-II convention for spaces, etc (see `GSASIIspc.StandardizeSpcName()`)

GSASII.GSASIIspc.**ElemPosition**(*SGData*)

Under development. Object here is to return a list of symmetry element types and locations suitable for say drawing them. So far I have the element type… getting all possible locations without lookup may be impossible!

GSASII.GSASIIspc.**GenAtom**(*XYZ*, *SGData*, *All=False*, *Uij=[]*, *Move=True*)

Generates the equivalent positions for a specified coordinate and space group

Parameters

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

Returns

[[XYZEquiv],Idup,[UijEquiv],spnflp]

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

GSASII.GSASIIspc.**GenHKL**(*HKL*, *SGData*)

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

GSASII.GSASIIspc.**GenHKLf**(*HKL*, *SGData*)

Uses old GSAS Fortran routine genhkl.for

Parameters

- **HKL** – [h,k,l] must be integral values for genhkl.for to work
- **SGData** – space group data obtained from SpcGroup

Returns

iabsnt,mulp,Uniq,phi

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

GSASII.GSASIIspc.**GetCSpqinel**(*SpnFlp*, *dupDir*)

returns Mxyz terms, multipliers, GUI flags

GSASII.GSASIIspc.**GetCSuinel**(*siteSym*)

returns Uij terms, multipliers, GUI flags & Uiso2Uij multipliers

GSASII.GSASIIspc.**GetCSxinel**(*siteSym*)

returns Xyz terms, multipliers, GUI flags

GSASII.GSASIIspc.**GetGenSym**(*SGData*)

Get the space group generator symbols :param SGData: from *SpcGroup()* LaueSym = ('-1','2/m','mmm','4/m','4/mmm','3R','3mR','3','3m1','31m','6/m','6/mmm','m3','m3m') LattSym = ('P','A','B','C','I','F','R')

GSASII.GSASIIspc.**GetHallSpaceGroup**(*SGData*)

Determine the Hall space group symbol for a GSAS-II space group object, if it exists (it will not if non-standard centering is used, perhaps also for other cases). Will return None if not found.

GSASII.GSASIIspc.**GetKNsym**(*key*)

Needs a doc string

GSASII.GSASIIspc.**GetLittleGrpOps**(*SGData*, *vec*)

Find rotation part of operators that leave vec unchanged

Parameters

- **SGData** – space group data structure as defined in SpcGroup above.
- **vec** – a numpy array of fractional vector coordinates

Returns

Little - list of operators [M,T] that form the little gropu

GSASII.GSASIIspc.**GetNXUPQsym**(*siteSym*)

The codes XUPQ are for lookup of symmetry constraints for position(X), thermal parm(U) & magnetic moments (P & Q)

GSASII.GSASIIspc.**GetOpFromCode**(*Code*, *SGData*)

takes Sym Op code from Draw Atoms table & returns space group operation details

GSASII.GSASIIspc.**GetOprName**(*key*)

Needs a doc string

GSASII.GSASIIspc.**GetOprPtrName**(*key*)

Needs a doc string

GSASII.GSASIIspc.**GetOprPtrNumber**(*key*)

Needs a doc string

GSASII.GSASIIspc.**GetSGSpin**(*SGData*, *MSgSym*)

get spin generators from magnetic space group symbol

GSASII.GSASIIspc.**GetSytSymChoice**(*sytsym*)

Get local sytsyms compatible with crystallographic sytsym - may be higher in sub-super point groups

Parameters

sytsym (*str*) – site symmetry symbol (such as '-1', '222', '2/m(z)')

Returns

list of allowed point groups

GSASII.GSASIIspc.**HStrainNames**(*SGData*)

Needs a doc string

GSASII.GSASIIspc.**Latt2text**(*Cen*)

From lattice centering vectors returns ';' delimited cell centering vectors

GSASII.GSASIIspc.**MT2text**(*Opr*, *reverse=False*)

From space group matrix/translation operator returns text version

GSASII.GSASIIspc.**MagSSText2MTS**(*Opr*, *G2=False*)

From magnetic super space group cif text returns matrix/translation + spin flip

GSASII.GSASIIspc.**MagSytSym**(*SytSym*, *dupDir*, *SGData*)

site sym operations: 1,-1,2,3,-3,4,-4,6,-6,m need to be marked if spin inversion

GSASII.GSASIIspc.**MagText2MTS**(*mcifOpr*, *CIF=True*)

From magnetic space group cif text returns matrix/translation + spin flip

GSASII.GSASIIspc.**MoveToUnitCell**(*xyz*)

Translates a set of coordinates so that all values are >=0 and < 1

Parameters

xyz – a list or numpy array of fractional coordinates

Returns

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

GSASII.GSASIIspc.**Muiso2Shkl**(*muiso*, *SGData*, *cell*)

this is to convert isotropic mustrain to generalized Shkls

GSASII.GSASIIspc.**MustrainCoeff**(*HKL*, *SGData*)

Needs a doc string

GSASII.GSASIIspc.**MustrainNames**(*SGData*)

Needs a doc string

GSASII.GSASIIspc.**Opposite**(*XYZ*, *toler=0.0002*)

Gives opposite corner, edge or face of unit cell for position within tolerance.
Result may be just outside the cell within tolerance

Parameters

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

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

GSASII.GSASIIspc.**ParseXYZ**(*sym*)

Parse a set of space group operations, such as '-x,-y,-z' or 'x-y,z,-x' etc. into an algebraic form returning a 3x3 matrix, M, and a length 3 vector, O, where the symmetry operation can be applied as

M . (x,y,z) + O

where M is a 3x3 matrix and O is a displacement vector. This is the same form generated by *SpcGroup()*.

Note that this code will process offsets either before the coordinate multipliers or after, so that either of these

1/2-y,1/2+x,z or -y+1/2,x-0.5,z

will both be parsed properly. Returns None if a symbol cannot be parsed.

GSASII.GSASIIspc.**SGErrors**(*IErr*)

Interprets the error message code from SpcGroup. Used in SpaceGroup.

Parameters
IErr – see SGError in *SpcGroup()*

Returns
ErrString - a string with the error message or "Unknown error"

GSASII.GSASIIspc.**SGPrint**(*SGData*, *AddInv=False*)

Print the output of SpcGroup in a nicely formatted way. Used in SpaceGroup

Parameters
SGData – from *SpcGroup()*

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

GSASII.GSASIIspc.**SGProd**(*OpA*, *OpB*)

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

GSASII.GSASIIspc.**SGPtGroup**(*SGData*)

Determine point group of the space group - done after space group symbol has been evaluated by SpcGroup. Only short symbols are allowed

Parameters
SGData – from *SpcGroup()*

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

GSASII.GSASIIspc.**SGpolar**(*SGData*)

Determine identity of polar axes if any

GSASII.GSASIIspc.**SSChoice**(*SGData*)

Gets the unique set of possible super space groups for a given space group

GSASII.GSASIIspc.**SSGModCheck**(*Vec*, *modSymb*, *newMod=True*)

Checks modulation vector compatibility with supersymmetry space group symbol. if newMod: Superspace group symbol takes precidence & the vector will be modified accordingly

GSASII.GSASIIspc.**SSGPrint**(*SGData*, *SSGData*, *AddInv=False*)

Print the output of SSpcGroup in a nicely formatted way. Used in SSpaceGroup

Parameters

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

Returns

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

GSASII.GSASIIspc.**SSLatt2text**(*SSGCen*)

Lattice centering vectors to text

GSASII.GSASIIspc.**SSMT2text**(*Opr*)

From superspace group matrix/translation operator returns text version

GSASII.GSASIIspc.**SSpaceGroup**(*SGSymbol*, *SSymbol*)

Print the output of SSpcGroup in a nicely formatted way.

Parameters

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

Returns

nothing

GSASII.GSASIIspc.**SSpcGroup**(*SGData*, *SSymbol*)

Determines supersymmetry information from superspace group name; currently only for (3+1) superlattices

Parameters

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

Returns

(SSGError,SSGData)

- SGError = 0 for no errors; >0 for errors (see SGErrors below for details)
- SSGData - is a dict (see *Superspace Group object*) with entries:
  - 'SSpGrp': full superspace group symbol, accidental spaces removed; for display only
  - 'SSGCen': 4D cell centering vectors [0,0,0,0] at least
  - 'SSGOps': 4D symmetry operations as [M,T] so that M*x+T = x'

GSASII.GSASIIspc.**SpaceGroup**(*SGSymbol*)

Print the output of SpcGroup in a nicely formatted way.

Parameters
SGSymbol – space group symbol (string) with spaces between axial fields

Returns
nothing

GSASII.GSASIIspc.**SpaceGroupNumber**(*spcgroup*)

Determine the space group number for a space group from H-M name. Will work for non-standard groups insofar as we can normalize – far from perfect.

GSASII.GSASIIspc.**SpcGroup**(*SGSymbol*)

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

Parameters
SGSymbol – space group symbol (string) with spaces between axial fields

Returns

(SGError,SGData)

- SGError = 0 for no errors; >0 for errors (see *SGErrors()* for details)
- SGData - is a dict (see *Space Group object*) with entries:
  - 'SpGrp': space group symbol, slightly cleaned up
  - 'SGFixed': True if space group data can not be changed, e.g. from magnetic cif; otherwise False
  - 'SGGray': True if 1' in symbol - gray group for mag. incommensurate phases
  - 'SGLaue': one of '-1', '2/m', 'mmm', '4/m', '4/mmm', '3R', '3mR', '3', '3m1', '31m', '6/m', '6/mmm', 'm3', 'm3m'
  - 'SGInv': boolean; True if centrosymmetric, False if not
  - 'SGLatt': one of 'P', 'A', 'B', 'C', 'I', 'F', 'R'
  - 'SGUniq': one of 'a', 'b', 'c' if monoclinic, '' otherwise
  - 'SGCen': cell centering vectors [0,0,0] at least
  - 'SGOps': symmetry operations as [M,T] so that M*x+T = x'
  - 'SGSys': one of 'triclinic', 'monoclinic', 'orthorhombic', 'tetragonal', 'rhombohedral', 'trigonal', 'hexagonal', 'cubic'
  - 'SGPolax': one of ' ', 'x', 'y', 'x y', 'z', 'x z', 'y z', 'xyz', '111' for arbitrary axes
  - 'SGPtGrp': one of 32 point group symbols (with some permutations), which is filled by SGPtGroup, is external (KE) part of supersymmetry point group
  - 'SSGKl': default internal (Kl) part of supersymmetry point group; modified in supersymmetry stuff depending on chosen modulation vector for Mono & Ortho
  - 'BNSlattsym': BNS lattice symbol & cenering op - used for magnetic structures

GSASII.GSASIIspc.**StandardizeSpcName**(*spcgroup*)

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

GSASII.GSASIIspc.**StringOpsProd**(*A*, *B*, *SGData*)

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

- 'SGCen': cell centering vectors [0,0,0] at least
- 'SGOps': symmetry operations as [M,T] so that M*x+T = x'

GSASII.GSASIIspc.**SytSym**(*XYZ*, *SGData*)

Generates the number of equivalent positions and a site symmetry code for a specified coordinate and space group

Parameters

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

Returns

a four element tuple:

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

GSASII.GSASIIspc.**Text2MT**(*mcifOpr*, *CIF=True*)

From space group cif text returns matrix/translation

GSASII.GSASIIspc.**TextOps**(*text*, *table*, *reverse=False*)

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

GSASII.GSASIIspc.**Trans2Text**(*Trans*)

from transformation matrix to text

GSASII.GSASIIspc.**UpdateSytSym**(*Phase*)

Update site symmetry/site multiplicity after space group/BNS lattice change

GSASII.GSASIIspc.**altSettingOrtho = {}**

A dictionary of alternate settings for orthorhombic unit cells

GSASII.GSASIIspc.**checkHKLextc**(*HKL*, *SGData*)

Checks if reflection extinct - does not check centering

Parameters

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

Returns

True if extinct; False if allowed

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

Checks if reflection magnetically extinct; does fullcheck (centering, too) uses algorthm from Gallego, et al., J. Appl. Cryst. 45, 1236-1247 (2012)

Parameters

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

Returns

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

GSASII.GSASIIspc.**fixMono**(*SpGrp*)

fixes b-unique monoclinics in e.g. P 1 2/1c 1 –> P 21/c

GSASII.GSASIIspc.**fullHM2shortHM**(*SpcGp*)

Accepts a full H-M space group symbol and returns a short H-M symbol that the space group interpreter can translate

GSASII.GSASIIspc.**offsetNorm**(*x*)

Translate a coordinate (or vector of symmetry offsets) into the range -1/6 < x <= 5/6, matching GSAS-II use.

GSASII.GSASIIspc.**sgequiv_2002_orthorhombic = {}**

A dictionary of orthorhombic space groups that were renamed in the 2002 Volume A, along with the pre-2002 name. The e designates a double glide-plane

GSASII.GSASIIspc.**spg2origins = {'A b a a': [-0.25, 0, -0.25], 'A c a a': [-0.25, -0.25, 0], 'B b a b': [0, -0.25, -0.25], 'B b c b': [-0.25, -0.25, 0], 'C c c a': [0, -0.25, -0.25], 'C c c b': [-0.25, 0, -0.25], 'F d - c': [-0.375, -0.375, -0.375], 'F d -3': [-0.125, -0.125, -0.125], 'F d -3 c': [-0.375, -0.375, -0.375], 'F d -3 m': [-0.125, -0.125, -0.125], 'F d 3': [-0.125, -0.125, -0.125], 'F d 3 m': [-0.125, -0.125, -0.125], 'F d d d': [-0.125, -0.125, -0.125], 'I 41/a': [0, -0.25, -0.125], 'I 41/a c d': [0, 0.25, -0.125], 'I 41/a m d': [0, 0.25, -0.125], 'P 4/n': [-0.25, -0.25, 0], 'P 4/n b m': [-0.25, -0.25, 0], 'P 4/n c c': [-0.25, -0.25, 0], 'P 4/n m m': [-0.25, -0.25, 0], 'P 4/n n c': [-0.25, -0.25, -0.25], 'P 42/n': [-0.25, -0.25, -0.25], 'P 42/n b c': [-0.25, -0.25, -0.25], 'P 42/n c m': [-0.25, 0.25, -0.25], 'P 42/n m c': [-0.25, 0.25, -0.25], 'P 42/n n m': [-0.25, 0.25, -0.25], 'P b a n': [-0.25, -0.25, 0], 'P c n a': [-0.25, 0, -0.25], 'P m m n': [-0.25, -0.25, 0], 'P m n m': [-0.25, 0, -0.25], 'P n -3 m': [-0.25, -0.25, -0.25], 'P n -3 n': [-0.25, -0.25, -0.25], 'P n 3 m': [-0.25, -0.25, -0.25], 'P n 3 n': [-0.25, -0.25, -0.25], 'P n c b': [0, -0.25, -0.25], 'P n m m': [0, -0.25, -0.25], 'P n n n': [-0.25, -0.25, -0.25], 'p n -3': [-0.25, -0.25, -0.25], 'p n 3': [-0.25, -0.25, -0.25]}**

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

GSASII.GSASIIspc.**spgHall = []**

Hall space group symbols indexed by GSAS-II space group name. This is indexed by a key which is generated from the name used in GSAS-II with spaces removed and all letters in lowercase. The value associated with the key consists of a list of two values: the first is the Hall name and the second is a Hermann-Mauguin name. Note that there may be several names used by GSAS-II that map to the same symmetry operators and the same Hall symbol (for example “P 21” and “P 1 21 1” and “P 63/m” and “P 63/m H”). There are also space group names that GSAS-II will accept that do not have Hall symbols (such as “I -1” or “F 21 21 21”).

GSASII.GSASIIspc.**spgbyNum = []**

Space groups indexed by number

GSASII.GSASIIspc.**spglist = {}**

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

GSASII.GSASIIspc.**splitSSsym**(*SSymbol*)

Splits supersymmetry symbol into two lists of strings

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

Module with miscellaneous routines for input and output from files.

6.6.1 GSASIIfiles Classes & Routines

Code for accessing files, including support for reading and writing instrument parameter files and exporting various types of data files.

This module has some routines that require wxPython, but imports for wx and GSAS-II GUI routines is done on a per-function basis so that this module can be imported for GSASIIscriptable use when wx is not installed.

GSASII.GSASIIfiles.**CleanupFromZip**(*label*, *cleanupList*)

Delete files extracted from a zip archive, typically created with GSASIIctrl.ExtractFileFromZip() during the data import process.

Note that image files should not be deleted as they will be reused every time the image is displayed, but everything else will be saved in the data tree and the file copy is not needed.

Parameters

- **label** (*str*) – for imports, this is the type of file being read
- **cleanupList** (*list*) – a list of files that have been extracted from the zip archive and can be deleted.

class GSASII.GSASIIfiles.**ExportBaseclass**(*G2frame*, *formatName*, *extension*, *longFormatName=None*)

Defines a base class for the exporting of GSAS-II results.

This class is subclassed in the various exports/G2export_*.py files. Those files are imported in GSASIIdataGUI.GSASII._init_Exports() which defines the appropriate menu items for each one and the .Exporter method is called directly from the menu item.

Routines may also define a .Writer method, which is used to write a single file without invoking any GUI objects.

CloseFile(*fp=None*)

Close a file opened in OpenFile

Parameters

fp (*file*) – the file object to be closed. If None (default) file object self.fp is closed.

ExportSelect(*AskFile='ask'*)

Selects histograms or phases when needed. Sets a default file name when requested into self.filename; always sets a default directory in self.dirname.

Parameters

AskFile (*bool*) – Determines how this routine processes getting a location to store the current export(s).

- if AskFile is 'ask' (default option), get the name of the file to be written; self.filename and self.dirname are always set. In the case where multiple files must be generated, the export routine should do this based on self.filename as a template.
- if AskFile is 'dir', get the name of the directory to be used; self.filename is not used, but self.dirname is always set. The export routine will always generate the file name.

- if AskFile is 'single', get only the name of the directory to be used when multiple items will be written (as multiple files) are used *or* a complete file name is requested when a single file name is selected. self.dirname is always set and self.filename used only when a single file is selected.
- if AskFile is 'default', creates a name of the file to be used from the name of the project (.gpx) file. If the project has not been saved, then the name of file is requested. self.filename and self.dirname are always set. In the case where multiple file names must be generated, the export routine should do this based on self.filename.
- if AskFile is 'default-dir', sets self.dirname from the project (.gpx) file. If the project has not been saved, then a directory is requested. self.filename is not used.

Returns
True in case of an error

GetAtoms(*phasenam*)

Gets the atoms associated with a phase. Can be used with standard or macromolecular phases

Parameters
phasenam (*str*) – the name for the selected phase

Returns

a list of items for eac atom where each item is a list containing: label, typ, mult, xyz, and td, where

- label and typ are the atom label and the scattering factor type (str)
- mult is the site multiplicity (int)
- xyz is contains a list with four pairs of numbers: x, y, z and fractional occupancy and their standard uncertainty (or a negative value)
- td is contains a list with either one or six pairs of numbers: if one number it is U_{iso} and with six numbers it is U_{11}, U_{22}, U_{33}, U_{12}, U_{13} & U_{23} paired with their standard uncertainty (or a negative value)

GetCell(*phasenam*, *unique=False*)

Gets the unit cell parameters and their s.u.'s for a selected phase

Parameters

- **phasenam** (*str*) – the name for the selected phase
- **unique** (*bool*) – when True, only directly refined parameters (a in cubic, a & alpha in rhombohedral cells) are assigned positive s.u. values. Used as True for CIF generation.

Returns
cellList,cellSig where each is a 7 element list corresponding to a, b, c, alpha, beta, gamma, volume where *cellList* has the cell values and *cellSig* has their uncertainties.

GetSeqCell(*phasenam*, *data_name*)

Gets the unit cell parameters and their s.u.'s for a selected phase and histogram in a sequential fit

Parameters

- **phasenam** (*str*) – the name for the selected phase
- **data_name** (*dict*) – the sequential refinement parameters for the selected histogram

Returns
cellList,cellSig where each is a 7 element list corresponding to a, b, c, alpha, beta, gamma, volume where *cellList* has the cell values and *cellSig* has their uncertainties.

InitExport(*event*)

Determines the type of menu that called the Exporter and misc initialization.

MakePWDRfilename(*hist*)

Make a filename root (no extension) from a PWDR histogram name

Parameters
hist (*str*) – the histogram name in data tree (starts with "PWDR ")

OpenFile(*fil=None*, *mode='w'*, *delayOpen=False*)

Open the output file

Parameters

- **fil** (*str*) – The name of the file to open. If None (default) the name defaults to self.dirname + self.filename. If an extension is supplied, it is not overridded, but if not, the default extension is used.
- **mode** (*str*) – The mode can 'w' to write a file, or 'a' to append to it. If the mode is 'd' (for debug), output is displayed on the console.

Returns
the file object opened by the routine which is also saved as self.fp

SetSeqRef(*data*, *hist*)

Set the exporter to retrieve results from a sequential refinement rather than the main tree

Write(*line*)

write a line of output, attaching a line-end character

Parameters
line (*str*) – the text to be written.

askSaveDirectory()

Ask the user to supply a directory name. Path name is used as the starting point for the next export path search.

Returns
a directory name (str) or None if Cancel is pressed

TODO: Can this be replaced with G2G.askSaveDirectory?

askSaveFile()

Ask the user to supply a file name

Returns
a file name (str) or None if Cancel is pressed

dumpTree(*mode='type'*)

Print out information on the data tree dicts loaded in loadTree. Used for testing only.

loadParmDict(*computeSU=False*)

Load the GSAS-II refinable parameters from the tree into a dict (self.parmDict). Update refined values to those from the last cycle and set the uncertainties for the refined parameters in another dict (self.sigDict).

Expands the parm & sig dicts to include values derived from constraints.

This could be made faster for sequential fits as info for each histogram is loaded later when iterating over histograms.

loadTree()

Load the contents of the data tree into a set of dicts (self.OverallParms, self.Phases and self.Histogram as well as self.powderDict & self.xtalDict)

- The childrenless data tree items are overall parameters/controls for the entire project and are placed in self.OverallParms
- Phase items are placed in self.Phases
- Data items are placed in self.Histogram. The key for these data items begin with a keyword, such as PWDR, IMG, HKLF,… that identifies the data type.

GSASII.GSASIIfiles.**FormatPadValue**(*val*, *maxdigits=None*)

Format a float to fit in `maxdigits[0]` spaces with maxdigits[1] after decimal.

Parameters

- **val** (*float*) – number to be formatted.
- **maxdigits** (*list*) – the number of digits & places after decimal to be used for display of the number (defaults to [10,2]).

Returns

a string with exactly maxdigits[0] characters (except under error conditions), but last character will always be a space

GSASII.GSASIIfiles.**FormatSigFigs**(*val*, *maxdigits=10*, *sigfigs=5*, *treatAsZero=1e-20*)

Format a float to use `maxdigits` or fewer digits with `sigfigs` significant digits showing (if room allows).

Parameters

- **val** (*float*) – number to be formatted.
- **maxdigits** (*int*) – the number of digits to be used for display of the number (defaults to 10).
- **sigfigs** (*int*) – the number of significant figures to use, if room allows
- **treatAsZero** (*float*) – numbers that are less than this in magnitude are treated as zero. Defaults to 1.0e-20, but this can be disabled if set to None.

Returns

a string with <= maxdigits characters (I hope).

GSASII.GSASIIfiles.**FormatValue**(*val*, *maxdigits=None*)

Format a float to fit in at most `maxdigits[0]` spaces with maxdigits[1] after decimal. Note that this code has been hacked from FormatSigFigs and may have unused sections.

Parameters

- **val** (*float*) – number to be formatted.
- **maxdigits** (*list*) – the number of digits, places after decimal and 'f' or 'g' to be used for display of the number (defaults to [10,2,'f']).

Returns

a string with <= maxdigits characters (usually).

GSASII.GSASIIfiles.**FormulaEval**(*string*)

Evaluates a algebraic formula into a float, if possible. Works properly on fractions e.g. 2/3 only with python 3.0+ division.

Expressions such as 2/3, 3*pi, sin(45)/2, 2*sqrt(2), 2**10 can all be evaluated.

Parameters
string (*str*) – Character string containing a Python expression to be evaluated.

Returns
the value for the expression as a float or None if the expression does not evaluate to a valid number.

GSASII.GSASIIfiles.**G2Print**(**args*, ***kwargs*)

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

Parameters
mode (*str*) – if specified, this should contain the mode for printing ('error', 'warn' or anything else). If not specified, the first argument of the print command (args[0]) should contain the string 'error' for error messages and 'warn' for warning messages (capitalization and additional letters ignored.)

GSASII.GSASIIfiles.**G2SetPrintLevel**(*level*)

Set the level of output from calls to *G2Print()*, which should be used in place of print() within GSASII. Settings for the mode are 'all', 'warn', 'error' or 'none'

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

GSASII.GSASIIfiles.**G2printLevel = 'all'**

This defines the level of output from calls to `GSASIIfiles.G2Print()`, which should be used in place of print() within GSASII where possible. Settings for this are 'all', 'warn', 'error' or 'none'. Best to change this with *G2SetPrintLevel()*.

See also
G2Print() G2SetPrintLevel().

GSASII.GSASIIfiles.**GetColumnMetadata**(*reader*)

Add metadata to an image from a column-type metadata file using *readColMetadata()*

Parameters
reader – a reader object from reading an image

GSASII.GSASIIfiles.**GetImageData**(*G2frame*, *imagefile*, *imageOnly=False*, *ImageTag=None*, *FormatName=''*)

Read a single image with an image importer. This is called to reread an image after it has already been imported with `GSASIIdataGUI.GSASII.OnImportGeneric()` (or `GSASIImiscGUI.ReadImages()` in Auto Integration) so it is not necessary to reload metadata.

Parameters
- **G2frame** (*wx.Frame*) – main GSAS-II Frame and data object.
- **imagefile** (*str*) – name of image file
- **imageOnly** (*bool*) – If True return only the image. Formerly, if False return more information (see below). Now must be True
- **ImageTag** (*int/str*) – specifies a particular image to be read from a file. First image is read if None (default).
- **formatName** (*str*) – the image reader formatName

Returns
an image as a numpy array. Formerly if imageOnly=False this would return a list of four items: Comments, Data, Npix and the Image, but now an exception occurs when imageOnly=False.

GSASII.GSASIIfiles.**ImportErrorMsg**(*errormsg=None*, *pkg={}*)

Store error message(s) from loading importers (usually missing packages. Or, report back all messages, if called with no argument.

Parameters

- **errormsg** (*str*) – a string containing the error message. If not supplied, the function returns the error message(s).
- **pkg** (*dict*) – a dict where the key is the name of the importer and the value is a list containing the packages that need to be installed to allow the importer to be used.

Returns

the error messages as a list (an empty list if there are none), only if errormsg is None (the default).

GSASII.GSASIIfiles.**LoadControls**(*Slines*, *data*)

Read values from a .imctrl (Image Controls) file

GSASII.GSASIIfiles.**LoadExportRoutines**(*parent*, *usetraceback=False*)

Loads all export routines from the GSASII.exports area and from a user-supplied area, ~/.GSASII/exports.

Parameters

- **parent** – a reference to the main GSAS-II window when called from the GUI or None when called from GSASIIscriptable
- **usetraceback** – if True (used for debug only) a traceback is shown when accessing the exporter object fails

Returns

a list of exporter routines (callable objects)

GSASII.GSASIIfiles.**LoadImportRoutines**(*prefix*, *errprefix=None*, *traceback=False*)

Loads import routines from the GSASII.imports area and from a user-supplied area, ~/.GSASII/imports.

Parameters

- **prefix** (*str*) – string used in the file name to differentiate the importer type (e.g. *G2img_xyz.py* is an importer for images. Will be one of 'img' (images), 'pdf' (pair distribution function), 'phase' (cell/coordinates), 'pwd' (powder diffraction), 'rfd' (reflectivity), 'sad' (small-angle scattering) or 'sfact' (single crystal).
- **errprefix** – no longer used
- **traceback** – no longer used

Returns

a list of reader routines (callable objects)

GSASII.GSASIIfiles.**NeededPackage**(*pkgDict*)

Store packages that are needed to add functionality to GSAS-II

Parameters

pkgDict (*dict*) – a dict where the key is the describes the routine or the unavailable functionality that requires addition of a Python package and the value is a list containing the packages that need to be installed. If a specific version of a package is required then indicate that by placing version information into the name (see https://docs.conda.io/projects/conda/en/4.6.0/_downloads/52a95608c49671267e40c689e0bc00ca/conda-cheatsheet.pdf). Note that the names containing "pkg=x.y" will be translated to "pkg==x.y" for pip installs, but more complex package specifications (see https://pip.pypa.io/en/latest/reference/requirement-specifiers/) will probably work only for conda installations.

Examples:

```
{'MIDAS Zarr importer':['zarr=3.18.*']}
{'HDF5 image importer':['h5py','hdf5']}
```

GSASII.GSASIIfiles.**PDFWrite**(*PDFentry*, *fileroot*, *PDFsaves*, *PDFControls*, *Inst={}*, *Limits=[]*)

Write PDF-related data (G(r), S(Q),…) into files, as selected.

Parameters

- **PDFentry** (*str*) – name of the PDF entry in the tree. This is used for comments in the file specifying where it came from; it can be arbitrary
- **fileroot** (*str*) – name of file(s) to be written. The extension will be ignored.
- **PDFsaves** (*list*) – flags that determine what type of file will be written: PDFsaves[0], if True writes a I(Q) file with a .iq extension PDFsaves[1], if True writes a S(Q) file with a .sq extension PDFsaves[2], if True writes a F(Q) file with a .fq extension PDFsaves[3], if True writes a G(r) file with a .gr extension PDFsaves[4], if True writes G(r) in a pdfGUI input file with a .gr extension. Note that if PDFsaves[3] and PDFsaves[4] are both True, the pdfGUI overwrites the G(r) file. PDFsaves[5], if True writes F(Q) & g(R) with .fq & .gr extensions overwrites these if selected by option 2, 3 or 4
- **PDFControls** (*dict*) – The PDF parameters and computed results
- **Inst** (*dict*) – Instrument parameters from the PDWR entry used to compute the PDF. Needed only when PDFsaves[4] is True.
- **Limits** (*list*) – Computation limits from the PDWR entry used to compute the PDF. Needed only when PDFsaves[4] is True.

GSASII.GSASIIfiles.**ReadInstprm**(*instLines*, *bank*, *Sample={}*)

Read contents of a GSAS-II (new) .instprm instrument parameter file

Parameters

- **instLines** (*list*) – contents of GSAS-II parameter file as a list of str; N.B. lines can be concatenated with ';'
- **bank** (*int*) – bank number to use when instprm file has values for multiple banks (noted by headers of '#BANK n:…'.). This is ignored for instprm files without those headers. If bank is None with multiple banks, the first bank is used. Note that multibank .instprm files are made by a "Save all profile" command in Instrument Parameters.
- **Sample** (*dict*) – A dict containing sample parameters, typically corresponding to rd.Sample, where rd is a reader object that is being read from. Sample parameters determined by instrument settings or information from the instprm file are placed here.

Returns

bank,instdict where bank is the sample parameter set number and instdict is a list containing the regular instrument parameter dict and the "extended" dict, currently only containing pdabc entries (if they exist)

Note if 'Type' is set as Debye-Scherrer or Bragg-Brentano this will be used and will set defaults in the sample parameters. Otherwise, a single-wavelength file will set Debye-Scherrer mode and dual wavelength will set Bragg-Brentano.

GSASII.GSASIIfiles.**RereadImageData**(*ImageReaderlist*, *imagefile*, *ImageTag=None*, *FormatName=''*)

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

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

Parameters

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

Returns

an image as a numpy array

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

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

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

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

GSASII.GSASIIfiles.**WriteInstprm**(*fp*, *InstPrm*, *InstPrm1*, *Sample={}*, *bank=None*)

Write the contents of a GSAS-II (new) .instprm instrument parameter file ToDo: use this inside G2frame.OnSave and G2frame.OnSaveAll

Parameters

- **fp** (*file*) – Pointer to open file to be written.
- **InstPrm** (*dict*) – Instrument parameters
- **InstPrm1** (*dict*) – "extended" instrument parameters
- **Sample** (*dict*) – Sample parameters (optional)
- **bank** (*int*) – Bank number. If not None (default), this causes a "#Bank" heading to be placed in the file before the parameters are written.

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

Evaluate the metadata for a line in the .par file

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

find 1st occurance of file in path

GSASII.GSASIIfiles.**makePdabcDict**(*pdabcString*)

if a pdabc entry is found in an instprm file, it will exist as a single string each row is delineated by a newline and must contains 5 comma separated values specifying d,TOF,alp,bet,sig.

This function will parse that string into a dictionary and then return it within a container dictionary

GSASII.GSASIIfiles.**openInNewTerm**(*project=None*, *g2script=None*, *pythonapp='/home/docs/checkouts/readthedocs.org/user_builds/gsas-ii/conda/latest/bin/python'*)

Open a new and independent GSAS-II session in separate terminal or console window and as a separate process that will continue even if the calling process exits. Intended to work on all platforms.

This could be used to run other scripts inside python other than GSAS-II

Parameters

- **project** (*str*) – the name of an optional parameter to be passed to the script (usually a .gpx file to be opened in a new GSAS-II session)

- **g2script** (*str*) – the script to be run. If None (default) the G2.py file in the same directory as this file will be used.
- **pythonapp** (*str*) – the Python interpreter to be used. Defaults to sys.executable which is usually what is wanted.
- **terminal** (*str*) – a name for a preferred terminal emulator

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

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

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

Parameters

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

Returns

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

The contents of the label file will look like this:

```
# define keywords
filename:lambda x,y: "{}_{:0>6}".format(x,y)|33,34
distance: float | 75
wavelength:lambda keV: 12.398425/float(keV)|9
pixelSize:lambda x: [74.8, 74.8]|0
ISOlikeDate: lambda dow,m,d,t,y:"{}-{}-{}T{} ({})".format(y,m,d,t,dow)|0,1,2,3,4
Temperature: float|53
FreePrm2: int | 34 | Free Parm2 Label
# define other variables
0:day
1:month
2:date
3:time
4:year
7:I_ring
```

This file contains three types of lines in any order.

- Named parameters are evaluated with user-supplied Python code (see subsequent information). Specific named parameters are used to determine values that are used for image interpretation (see table, below). Any others are copied to the Comments subsection of the Image tree item.
- Column labels are defined with a column number (integer) followed by a colon (:) and a label to be assigned to that column. All labeled columns are copied to the Image's Comments subsection.
- Comments are any line that does not contain a colon.

Note that columns are numbered starting at zero.

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

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

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

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

The examples above are discussed here:

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

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

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

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

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

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

`distance: float | 75`
Here the contents of column 75 will be converted to a floating point number by calling float on it. Note that the spaces here are ignored.

`wavelength:lambda keV: 12.398425/float(keV)|9`
Here we define an algebraic expression to convert an energy in keV to a wavelength and pass the contents of column 9 as that input energy

`pixelSize:lambda x: [74.8, 74.8]|0`
In this case the pixel size is a constant (a list of two numbers). The first column is passed as an argument as at least one argument is required, but that value is not used in the expression.

`ISOlikeDate: lambda dow,m,d,t,y:"{}-{}-{}T{} ({})".format(y,m,d,t,dow)|0,1,2,3,4`
This example defines a parameter that takes items in the first five columns and formats them in a different way. This parameter is not one of the pre-defined parameter names below. Some external code could be used to change the month string (argument `m`) to a integer from 1 to 12.

`FreePrm2: int | 34 | Free Parm2 Label`
In this example, the contents of column 34 will be converted to an integer and placed as the second free-named parameter in the Sample Parameters after an integration. The label for this parameter will be changed to "Free Parm2 Label".

Pre-defined parameter names

keyword	required	type	Description
filename	yes	str or list	generates the file name prefix for the matching image file (MyImage001 for file /tmp/MyImage001.tif) or a list of file names.
polarization	no	float	generates the polarization expected based on the monochromator angle, defaults to 0.99.
center	no	list of 2 floats	generates the approximate beam center on the detector in mm, such as [204.8, 204.8].
distance	yes	float	generates the distance from the sample to the detector in mm
pixelSize	no	list of 2 floats	generates the size of the pixels in microns such as [200.0, 200.0].
wavelength	yes	float	generates the wavelength in Angstroms

GSASII.GSASIIfiles.**readColMetadataLabels**(*lblFil*)

Read the .*lbls file and setup for metadata assignments

GSASII.GSASIIfiles.**readMasks**(*filename*, *masks*, *ignoreThreshold*)

Read a GSAS-II masks file

GSASII.GSASIIfiles.**sfloat**(*S*)

Convert a string to float. An empty field or a unconvertable value is treated as zero

GSASII.GSASIIfiles.**striphist**(*var*, *insChar=''*)

strip a histogram number from a var name

GSASII.GSASIIfiles.**trim**(*val*)

Simplify a string containing leading and trailing spaces as well as newlines, tabs, repeated spaces etc. into a shorter and more simple string, by replacing all ranges of whitespace characters with a single space.

Parameters

val (*str*) – the string to be simplified

Returns

the (usually) shortened version of the string

6.7 *GSASIImpsubs: routines used in multiprocessing*

6.7.1 GSASIImpsubs Classes & Routines

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

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

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

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

GSASII.GSASIImpsubs.**ComputePwdrProfCW**(*profList*)

Compute the peaks profile for a set of CW peaks and add into the yc array

GSASII.GSASIImpsubs.**ComputePwdrProfCWA**(*profList*)

Compute the peaks profile for a set of TOF peaks and add into the yc array

GSASII.GSASIImpsubs.**ComputePwdrProfCWB**(*profList*)

Compute the peaks profile for a set of TOF peaks and add into the yc array

GSASII.GSASIImpsubs.**ComputePwdrProfED**(*profList*)

Compute the peaks profile for a set of TOF peaks and add into the yc array

GSASII.GSASIImpsubs.**ComputePwdrProfTOF**(*profList*)

Compute the peaks profile for a set of TOF peaks and add into the yc array

GSASII.GSASIImpsubs.**InitFobsSqGlobals**(*x1*, *ratio1*, *shl1*, *xB1*, *xF1*, *im1*, *lamRatio1*, *kRatio1*, *xMask1*, *Ka21*)

Initialize for the computation of Fobs Squared for powder histograms. Puts lots of junk into the global namespace in this module.

GSASII.GSASIImpsubs.**InitMP**(*allowMP=True*)

Called to initialize use of Multiprocessing

GSASII.GSASIImpsubs.**InitPwdrProfGlobals**(*im1*, *x1*)

Initialize for the computation of Fobs Squared for powder histograms. Puts lots of junk into the global namespace in this module.

GSASII.GSASIImpsubs.**ResetMP**()

Call after changing Config var 'Multiprocessing_cores' to force a resetting of the useMP from the parameter.

6.8 *Module nistlat: NIST*LATTICE cell computations*

6.8.1 nistlat Classes & Routines

This implements an interface to the NIST*LATTICE code using the Spring 1991 program version. NIST*LATTICE, "A Program to Analyze Lattice Relationships" was created by Vicky Lynn Karen and Alan D. Mighell (National Institute of Standards and Technology, Materials Science and Engineering Laboratory, Gaithersburg, Maryland 20899.) Minor code modifications made to provide more significant digits for cell reduction matrix terms.

Please cite V. L. Karen and A. D. Mighell, NIST Technical Note 1290 (1991), https://nvlpubs.nist.gov/nistpubs/Legacy/TN/nbstechnicalnote1290.pdf; and V. L. Karen & A. D. Mighell, U.S. Patent 5,235,523, https://patents.google.com/patent/US5235523A/en?oq=5235523 if this module is used.

GSASII.nistlat.**CellSymSearch**(*cellin*, *center*, *tolerance=[0.2, 0.2, 0.2, 1, 1, 1]*, *mode=0*, *deltaV=2*, *output=None*)

Search for a higher symmetry lattice related to an input unit cell, and optionally to the supercells and/or subcells with a specified volume ratio to the input cell.

Parameters

- **cellin** (*list*) – six lattice constants as float values
- **center** (*str*) – cell centering code; one of P/A/B/C/F/I/R Note that 'R' is used for rhombohedral lattices in either hexagonal or rhombohedral (primitive) cells
- **tolerance** (*list*) – comparison tolerances for a, b, c, alpha, beta & gamma (defaults to [0.2,0.2,0.2,1.,1.,1.]
- **mode** (*int*) –
  - 0: use only input cell,
  - 1: generate supercells,

  - 2: generate subcells
  - 3: generate sub- and supercells
- **deltaV** (*int*) – volume ratios for sub/supercells if mode != 0 as ratio of original cell to smallest subcell or largest supercell to original cell. Ignored if mode=0. Otherwise should be 2, 3, 4 or 5
- **output** (*str*) – name of file to write the NIST*LATTICE output. Default is None, which does not produce a file.

Returns

a list of processed cells (only one entry in list when mode=0) where for each cell the the following items are included:

- conventional input cell;
- reduced input cell;
- symmetry-generated conventional cell;
- symmetry-generated reduced cell;
- matrix to convert sym-generated output cell to input conventional cell

GSASII.nistlat.**CompareCell**(*cell1*, *center1*, *cell2*, *center2*, *tolerance=[0.2, 0.2, 0.2, 1, 1, 1]*, *mode='I'*, *vrange=8*, *output=None*)

Search for matrices that relate two unit cells

Parameters

- **cell1** (*list*) – six lattice constants as float values for 1st cell
- **center1** (*str*) – cell centering code for 1st cell; one of P/A/B/C/F/I/R Note that 'R' is used for rhombohedral lattices in either hexagonal or rhombohedral (primitive) cells
- **cell2** (*list*) – six lattice constants as float values for 2nd cell
- **center2** (*str*) – cell centering code for 2nd cell (see center1)
- **tolerance** (*list*) – comparison tolerances for a, b, c, alpha, beta & gamma (defaults to [0.2,0.2,0.2,1.,1.,1.]
- **mode** (*str*) – search mode, which should be either 'I' or 'F' 'I' provides searching with integral matrices or 'F' provides searching with integral and fractional matrices
- **vrange** (*int*) – maximum matrix term range. Must be 1 <= vrange <= 10 for mode='F' or Must be 1 <= vrange <= 40 for mode='I'
- **output** (*str*) – name of file to write the NIST*LATTICE output. Default is None, which does not produce a file.

Returns

A list of matrices that match cell1 to cell2 where each entry contains (det, im, m, tol, one2two, two2one) where:

- det is the determinant, giving the volume ratio between cells
- im relates the reduced cell for cell1 to the reduced cell for cell2
- m relates the reduced cell for cell2 to the reduced cell for cell1
- **tol shows the quality of agreement, as six differences between the** two reduced cells

- one2two: a numpy matrix that transforms cell1 to cell2
- two2one: a numpy matrix that transforms cell2 to cell1

GSASII.nistlat.**ConvCell**(*redcell*)

Converts a reduced cell to a conventional cell

Parameters

redcell (*list*) – unit cell parameters as 3 cell lengths and 3 angles (in degrees)

Returns

tuple (cell,center,setting,mat) where:

- cell: has the six cell dimensions for the conventional cell;
- center: is P/A/B/C/F/I/R;
- setting: is ' ' except for rhombohedral symmetry (center=R), where it will always be H (for hexagonal cell choice);
- mat: is the matrix that gives the conventional cell when the reduced cell is multiplied by mat.

GSASII.nistlat.**ReduceCell**(*center*, *cellin*, *mode=0*, *deltaV=0*, *output=None*)

Compute reduced cell(s) with NIST*LATTICE

Parameters

- **center** (*str*) – cell centering code; one of P/A/B/C/F/I/R Note that 'R' is used for rhombohedral lattices in either hexagonal or rhombohedral (primitive) cells
- **cellin** (*list*) – six lattice constants as float values
- **mode** (*int*) –
  - 0: reduction,
  - 1: generate supercells,
  - 2: generate subcells
  - 3: generate sub- and supercells
- **deltaV** (*int*) – volume ratios for sub/supercells if mode != 0 as ratio of original cell to smallest subcell or largest supercell to original cell. Ignored if mode=0. Otherwise should be 2, 3, 4 or 5
- **output** (*str*) – name of file to write the NIST*LATTICE output. Default is None, which does not produce a file.

Returns

a dict with two items, 'input' and 'output'. The value for 'input' is the input cell as (cell,center,setting). The value for 'output' is a list of reduced cells of form (d,cell,vol,mat,center,setting). In these:

- cell: a list with the six cell dimensions;
- center: is as above (always 'P' on output);
- setting: is ' ' except for rhombohedral symmetry where it may be R or H for the cell type;
- d: is the volume ratio for new cell over input cell;
- vol: is volume of output cell
- mat: is the matrix that gives the output cell when the input cell is multiplied by mat.

GSASII.nistlat.**showCell**(*cell*, *center='P'*, *setting=' '*, **ignored*)

show unit cell input or output nicely formatted.

Parameters

- **cell** (*list*) – six lattice constants as float values; a 7th volume value is ignored if present.
- **center** (*str*) – cell centering code; one of P/A/B/C/F/I/R Note that 'R' is used for rhombohedral lattices in either rhombohedral (primitive) or hexagonal cells.
- **setting** (*str*) – is ' ' except for rhombohedral symmetry where it will be R or H for the cell type.

Returns

a formatted string

GSASII.nistlat.**uniqCells**(*cellList*)

remove duplicated cells from a cell output list from *ReduceCell()*

Parameters

cellList (*list*) – A list of reduced cells where each entry represents a reduced cell as (_,cell,_,_,center,...) where cell has six lattice constants and center is the cell centering code (P/A/B/C/F/I/R).

Returns

a list as above, but where each unique cell is listed only once

6.9 *ReadMarCCDFrame: Read Mar Files*

class GSASII.ReadMarCCDFrame.**marFrame**(*File*, *byteOrd='<'*, *IFD={}*)

A class to extract correct mar header and image info from a MarCCD file

Parameters

- **File** (*str*) – file object [from open()]
- **byteOrd** – '<' (default) or '>'
- **IFD** (*dict*) – ?

6.10 *G2shapes: Compute SAS particle shapes*

Program SHAPES from "A New Algroithm for the Reconstruction of Protein Molecular Envelopes from X-ray Solution Scattering Data", John Badger, J. Appl. Cryst. (2019) 52, 937-944. (DOI: 10.1107/S1600576719009774) modified to run inside GSAS-II.

6.11 *tutorialIndex: index to GSAS-II tutorials*

The file *tutorialIndex.py* provides information on what tutorials are currently available in GSAS-II. The information on this is placed in this file and all indices to tutorials are generated from this.

6.11.1 tutorialIndex Contents

A catalog of GSAS-II tutorials with headings. This is the master list of GSAS-II tutorials and must be updated when tutorials are added. Each item has either one or three items. Titles are single item in a list or tuple. Tutorials have four items:

(a) the name of the directory,

(b) the name of the web page,

(c) a title for the tutorial and

(d) a short text description (optional).

Tutorials that depend on a previous tutorial being completed should have the title for the tutorial indented by five spaces.

Note that `GSASIIpath.tutorialCatalog` is generated from this table using the *makeGitTutorial.py* script (see https://gsas-ii.readthedocs.io/en/latest/GSASIIscripts.html#other-scripts) in the GSAS-II tutorials repo (https://github.com/AdvancedPhotonSource/GSAS-II-tutorials) and which creates the tutorials.html file in that repo.

6.12 *k_vector_search: k-vector utilities*

This contains routines used to perform k-vector (magnetic scattering lattice expansion) determination.

6.12.1 k_vector_search Contents

class GSASII.k_vector_search.**kVector**(*bravfSym: str*, *cell: list*, *positions: list*, *numbers: list*, *nucPeaks: list*, *superPeaks: list*, *threshold: float*, *option: int = 0*, *kstep: list = None*, *processes: int = 1*)

For k-vector search, given the input structure, the nucleus and satellite diffraction peaks.

Parameters

- **bravfSym** – Bravais lattice symbol
- **cell** – a *3x3* list of floats (cell[0] is the first lattice vector, etc.) The cell vectors should be corresponding to the primitive following the ITA convention.
- **positions** – a *Nx3* list of floats with the atomic coordinates in fractional coordinates (i.e., w.r.t. the cell vectors)
- **numbers** – a length-*N* list with integers identifying uniquely the atoms in the cell
- **nucPeaks** – a *mx4* list for holding the nucleus diffraction peaks (nucPeaks[0] has 4 entries, giving the hkl indeces corresponding to the conventional unit cell setting (ITA), together with the d-spacing)
- **superPeaks** – a length-*n* list for the satellite peak positions in *d*.
- **threshold** – specify the delta_d/d threshold. When the distance between two peaks is smaller than such a threshold, they would then be considered as identical. When searching for the single k vector, if the maximum value among those distances between the nominal and observed positions of the satellite peaks is smaller than the threshold, the corresponding k vector will be returned directly as the optimal solution.
- **option** – (optional) control the scope for k vector search in the Brillouin zone, *0* for high symmetry points only, *1* for high symmetry and edges, *2* for the whole Brillouin zone. Default: 0
- **kstep** – (optional) step of k values for searching over the k grid. Default: [.01, .01, .01]
- **processes** – (optional) the number of processes for parallel processing. Default: 1

hklConvToPrim(*hkl: list*) → ndarray

Convert the hkl indeces in the conventional cell setting to the primitive cell setting.

Parameters

hkl – input hkl indeces in the conventional cell setting

Returns
a list containing the hkl indeces in the primitive cell setting.

insIntoSortedList(*lst: list*, *new_val: float*) → tuple

Insert a new entry into the ascendingly sorted list.

Parameters

- **lst** – an ascendingly sorted list
- **new_val** – the new entry to be inserted into the sorted list

Returns
tuple containing the new sorted list and the index of the new entry in the new sorted list.

kOptFinder() → list

This is the kernel of the class, defining the method for searching over the Brillouin zone for optimal k vector that best explains the observed satellite peaks, given the already refined nucleus structure.

Returns
the list of top candidates of the optimal k vector (in the reciprocal *primitive cell* setting). If the top one is below the threshold, i.e., the distance between the nominal and observed positions of those satellite peaks smaller than the uncertainty of peak positions (which is determined by the instrument resolution), only one candidate will be returned. Otherwise, top 10 candidates will be returned.

kVecPrimToConv(*k_vec: list*) → ndarray

Convert the k vector in the reciprocal primitive lattice setting to that in the conventional cell setting.

Parameters
k_vec – the k vector in the reciprocal primitive lattice setting

Returns
the k vector in the conventional cell setting

kpathFinder() → dict

Provided the structure inputs, the routine will be collecting the inputs into a structure tuple which will be fed into the *get_path* routine in the *seekpath* module. The special k points and the k-path will be returned.

Returns
the dictionary containing the k-points, k-path and the reciprocal space primitive lattice vectors.

pointOnVector(*s_point: list*, *e_point: list*, *distance: float*) → list

Grab the coordinate of a point on a vector specified by the starting and ending points. The distance from the point on the vector to the starting point should be given as the parameter.

Parameters

- **s_point** – a list for the coordinate of the starting point
- **e_point** – a list for the coordinate of the ending point
- **distance** – the distance away from the starting point

Returns
the coordinate of the point on the vector

unique_closest(*list1: list*, *list2: list*) → tuple

Assign each member in *list2* with a unique member in *list1*. The assignment should guarantee that the sum of the squared difference between the mapping pairs is minimized. Here, the codes were created by the GPT-4-Turbo model, and the underlying algorithm used is the Hungarian algorithm. Here follows are some useful links,

https://medium.com/math-simplified/the-perfect-matching-1be8b028183c https://brilliant.org/wiki/hungarian-matching/

Parameters

- **list1** – the pool of satellite peaks to be generated by a certain k vector, given the nucleus peaks.
- **list2** – the list of observed satellite peaks.

Returns

a tuple containing two entries, the first being the mapping, and the second being the sum of the squared difference between the mapping pairs.

updateCandidateList(*kpoint: list*, *k_opt_list: list*, *k_opt_dist: list*, *ave_dd: list*, *max_dd: list*, *try_neg: bool*) → tuple

For a given k point, we want to find out such a one-to-one matching between the calculated satellite peaks and those observed ones that gives the minimal overall distance. The optimized overall distance will be treated as the *indicator distance* corresponding to the specified k-vector. If the *indicator distance* happens to be smaller than the uncertainty of peak positions (which is determined by the instrument resolution), only the top candidate will be returned.

Parameters

- **kpoint** – the trial k vector
- **k_opt_list** – list of top candidates of k vectors
- **k_opt_dist** – list of the *indicator distances* of those top candidates of k vectors
- **ave_dd** – list of ave. delta d values of those top candidates of k vectors
- **max_dd** – list of max. delta d values of those top candidates of k vectors
- **try_neg** – flag for controlling whether to try the negative k vector corresponding to the input k vector trial. For the corners of the irreducible Brillouin zone wedge, we don't need to try their negatives, as the k vector, in that case, is different from its negative by a lattice vector in reciprocal space and thus they are equivalent to each other.

Returns

tuple of the updated list of top candidates of k vectors and the updated list of the *indicator distances*

GSASII.k_vector_search.**lat_params_to_vec**(*lat_params: list*) → list

Construct lattice vectors from lattice parameters, according to the convention as detailed in the following post,

https://iris2020.net/2024-03-04-latt_params_to_latt_vecs/

Parameters

lat_params – list of lattice parameters a, b, c, alpha, beta and gamma. Angles should be given in degree.

Returns

lattice vectors in the list form, namely, the a, b and c lattice vectors given in the Cartesian coordinate

GSASII.k_vector_search.**unique_id_gen**(*string_list: list*) → list

Generate unique IDs for strings included in the string list and the same string will be assigned with the same ID.

Parameters

string_list – list of strings

Returns

list of integer IDs

6.13 *pathHacking utilities*

When GSAS-II is installed via Git (gsas2main for example, as opposed to pixi), the location of GSAS-II is "outside" the Python file structure. It then becomes necessary to set the Python *sys.path* variable to find GSAS-II files. This practice is discouraged and commands to do this are "exiled" to file *pathHacking.py*.

CHAPTER

SEVEN

GSAS-II GUI SUPPORT MODULES

The modules documented here provide GUI or graphics capabilities that are used in multiple sections of the GSAS-II GUI or graphics.

7.1 GSASIIctrlGUI: Custom GUI controls

A library of specialized widgets (GUI controls) for use in the GSAS-II data window or dialogs:

Class or function name	Description
EnumSelector	A combo box with a built-in call back routine that automatically sets a dict or list entry.
G2ChoiceButton	A customized wx.Choice that automatically initializes to the initial value and saves the choice directly into a dict or list value. Optionally calls function when a choice is selected
G2CheckBox	A customized wx.CheckBox that automatically initializes to the initial value and saves the choice directly into a dict or list value. Optionally calls function when a choice is selected
G2CheckBoxFrontLbl()	A version of *G2CheckBox* that places the label for the check box in front. Otherwise works the same.
G2RadioButtons()	Creates a series of grouped radio buttons.
G2SliderWidget	A customized combination of a wx.Slider and a validated wx.TextCtrl (see *ValidatedTxtCtrl*).
G2Slider	A wrapped version of wx.Slider that implements scaling
G2SpinWidget	A customized combination of a wx.SpinButton and a validated wx.TextCtrl (see *ValidatedTxtCtrl*).
G2MultiChoiceWindow	Similar to *G2MultiChoiceDialog* but provides a sizer that can be placed in a frame or panel.
HelpButton	Creates a button labeled with a "?" that when pressed displays help text in a modal message window or web browser.
OrderBox	Creates a wx.Panel with scrollbars where items can be ordered into columns.
SortableLstCtrl	Creates a wx.Panel for a table of data that can be sorted by clicking on a column label.
ValidatedTxtCtrl	A text control with a built-in call back routine to set dict or list elements. Optionally validates input as float, int or for strings non-blank. Value is set when focus changes
HorizontalLine()	Places a line in a Frame or Dialog to separate sections.
ScrolledStaticText	A wx.StaticText widget that fits a large string into a small space by scrolling it
ReadOnlyTextCtrl()	A wx.TextCtrl widget to be used wx.StaticText (no edits allowed) text appears in a box.
setColorButton()	A button for color selection as a replacement for wx.ColourSelect

GSAS-II-provided Dialog (full window) routines:

Class or function name	Description
G2MessageBox()	Displays text typically used for errors or warnings.
ShowScrolledInfo()	Dialog to display longer text where scrolling is possibly needed
gpxFileSelector	File browser dialog for opening existing .gpx files
GetImportFile()	Gets one or more files from the appropriate import directory, which can be overridden. Arguments follow those of `wx.FileDialog()`
G2FileBrowser()	An alternate file selection dialog to use in place of `wx.FileDialog()`. Offers a filter and breaks long lists into ranges.
askSaveFile()	Get a file name from user
askSaveDirectory()	Get a directory name from user
MultiColumnSelection	A dialog that builds a multicolumn table, word wrapping is used for the 2nd, 3rd,… columns.
MultiColMultiSelDlg()	Calls a dialog that allows multiple selection from a multicolumn table

continues on next page

Table 1 – continued from previous page

Class or function name	Description
MultiDataDialog	Dialog to obtain multiple data values from user, with optional range validation; items can be float, str or bool
MultiIntegerDialog	Dialog to obtain multiple integer values from user, with a description for each value and optional defaults.
MultiStringDialog	Dialog to obtain multiple string values from user, with a description for each value and optional
DisAglDialog	Distance/Angle Controls input dialog.
FlagSetDialog	Dialog that provides a table of items along with a checkbox for each. defaults.
G2ColumnIDDialog	A dialog for matching column data to desired items; some columns may be ignored.
G2HistoDataDialog	A dialog for global edits to histogram data globally
G2MultiChoiceDialog	Dialog similar to wx.MultiChoiceDialog, but provides a filter to select choices and buttons to make selection of multiple items more simple.
G2SingleChoiceDialog	Dialog similar to wx.SingleChoiceDialog, but provides a filter to help search through choices.
SGMagSpinBox	Special version of wx.MessageBox that displays magnetic spin text
SGMessageBox	Special version of wx.MessageBox that displays space group & super space group text in two blocks
SingleFloatDialog	Dialog to obtain a single float value from user, with optional range validation.
SingleIntDialog	Dialog to obtain a single integer value from user, with optional range validation.
SingleStringDialog	Dialog to obtain a single string value from user, with optional an optional default value.
GetItemOrder()	Dialog for ordering items into columns
ScrolledMultiEditor	Dialog for editing many dict- or list-contained items. with validation. Results are placed in dict or list.
CallScrolledMultiEditor()	Routine for editing many dict- or list-contained items. using the *ScrolledMultiEditor* dialog
G2ScrolledGrid()	Displays a multicolumn table of information with possible scroll bars
ShowScrolledColText()	Displays tabular text with scrolling where needed
ItemSelector()	Select a single item or multiple items from list of choices. Creates and then destroys a wx.Dialog and returns the selections(s).
SelectEdit1Var()	Select a variable from a list, then edit it and select histograms to copy it to.
BlockSelector()	Select a single block for instrument parameters
MultipleBlockSelector()	Select one or more blocks of data, used for CIF powder histogram imports only
MultipleChoicesSelector()	Dialog for displaying fairly complex choices, used for CIF powder histogram imports only
PhaseSelector()	Select a phase from a list (used for phase importers)

Miscellaneous GUI support routines:

Function name	Description
Define_wxId()	Create a unique wx.Id symbol in _initMenus in *GSASII.GSASIIdataGUI*. Such symbols are needed when the menu item is defined in a different location from the wx.Bind that links the menu item to a function. This function allows all the menu Ids to be defined as the menus are created in one place and then can be used in Bind elsewhere in the code.
`openInNewTerm()`	opens a Python routine (usually `G2.py`) in a new terminal window (works on all platforms)

Other miscellaneous non-GUI routines that may be of use for GUI-related actions:

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

7.1.1 GSASIIctrlGUI Classes & Routines

Documentation for all the routines in module `GSASIIctrlGUI` follows.

class GSASII.GSASIIctrlGUI.**ASCIIValidator**(*result=None*, *key=None*)

A validator to be used with a TextCtrl to prevent entering characters other than ASCII characters.

The value is checked for validity after every keystroke

If an invalid number is entered, the box is highlighted. If the number is valid, it is saved in result[key]

Parameters

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

Clone()

Create a copy of the validator, a strange, but required component

OnChar(*event*)

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

TestValid(*tc*)

Check if the value is valid by casting the input string into ASCII.

Save it in the dict/list where the initial value was stored

Parameters

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

TransferFromWindow()

Needed by validator, strange, but required component

TransferToWindow()

Needed by validator, strange, but required component

GSASII.GSASIIctrlGUI.**BlockSelector**(*ChoiceList*, *ParentFrame=None*, *title='Select a block'*, *size=None*, *header='Block Selector'*, *useCancel=True*)

Provide a wx dialog to select a single block where one must be selected. Used for selecting for banks for instrument parameters if the file contains more than one set.

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

Wrapper routine to call a ScrolledMultiEditor dialog. See *ScrolledMultiEditor* for parameter definitions.

Returns

True if the OK button is pressed; False if the window is closed with the system menu or the Cancel button.

GSASII.GSASIIctrlGUI.**Define_wxId**(**args*)

routine to create unique global wx Id symbols in this module.

class GSASII.GSASIIctrlGUI.**DisAglDialog**(*parent*, *data*, *default*, *Reset=True*, *Angle=True*)

Distance/Angle Controls input dialog. After `ShowModal()` returns, the results are found in dict `self.data`, which is accessed using *GetData()*.

Parameters

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

Draw(*data*)

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

GetData()

Returns the values from the dialog

OnOk(*event*)

Called when the OK button is pressed

OnReset(*event*)

Called when the Reset button is pressed

class GSASII.GSASIIctrlGUI.**EnumSelector**(*parent*, *dct*, *item*, *choices*, *values=None*, *OnChange=None*, ***kw*)

A customized `wxpython.ComboBox` that selects items from a list of choices, but sets a dict (list) entry to the corresponding entry from the input list of values.

Parameters

- **parent** (*wx.Panel*) – the parent to the `ComboBox` (usually a frame or panel)
- **dct** – a dict or list to contain the value set for the `ComboBox`.

- **item** – the dict key (or list index) where `dct[item]` will be set to the value selected in the `ComboBox`. Also, dct[item] contains the starting value shown in the widget. If the value does not match an entry in `values`, the first value in `choices` is used as the default, but `dct[item]` is not changed.
- **choices** (*list*) – a list of choices to be displayed to the user such as

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

  Note that these options will correspond to the entries in `values` (if specified) item by item.
- **values** (*list*) – a list of values that correspond to the options in `choices`, such as

```
[0,1,2]
```

  The default for `values` is to use the same list as specified for `choices`.
- **OnChange** (*function*) – an optional routine that will be called when the `ComboBox` can be specified.
- **(other)** – additional keyword arguments accepted by `ComboBox` can be specified.

GSASII.GSASIIctrlGUI.**ExtractFileFromZip**(*filename*, *selection=None*, *confirmread=True*, *confirmoverwrite=True*, *parent=None*, *multipleselect=False*, *msg=''*)

If the filename is a zip file, extract a file from that archive.

Parameters

- **selection** (*list*) – used to predefine the name of the file to be extracted. Filename case and zip directory name are ignored in selection; the first matching file is used.
- **confirmread** (*bool*) – if True asks the user to confirm before expanding the only file in a zip
- **confirmoverwrite** (*bool*) – if True asks the user to confirm before overwriting if the extracted file already exists
- **multipleselect** (*bool*) – if True allows more than one zip file to be extracted, a list of file(s) is returned. If only one file is present, do not ask which one, otherwise offer a list of choices (unless selection is used).
- **msg** (*str*) – a message explaining what is being read. Default is blank.

Returns

the name of the file that has been created or a list of files (see multipleselect)

If the file is not a zipfile, return the name of the input file. If the zipfile is empty or no file has been selected, return None

class GSASII.GSASIIctrlGUI.**FlagSetDialog**(*parent*, *title*, *colnames*, *rownames*, *flags*)

Creates popup with table of variables to be checked for e.g. refinement flags

GSASII.GSASIIctrlGUI.**G2AfterFit**(*parent*, *msg*, *title='Error'*, *vartbl=[]*, *txtwidth=300*)

Shows the results from a refinement

Parameters

- **parent** (*wx.Frame*) – pointer to parent of window, usually G2frame
- **msg** (*str*) – text from refinement results
- **title** (*str*) – text to label window

- **vartbl** (*list*) – a list of lists. The contents of each inner list item will be [var-name, val-before, val-after, sigma, meaning]
- **txtwidth** (*int*) – width (in pixesl) to display msg. Defaults to 300

class GSASII.GSASIIctrlGUI.**G2Button**(*parent*, *id=-1*, *label=''*, *locationcode=''*, *handler=None*, **args*, ***kwargs*)

A version of wx.Button. Bindings are saved in the object, and are looked up rather than directly set with a bind.

Parameters

- **parent** (*wx.Panel*) – parent widget
- **id** (*int*) – Id for button
- **label** (*str*) – label for button
- **handler** (*function*) – a routine to call when the button is pressed

onPress(*event*)

create log event and call handler

class GSASII.GSASIIctrlGUI.**G2CheckBox**(*parent*, *label*, *loc*, *key*, *OnChange=None*)

A customized version of a CheckBox that automatically initializes the control to a supplied list or dict entry and updates that entry as the widget is used.

Parameters

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

GSASII.GSASIIctrlGUI.**G2CheckBoxFrontLbl**(*parent*, *label*, *loc*, *key*, *OnChange=None*)

A customized version of a CheckBox that automatically initializes the control to a supplied list or dict entry and updates that entry as the widget is used. Same as *G2CheckBox* except the label is placed before the CheckBox and returns a sizer rather than the G2CheckBox.

If the CheckBox is needed, reference Sizer.myCheckBox.

class GSASII.GSASIIctrlGUI.**G2ChoiceButton**(*parent*, *choiceList*, *indLoc=None*, *indKey=None*, *strLoc=None*, *strKey=None*, *onChoice=None*, ***kwargs*)

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

Parameters

- **parent** (*wx.Panel*) – name of panel or frame that will be the parent to the widget. Can be None.

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

Example 1:

```
data['Orientation'] = 1
choice = G2G.G2ChoiceButton(dlg,
                    ['Horizontal','Vertical'],
                    data,'Orientation',
                    onChoice=replot)
```

This will show "Vertical" as the initial value, and based on what is selected, `data['Orientation']` will be set to 0 or 1 (as an int value).

Example 2:

```
data[0]['Font'] = '8'
choice = G2G.G2ChoiceButton(dlg,
                    ['6','8','10','12','14','16'],
                    None,None,
                    data[0],'Font',
                    onChoice=replot)
```

This will show "8" as the initial value, and based on what is selected, `data[0]['Font']` will be set to a string with one of the size options ("6"… "16").

setByString(*string*)

Find an entry matching string and select it

class GSASII.GSASIIctrlGUI.**G2ColumnIDDialog**(*parent*, *title*, *header*, *Comments*, *ChoiceList*, *ColumnData*, *monoFont=False*, ***kw*)

A dialog for matching column data to desired items; some columns may be ignored.

Parameters

- **ParentFrame** (*wx.Frame*) – reference to parent frame
- **title** (*str*) – heading above list of choices
- **header** (*str*) – Title to place on window frame
- **ChoiceList** (*list*) – a list of possible choices for the columns
- **ColumnData** (*list*) – lists of column data to be matched with ChoiceList

- **monoFont** (*bool*) – If False (default), use a variable-spaced font; if True use a equally-spaced font.
- **kw** – optional keyword parameters for the wx.Dialog may be included such as size [which defaults to *(320,310)*] and style (which defaults to `wx.DEFAULT_DIALOG_STYLE | wx.RESIZE_BORDER | wx.CENTRE | wx.OK | wx.CANCEL`); note that `wx.OK` and `wx.CANCEL` controls the presence of the eponymous buttons in the dialog.

Returns
the name of the created dialog

GetSelection()

Returns the selected sample parm for each column

GSASII.GSASIIctrlGUI.**G2FileBrowser**(*parent*, *message*, *defaultDir*, **args*, *style=1*, *wildcard='any file (*.*)|*.*'*, ***kwargs*)

Create a file browser for selecting one or more files from a directory. The user may select the directory where the files and must select an extension from a specified list. Optionally, a filter may be supplied to ignore files not matching a glob pattern. When a large number of files is found, the file list is broken into ranges.

Arguments used are the same as in `wx.FileDialog()`. Selection of multiple files is allowed if argument style includes wx.FD_MULTIPLE. Note that other positional or keyword parameters, other than those below are allowed, but are ignored.

Parameters

- **parent** – parent window to dialog to be created
- **message** – message to place at top of dialog
- **defaultDir** – starting directory
- **style** – selection style. Bit settings other than wx.FD_MULTIPLE are ignored. When wx.FD_MULTIPLE is specified, multiple files can be selected.
- **wildcard** – defines the extensions allowed for selecting files. of form "type A (*.A;*.B;*.c)|*.A;*.B;*.c|any file (.)|*.*". See `wx.FileDialog()` for more information on this.

Returns
a list of selected files. Will contain only one file unless style=wx.FD_MULTIPLE has been used.

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

A dialog for editing histogram data globally.

Parameters

- **ParentFrame** (*wx.Frame*) – reference to parent frame
- **title** (*str*) – heading above list of choices
- **header** (*str*) – Title to place on window frame
- **ParmList** (*list*) – a list of names for the columns
- **ParmFmt** (*list*) – a list of formatting strings for the columns
- **list** – HistoList: a list of histogram names
- **ParmData** (*list*) – a list of lists of data matched to ParmList; one for each item in HistoList
- **monoFont** (*bool*) – If False (default), use a variable-spaced font; if True use a equally-spaced font.

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

Returns
the modified ParmData

GetData()

Returns the modified ParmData

class GSASII.GSASIIctrlGUI.**G2LstCtrl**(*parent*, *ID=-1*, *pos=wx.Point(-1, -1)*, *size=wx.Size(-1, -1)*, *style=0*)

Creates a custom ListCtrl with support for images in column labels

GetColumnSorter()

Custom sorter that handles absolute numerical values for columns 1, 2, and 3 (2nd, 3rd, 4th columns)

GetSortImages()

Returns a tuple of image list indexesthe indexes in the image list for an image to be put on the column header when sorting in descending order.

GSASII.GSASIIctrlGUI.**G2MessageBox**(*parent*, *msg*, *title='Error'*)

Simple code to display a error or warning message

TODO: replace wx.MessageDialog with one derived from wx.Dialog because on most platforms wx.MessageDialog is a native widget and CentreOnParent will not function.

class GSASII.GSASIIctrlGUI.**G2MultiChoiceDialog**(*parent*, *title*, *header*, *ChoiceList*, *toggle=True*, *monoFont=False*, *filterBox=True*, *extraOpts={}*, *selected=[]*, ***kw*)

A dialog similar to wx.MultiChoiceDialog except that buttons are added to set all choices and to toggle all choices and a filter is available to select from available entries. Note that if multiple entries are placed in the filter box separated by spaces, all of the strings must be present for an item to be shown.

Parameters

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

note that *wx.OK* and *wx.CANCEL* style items control the presence of the eponymous buttons in the dialog.

Returns
the name of the created dialog

Filter(*event*)

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

GetSelections()

Returns a list of the indices for the selected choices

OnCheck(*event*)

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

SetRange(*event*)

Respond to a press of the Set Range button. Set the range flag and the caption next to the button

SetSelections(*selList*)

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

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

onChar(*event*)

Respond to keyboard events in the Filter box

class GSASII.GSASIIctrlGUI.**G2MultiChoiceWindow**(*parent*, *title*, *ChoiceList*, *SelectList*, *toggle=True*, *monoFont=False*, *filterBox=True*, *OnChange=None*, *OnChangeArgs=[]*, *helpText=None*)

Creates a sizer similar to G2MultiChoiceDialog except that buttons are added to set all choices and to toggle all choices. This is placed in a sizer, so that it can be used in a frame or panel.

Parameters

- **parent** – reference to parent frame/panel
- **title** (*str*) – heading above list of choices
- **ChoiceList** (*list*) – a list of choices where one more will be selected
- **SelectList** (*list*) – a list of selected choices
- **toggle** (*bool*) – If True (default) the toggle and select all buttons are displayed
- **monoFont** (*bool*) – If False (default), use a variable-spaced font; if True use a equally-spaced font.
- **filterBox** (*bool*) – If True (default) an input widget is placed on the window and only entries matching the entered text are shown.
- **OnChange** (*function*) – a reference to a callable object, that is called each time any a choice is changed. Default is None which will not be called.
- **OnChangeArgs** (*list*) – a list of arguments to be supplied to function OnChange. The default is a null list.

Returns
the name of the created sizer

Filter(*event*)

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

GetSelections()

Returns a list of the indices for the selected choices

OnCheck(*event*)

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

SetRange(*event*)

Respond to a press of the Set Range button. Set the range flag and the caption next to the button

SetSelections(*selList*)

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

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

onChar(*event*)

Respond to keyboard events in the Filter box

GSASII.GSASIIctrlGUI.**G2RadioButtons**(*parent*, *loc*, *key*, *choices*, *values=None*, *OnChange=None*)

A customized version of wx.RadioButton that returns a list of coupled RadioButtons

Parameters

- **parent** (*wx.Panel*) – name of panel or frame that will be the parent to the widgets. Can be None.
- **loc** (*dict/list*) – the dict or list with the initial value to be placed in the CheckBox.
- **key** (*int/str*) – the dict key or the list index for the value to be edited by the CheckBox. The `loc[key]` element must exist. The CheckButton will be initialized from this value.
- **choices** (*list*)
- **values** (*list*)
- **OnChange** (*function*) – specifies a function or method that will be called when the CheckBox is changed (Default is None). The called function is supplied with one argument, the calling event.

class GSASII.GSASIIctrlGUI.**G2RefinementProgress**(*title='Refinement progress'*, *message='All data Rw ='*, *maximum=101*, *parent=None*, *trialMode=False*, *seqLen=0*, *seqShow=3*, *style=None*)

Defines an replacement for wx.ProgressDialog to be used for showing refinement progress.

Parameters

- **title** (*str*) – string to place on border of window (default is 'Refinement progress').
- **message** (*str*) – initial string to place on top line of window.
- **maximum** (*int*) – maximum value for progress gauge bar on bottom of window.

- **parent** (*wx.Frame*) – parent window for creation of this dialog
- **trialMode** (*bool*) – Set to True for Levenberg-Marquardt fitting where Rw may be computed several times for a single cycle. Call *AdvanceCycle()* when trialMode is True to indicate that a cycle has been completed. Default is False.
- **seqLen** (*int*) – Number of histograms in sequential fit. A value of zero (default) means that the fit is not a sequential fit.
- **seqShow** (*int*) – Number of histograms to shown in a sequential fit (default 3)
- **style** (*int*) – optional parameters that determine how the dialog is is displayed.

AdvanceCycle(*cycle=None*)

Call this directly with Levenberg-Marquardt fitting after a cycle completes. Plots the results.

Destroy()

Destroy the window, but allow events to clear before doing so

SetHistogram(*nextHist*, *histLbl*)

Set this before beginning processing of each histogram

SetMaxCycle(*value*)

Set the maximum number of cycles or histograms (sequential fit). Used to scale settings so the gauge bar completes close to 100%. Ignored for sequential refinements.

Update(*value=None*, *newmsg=''*)

designed to work with calls intended for wx.ProgressDialog.Update the value is assumed to be the current wR value for the histogram selected with SetHistogram and newmsg goes into the 2nd status line.

GSASII.GSASIIctrlGUI.**G2ScrolledGrid**(*G2frame*, *lbl*, *title*, *tbl*, *colLbls*, *colTypes*, *maxSize=(600, 300)*, *comment=''*)

Display a scrolled table of information in a dialog window

Parameters

- **G2frame** (*wx.Frame*) – parent for dialog
- **lbl** (*str*) – label for window
- **title** (*str*) – window title
- **tbl** (*list*) – list of lists where inner list is each row
- **colLbls** (*list*) – list of str with labels for each column
- **colTypes** (*list*) – Data types for each column (such as wg.GRID_VALUE_STRING,wg.GRID_VALUE_FLOAT)
- **maxSize** (*list*) – Maximum size for the table in points. Defaults to (600,300)
- **comment** (*str*) – optional text that appears below table

Example:

```
row = ['item1',1.234,'description of item']
colTypes = [wg.GRID_VALUE_STRING,wg.GRID_VALUE_FLOAT+':8,4',wg.GRID_VALUE_STRING]
colLbls = ['item name','value','Description']
G2ScrolledGrid(frm,'window label','title',20*[row],colLbls,colTypes)
```

class GSASII.GSASIIctrlGUI.**G2SingleChoiceDialog**(*parent*, *title*, *header*, *ChoiceList*, *monoFont=False*, *filterBox=True*, ***kw*)

A dialog similar to wx.SingleChoiceDialog except that a filter can be added.

Parameters

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

Returns

the name of the created dialog

Example:

```
dlg = G2SingleChoiceDialog(G2frame,'Select option from list',
                                'Select option',optList)
dlg.CenterOnParent()
try:
    if dlg.ShowModal() == wx.ID_OK:
        sel = optList[dlg.GetSelection()]
    else:
        return
finally:
    dlg.Destroy()
```

GetSelection()

Returns the index of the selected choice

class GSASII.GSASIIctrlGUI.**G2Slider**(*parent*, *id=-1*, *value=0*, *minValue=0*, *maxValue=100*, **arg*, ***kwarg*)

Wrapper around wx.Slider widget that implements scaling Also casts floats as integers to avoid py3.10+ errors

SetLineSize(*lineSize*) → None

Sets the line size for the slider.

SetMax(*maxValue*) → None

Sets the maximum slider value.

SetMin(*minValue*) → None

Sets the minimum slider value.

SetValue(*value*) → None

Sets the slider position.

GSASII.GSASIIctrlGUI.**G2SliderWidget**(*parent*, *loc*, *key*, *label*, *xmin*, *xmax*, *iscale*, *onChange=None*, *onChangeArgs=[]*, *sizer=None*, *nDig=None*, *size=(50, 20)*, *slsize=(200, 25)*)

A customized combination of a wx.Slider and a validated wx.TextCtrl (see *ValidatedTxtCtrl*) that allows either a slider or text entry to set a value within a range.

Parameters

- **parent** (*wx.Panel*) – name of panel or frame that will be the parent to the TextCtrl. Can be None.
- **loc** (*dict/list*) – the dict or list with the initial value to be placed in the TextCtrl.
- **key** (*int/str*) – the dict key or the list index for the value to be edited by the TextCtrl. The `loc[key]` element must exist and should have a float value. It will be forced to an initial value between xmin and xmax.
- **label** (*str*) – A label to be placed to the left of the slider.
- **xmin** (*float*) – the minimum allowed valid value.
- **xmax** (*float*) – the maximum allowed valid value.
- **iscale** (*float*) – number to scale values to integers, which is what the Scale widget uses. If the xmin=1 and xmax=4 and iscale=1 then values only the values 1,2,3 and 4 can be set with the slider. However, if iscale=2 then the values 1, 1.5, 2, 2.5, 3, 3.5 and 4 are all allowed.
- **onChange** (*callable*) – function to call when value is changed. Default is None where nothing will be called.
- **onChangeArgs** (*list*) – arguments to be passed to onChange function when called.
- **sizer** (*wx.Sizer*) – a reference to a wx.BoxSizer to be used to place the TextCtrl and Slider into. If None (the default), the sizer is created.
- **size** (*tuple*) – size of the TextCtrl [defaults to (50,20)]
- **slsize** (*tuple*) – slsize of the Slider [defaults to (200,25)]

Returns

returns a wx.BoxSizer containing the widgets

GSASII.GSASIIctrlGUI.**G2SpinWidget**(*parent*, *loc*, *key*, *label*, *xmin=None*, *xmax=None*, *nDig=None*, *onChange=None*, *typeHint=<class 'int'>*, *onChangeArgs=[]*, *size=(50, 25)*)

A customized combination of a wx.SpinButton and a validated wx.TextCtrl (see *ValidatedTxtCtrl*) that allows either a the spin button or text entry to set a value within a range.

Parameters

- **parent** (*wx.Panel*) – name of panel or frame that will be the parent to the TextCtrl. Can be None.
- **loc** (*dict/list*) – the dict or list with the initial value to be placed in the TextCtrl.
- **key** (*int/str*) – the dict key or the list index for the value to be edited by the TextCtrl. The `loc[key]` element must exist and should have a float or int value. It will be forced to an integer initial value between xmin and xmax.
- **label** (*str*) – A label to be placed to the left of the entry widget.
- **xmin** (*int*) – the minimum allowed valid value. If None it is ignored.
- **xmax** (*int*) – the maximum allowed valid value. If None it is ignored.

- **nDig** (*list*) – as defined for ValidatedTxtCtrl
- **onChange** (*callable*) – function to call when value is changed. Default is None where nothing will be called.
- **onChangeArgs** (*list*) – arguments to be passed to onChange function when called.
- **size** (*int*) – size of TextCtrl in pixels. Defaults to (50,25).

Returns
returns a wx.BoxSizer containing the widgets

class GSASII.GSASIIctrlGUI.**G2TreeCtrl**(*parent=None*, **args*, ***kwargs*)

Create a wrapper around the standard TreeCtrl so we can "wrap" various events.

AppendItem(*parent*, *text*, *image=-1*, *selImage=-1*, *data=None*)

override the standard method so font size can be set

ConvertRelativeHistNum(*histtype*, *histnum*)

Converts a histogram type and relative histogram number to a histogram name in the current project

ConvertRelativePhaseNum(*phasenum*)

Converts relative phase number to a phase name in the current project

GetImageLoc(*TreeId*)

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

GetItemPyData(*treeId*)

GetItemData(item) -> TreeItemData

Returns the tree item data associated with the item.

GetRelativeHistNum(*histname*)

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

GetRelativePhaseNum(*phasename*)

Returns a phase number if the string matches a phase name or else returns the original string

RestoreExposedItems()

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

SaveExposedItems()

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

SetItemPyData(*treeId*, *data*)

SetItemData(item, data) -> None

Sets the item client data.

UpdateImageLoc(*TreeId*, *imagefile*)

Saves a new imagefile name in the Tree. Handles cases where the image name is specified, as well as where the image file name is a tuple containing the image file and an image number

class GSASII.GSASIIctrlGUI.**GSGrid**(*parent*, *name=''*)

Basic wx.Grid implementation

InstallGridToolTip(*rowcolhintcallback*, *colLblCallback=None*, *rowLblCallback=None*)

code to display a tooltip for each item on a grid from http://wiki.wxpython.org/wxGrid%20ToolTips (buggy!), expanded to column and row labels using hints from https://groups.google.com/forum/#!topic/wxPython-users/bm8OARRVDCs

Parameters

- **rowcolhintcallback** (*function*) – a routine that returns a text string depending on the selected row and column, to be used in explaining grid entries.
- **colLblCallback** (*function*) – a routine that returns a text string depending on the selected column, to be used in explaining grid columns (if None, the default), column labels do not get a tooltip.
- **rowLblCallback** (*function*) – a routine that returns a text string depending on the selected row, to be used in explaining grid rows (if None, the default), row labels do not get a tooltip.

SetTable(*table*, **args*, ***kwargs*)

Overrides the standard SetTable method with one that uses GridFractionEditor for all numeric columns (unless useFracEdit is false)

completeEdits()

complete any outstanding edits

setupPopup(*lblList*, *callList*)

define a callback that creates a popup menu. The rows associated with the items selected items are selected in the table and if an item is called from the menu, the corresponding function is called to perform an action on the

Parameters

- **lblList** (*list*) – list of str items that will be placed in the popup menu
- **callList** (*list*) – list of functions to be called when a

Returns

a callback that can be used to create the menu

Sample usage:

```
lblList = ('Delete','Set atom style','Set atom label',
               'Set atom color','Set view point','Generate copy',
               'Generate surrounding sphere','Transform atoms',
               'Generate bonded')
callList = (DrawAtomsDelete,DrawAtomStyle, DrawAtomLabel,
                DrawAtomColor,SetViewPoint,AddSymEquiv,
                AddSphere,TransformSymEquiv,
                FillCoordSphere)
onRightClick = drawAtoms.setupPopup(lblList,callList)
drawAtoms.Bind(wg.EVT_GRID_CELL_RIGHT_CLICK, onRightClick)
drawAtoms.Bind(wg.EVT_GRID_LABEL_RIGHT_CLICK, onRightClick)
```

class GSASII.GSASIIctrlGUI.**GSNoteBook**(*parent*, *name=''*, *size=None*, *style=257*)

Notebook used in various locations; implemented with wx.aui extension

ChangeSelection(*n*) → int

Changes the selection for the given page, returning the previous selection.

FindPage(*page*) → int

Returns the index of the specified tab window or wxNOT_FOUND if not found.

GSASII.GSASIIctrlGUI.**GetCite**(*key*, *wrap=None*, *indent=None*)

Return citation information, optionally with text wrapping.

GSASII.GSASIIctrlGUI.**GetConfigValsDocs**()

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

Returns

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

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

GSASII.GSASIIctrlGUI.**GetExportPath**(*G2frame*)

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

Returns

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

GSASII.GSASIIctrlGUI.**GetImportFile**(*G2frame*, *message*, *defaultDir=''*, *defaultFile=''*, *style=1*, *parent=None*, **args*, ***kwargs*)

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

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

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

Returns

a list of files or an empty list

GSASII.GSASIIctrlGUI.**GetImportPath**(*G2frame*)

Determines the default location to use for importing files. Tries sequentially G2frame.TutorialImportDir, config var Import_directory, G2frame.LastImportDir and G2frame.LastGPXdir

Returns

a string containing the path to be used when reading files or '.' if none of the above are specified.

GSASII.GSASIIctrlGUI.**GetItemOrder**(*parent*, *keylist*, *vallookup*, *posdict*)

Creates a dialog where items can be ordered into columns

Parameters

- **keylist** (*list*) – is a list of keys for column assignments
- **vallookup** (*dict*) – is a dict keyed by names in keylist where each item is a dict. Each inner dict contains variable names as keys and their associated values

- **posdict** (*dict*) – is a dict keyed by names in keylist where each item is a dict. Each inner dict contains column numbers as keys and their associated variable name as a value. This is used for both input and output.

class GSASII.GSASIIctrlGUI.**GridFractionEditor**(*grid*)

A grid cell editor class that allows entry of values as fractions as well as sine and cosine values [as s() and c(), sin() or sind(), etc]. Any valid Python expression will be evaluated.

The current value can be incremented, multiplied or divided by prefixing an expression by +, * or / respectively.

ApplyEdit(*row*, *col*, *grid*)

Called only in wx >= 2.9 Save the value of the control into the grid if EndEdit() returns as True

BeginEdit(*row*, *col*, *grid*) → None

Fetch the value from the table and prepare the edit control to begin editing.

Clone() → GridCellEditor

Create a new object which is the copy of this one.

Create(*parent*, *id*, *evtHandler*) → None

Creates the actual edit control.

EndEdit(*row*, *col*, *grid*, *oldval*)

End editing the cell.

This function must check if the current value of the editing cell is valid and different from the original value in its string form. If not then simply return None. If it has changed then this method should save the new value so that ApplyEdit can apply it later and the string representation of the new value should be returned.

Notice that this method shoiuld not modify the grid as the change could still be vetoed.

Reset() → None

Reset the value in the control back to its starting value.

SetSize(*rect*) → None

Size and position the edit control.

StartingKey(*event*) → None

If the editor is enabled by pressing keys on the grid, this will be called to let the editor do something about that first key if desired.

class GSASII.GSASIIctrlGUI.**HelpButton**(*parent*, *msg=''*, *helpIndex=''*, *wrap=None*)

Create a help button that displays help information. The text can be displayed in a modal message window or it can be a reference to a location in the gsasII help web pages, in which case that page is opened in a web browser.

TODO: it might be nice if it were non-modal: e.g. it stays around until the parent is deleted or the user closes it, but this did not work for me.

Parameters

- **parent** – the panel/frame where the button will be placed
- **msg** (*str*) – the help text to be displayed. Indentation on multiline help text is stripped (see *StripIndents()*). If wrap is set as non-zero, all new lines are
- **helpIndex** (*str*) – selection for the help information in the GSAS-II help files, in the form of an anchor string. That anchor is looked up to find the file name and the URL is constructed from:

  <location> + <filename> + "#" + helpIndex

- **wrap** (*int*) – if specified, the text displayed is reformatted by wrapping it to fit in wrap pixels. Default is None which prevents wrapping.

GSASII.GSASIIctrlGUI.**HistogramNameTemplate**(*exporter*, *stripChars*)

Dialog to obtain a string value for grouping histograms

Parameters

- **exporter** (*obj*) – reference to exporter object
- **stripChars** (*str*) – Characters that are not allowed in file names

GSASII.GSASIIctrlGUI.**HorizontalLine**(*sizer*, *parent*)

Draws a horizontal line as wide as the window.

GSASII.GSASIIctrlGUI.**ImportMsg**(*parent*, *msgs*)

Show a message with the warnings from importers that could not be installed (due to uninstalled Python packages). Then offer the chance to install GSAS-II packages using *SelectPkgInstall()*

GSASII.GSASIIctrlGUI.**ItemSelector**(*ChoiceList*, *ParentFrame=None*, *title='Select an item'*, *size=None*, *header='Item Selector'*, *useCancel=True*, *multiple=False*)

Provide a wx dialog to select a single item or multiple items from list of choices

Parameters

- **ChoiceList** (*list*) – a list of choices where one will be selected
- **ParentFrame** (*wx.Frame*) – Name of parent frame (default None)
- **title** (*str*) – heading above list of choices (default 'Select an item')
- **size** (*wx.Size*) – Size for dialog to be created (default None – size as needed)
- **header** (*str*) – Title to place on window frame (default 'Item Selector')
- **useCancel** (*bool*) – If True (default) both the OK and Cancel buttons are offered
- **multiple** (*bool*) – If True then multiple items can be selected (default False)

Returns

the selection index or None or a selection list if multiple is true

Called by GSASIIdataGUI.OnReOrgSelSeq() Which is not fully implemented.

Example:

```
choices = ('NXazint1d 1D file','NXazint1d 2D file')
sel = G2G.ItemSelector(choices, ParentFrame=ParentFrame,
                       header='Select file section',
                       title='Select the section of the file to read')
if sel is None: return False
```

GSASII.GSASIIctrlGUI.**Load2Cells**(*G2frame*, *phase*)

Accept two unit cells and use NIST*LATTICE to search for a relationship that relates them.

The first unit cell is initialized as the currently selected phase and the second unit cell is set to the first different phase from the tree. The user can initialize the cell parameters to select a different phase for either cell or can type in the values themselves.

Parameters

- **G2frame** (*wx.Frame*) – The main GSAS-II window
- **phase** (*dict*) – the currently selected frame

GSASII.GSASIIctrlGUI.**MultiColMultiSelDlg**(*parent*, *title*, *header*, *colInfo*, *choices*)

Provides a dialog widget that can be used to select multiple items from a multicolumn list.

Parameters

- **parent** (*wx.Frame*) – the parent frame (or None)
- **title** (*str*) – A title for the dialog window
- **header** (*str*) – A instruction string for the dialog window
- **colInfo** (*list*) – contains three items for each column: a label for the column, a width for the column (in pixels), and True if the column should be right justified.
- **choices** (*list*) – a nested list with values for each row in the table. Within each row should be a list of values for each column. There must be at least one value, but it is OK to have more or fewer values than there are column labels (colInfo). Extra are ignored and unspecified columns are left blank.

Returns

a list of bool values for each entry in choices, True if selected, or None is the dialog is cancelled.

Example use:

```
choices = [('xmltodict', 'Bruker .brml Importer'),
           ('zarr', 'MIDAS Zarr importer'),
           ('h5py', 'HDF5 image importer'),
           ('hdf5', 'HDF5 image importer')]
colInfo = [('package', 50, False),
           ('needed by', 200, True)]
res = G2G.MultiColMultiSelDlg(parent, 'window title', 'Instructions', colInfo,␣
↪choices)
```

class GSASII.GSASIIctrlGUI.**MultiColumnSelection**(*parent*, *title*, *colLabels*, *choices*, *colWidths*, *checkLbl=''*, *height=400*, *centerCols=False*, **args*, ***kw*)

Defines a Dialog widget that can be used to select an item from a multicolumn list. The first column should be short, but remaining columns are word-wrapped if the length of the information extends beyond the column.

When created, the dialog will be shown and <dlg>.Selection will be set to the index of the selected row, or -1. Be sure to use <dlg>.Destroy() to remove the window after reading the selection. If the dialog cannot be shown because a very old version of wxPython is in use, <dlg>.Selection will be None.

If checkLbl is provided with a value, then a set of check buttons starts the table and <dlg>.Selections has the checked rows.

Parameters

- **parent** (*wx.Frame*) – the parent frame (or None)
- **title** (*str*) – A title for the dialog window
- **colLabels** (*list*) – labels for each column
- **choices** (*list*) – a nested list with a value for each row in the table. Within each value should be a list of values for each column. There must be at least one value, but it is OK to have more or fewer values than there are column labels (colLabels). Extra are ignored and unspecified columns are left blank.
- **colWidths** (*list*) – a list of int values specifying the column width for each column in the table (pixels). There must be a value for every column label (colLabels).

- **checkLbl** (*str*) – A label for a row of checkboxes added at the beginning of the table. This option seems to be broken.
- **height** (*int*) – an optional height (pixels) for the table (defaults to 400)
- **centerCols** (*bool*) – if True, items in each column are centered. Default is False

Example use:

```
lbls = ('col 1','col 2','col 3')
choices=(['test1','explanation of test 1'],
         ['b', 'a really really long line that will be word-wrapped'],
         ['test3','more explanation text','optional 3rd column text'])
colWidths=[200,400,100]
dlg = MultiColumnSelection(frm,'select tutorial',lbls,choices,colWidths)
value = choices[dlg.Selection][0]
dlg.Destroy()
```

class GSASII.GSASIIctrlGUI.**MultiDataDialog**(*parent*, *title*, *prompts*, *values*, *limits=[[0.0, 1.0]]*, *testfxns=None*, *formats=['%.5g']*, *header=None*)

Dialog to obtain multiple values from user. Use `dlg.GetValues()` to get the values set in the window.

Parameters

- **parent** (*wx.Frame*) – parent frame for dialog to be created
- **title** (*str*) – title to place on top of window
- **prompts** (*list*) – a string to describe each item. Each entry in this list will designate a row in the generated window.
- **values** (*list*) – a list of initial values for each item. Use a nested list when multiple entries are placed on a single row of the window (see discussion of formats, below). Number of items in the outer list should match the length of `prompts`. The total number of items should match `formats`.
- **limits** (*list*) – A nested list with an upper and lower value for each item or for a choice/edit control a list of allowed values. Use a nested list when multiple entries are placed on a single row of the window (see discussion of formats, below). Number of items in the outer list should match the length of `prompts`. The total number of items should match `formats`.
- **testfxns** (*list*) – A nested list of string test functions. The total number of items should match `formats` or should be left as the default (None).
- **formats** (*list*) – A list of values for each entry in the window. Several different types of values are possible:
  - An "old-style" format string (e.g. `%5d` or `%.3f`) which will be used to display each item's value
  - Or a keyword that specifies how the values are used. Allowed keywords are:
    * `choice`: for a pull-down list;
    * `bool`: for a yes/no checkbox;
    * `str`: for a text entry
    * `edit`: for a pull-down list that allows one to enter an arbitrary value.
  - Alternately, a value can be a list of items, in which case multiple entries are placed on a single row of the window. When this is done, any value in the list other than `choice` or `edit` is used as text to be placed between the ComboBoxes.

The number of items in the outer list should match the length of `prompts`.

- **header** (*str*) – a string to be placed at the top of the window. Ignored if None (the default.)

Example 1:

```
dlg = G2G.MultiDataDialog(G2frame,title='ISOCIF search',
        prompts=['lattice constants tolerance',
                 'coordinate tolerance',
                 'occupancy tolerance'],
        values=[0.001,0.01,0.1],
        limits=3*[[0.,2.]],formats=3*['%.5g'],
        header=isoCite)
dlg.ShowModal()
latTol,coordTol,occTol = dlg.GetValues()
dlg.Destroy()
```

Example 2:

```
nm = [' ','0','1','-1','2','-2','3','-3','4','5','6','7','8','9']
dm = ['1','2','3','4','5','6']
kfmt = ['choice','/','choice',',    ','choice','/','choice',',    ','choice','/',
↪'choice',' ']
dlg = MultiDataDialog(G2frame,title='options',
        prompts=[' k-vector 1 (x,y,z)',
                 ' k-vector 2 (x,y,z)'],
        values=[3*['0','','2',''],3*[' ','','2','']],
        limits=[3*[nm[1:],'',dm,''],3*[nm,'',dm,'']],
        formats=[kfmt,kfmt])
if dlg.ShowModal() == wx.ID_OK: print(dlg.GetValues())
```

class GSASII.GSASIIctrlGUI.**MultiIntegerDialog**(*parent*, *title*, *prompts*, *values*)

Input a series of integers based on prompts

class GSASII.GSASIIctrlGUI.**MultiStringDialog**(*parent*, *title*, *prompts*, *values=[]*, *size=-1*, *addRows=False*, *hlp=None*, *lbl=None*)

Dialog to obtain a multi string values from user

Parameters

- **parent** (*wx.Frame*) – name of parent frame
- **title** (*str*) – title string for dialog
- **prompts** (*list*) – list of strings to tell user what they are inputting
- **values** (*list*) – list of str default input values, if any
- **size** (*int*) – length of the input box in pixels
- **addRows** (*bool*) – if True, users can add rows to the table (default is False)
- **hlp** (*str*) – if supplied, a help button is added to the dialog that can be used to display the supplied help text in this variable.
- **lbl** (*str*) – label placed at top of dialog

Returns

a wx.Dialog instance

GetValues()

Use this method to get the value(s) entered by the user

Returns
a list of strings entered by user

Show()

Use this method after creating the dialog to post it

Returns
True if the user pressed OK; False if the User pressed Cancel

GSASII.GSASIIctrlGUI.**MultipleBlockSelector**(*ChoiceList*, *ParentFrame=None*, *title='Select a block'*, *size=None*, *header='Block Selector'*)

Provide a wx dialog to select a block of data if the file contains more than one set of data and one must be selected. Used in `G2pwd_CIF` only.

Returns
a list of the selected blocks

class GSASII.GSASIIctrlGUI.**MultipleChoicesDialog**(*choicelist*, *headinglist*, *head='Select options'*, *title='Please select from options below'*, *parent=None*)

A dialog that offers a series of choices, each with a title and a wx.Choice widget. Intended to be used Modally. typical input:

- choicelist=[('a','b','c'), ('test1','test2'),('no choice',)]
- headinglist = ['select a, b or c', 'select 1 of 2', 'No option here']

selections are placed in self.chosen when OK is pressed

Also see GSASIIctrlGUI

GSASII.GSASIIctrlGUI.**MultipleChoicesSelector**(*choicelist*, *headinglist*, *ParentFrame=None*, ***kwargs*)

A modal dialog that offers a series of choices, each with a title and a wx.Choice widget. Used in `G2pwd_CIF` only.

Typical input:

- choicelist=[('a','b','c'), ('test1','test2'),('no choice',)]
- headinglist = ['select a, b or c', 'select 1 of 2', 'No option here']

optional keyword parameters are: head (window title) and title returns a list of selected indicies for each choice (or None)

class GSASII.GSASIIctrlGUI.**MyHelp**(*frame*, *includeTree=False*, *morehelpitems=[]*)

A class that creates the contents of a help menu. The menu will start with two entries:

- 'Help on <helpType>': where helpType is a reference to an HTML page to be opened
- About: opens an About dialog using OnHelpAbout. N.B. on the Mac this gets moved to the App menu to be consistent with Apple style.

NOTE: for this to work properly with respect to system menus, the title for the menu must be &Help, or it will not be processed properly:

```
menu.Append(menu=MyHelp(self,...),title="&Help")
```

OnCheckUpdates(*event*)

Check if the GSAS-II repository has an update for the current source files and perform that update if requested.

OnHelpAbout(*event*)

Display an 'About GSAS-II' box

OnHelpById(*event*)

Called when Help on… is pressed in a menu. Brings up a web page for documentation. Uses the G2frame.dataWindow.helpKey value to select what help is shown, unless a special help key value has been defined for the calling menu id (via a lookup in self.HelpById[], which is used for Tutorials & Overall help). Note that G2frame.dataWindow.helpKey SelectDataTreeItem and reflects the data tree item that has been selected.

Note that self here should be a child of the main window (G2frame) where self.frame is G2frame

OnSelectVersion(*event*)

Allow the user to select a specific version of GSAS-II

class GSASII.GSASIIctrlGUI.**NumberValidator**(*typ*, *positiveonly=False*, *xmin=None*, *xmax=None*, *exclLim=[False, False]*, *result=None*, *key=None*, *OKcontrol=None*, *CIFinput=False*)

A validator to be used with a TextCtrl to prevent entering characters other than digits, signs, and for float input, a period and exponents.

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

Parameters

- **typ** (*type*) – the base data type. Must be int or float.
- **positiveonly** (*bool*) – If True, negative integers are not allowed (default False). This prevents the + or - keys from being pressed. Used with typ=int; ignored for typ=float.
- **xmin** (*number*) – Minimum allowed value. If None (default) the lower limit is unbounded
- **xmax** (*number*) – Maximum allowed value. If None (default) the upper limit is unbounded
- **exclLim** (*list*) – if True exclude xmin/xmax value ([exclMin,exclMax]); (Default=[False,False])
- **result** (*dict/list*) – List or dict where value should be placed when valid
- **key** (*any*) – key to use for result (int for list)
- **OKcontrol** (*function*) – function or class method to control an OK button for a window. Ignored if None (default)
- **CIFinput** (*bool*) – allows use of a single '?' or '.' character as valid input.

CheckInput(*previousInvalid*)

called to test every change to the TextCtrl for validity and to change the appearance of the TextCtrl

Anytime the input is invalid, call self.OKcontrol (if defined) because it is fast. If valid, check for any other invalid entries only when changing from invalid to valid, since that is slower.

Parameters
previousInvalid (*bool*) – True if the TextCtrl contents were invalid prior to the current change.

Clone()

Create a copy of the validator, a strange, but required component

OnChar(*event*)

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

ShowValidity(*tc*)

Set the control colors to show invalid input

Parameters

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

TestValid(*tc*)

Check if the value is valid by casting the input string into the current type.

Set the invalid variable in the TextCtrl object accordingly.

If the value is valid, save it in the dict/list where the initial value was stored, if appropriate.

Parameters

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

TransferFromWindow()

Needed by validator, strange, but required component

TransferToWindow()

Needed by validator, strange, but required component

class GSASII.GSASIIctrlGUI.**OpenGitTutorial**(*parent*)

Open a tutorial web page from the git repository, optionally copying the tutorial's exercise data file(s) to the local disk.

ChooseTutorial2(*choices*)

Select tutorials from a two-column table, when possible

SelectAndDownload(*event*)

Shows a list of all tutorials so user can select one to view. The data files associated with that directory are then downloaded.

SelectDownloadLoc(*event*)

Select a download location, Cancel resets to the default

SetTutorialPath()

Get the tutorial location if set; if not pick a default directory in a logical place

onWebBrowse(*event*)

Shows a list of all tutorials so user can select one to view.

Returns

the name of the directory where the tutorial is located, which is used if called from *SelectAndDownload()*.

class GSASII.GSASIIctrlGUI.**OrderBox**(*parent*, *keylist*, *vallookup*, *posdict*, **arg*, ***kw*)

Creates a panel with scrollbars where items can be ordered into columns

Parameters

- **keylist** (*list*) – is a list of keys for column assignments
- **vallookup** (*dict*) – is a dict keyed by names in keylist where each item is a dict. Each inner dict contains variable names as keys and their associated values
- **posdict** (*dict*) – is a dict keyed by names in keylist where each item is a dict. Each inner dict contains column numbers as keys and their associated variable name as a value. This is used for both input and output.

OnChoice(*event*)

Called when a column is assigned to a variable

GSASII.GSASIIctrlGUI.**PhaseSelector**(*ChoiceList*, *ParentFrame=None*, *title='Select a phase'*, *size=None*, *header='Phase Selector'*)

Provide a wx dialog to select a phase, used in importers if a file contains more than one phase

class GSASII.GSASIIctrlGUI.**PickTwoDialog**(*parent*, *title*, *prompt*, *names*, *choices*)

This does not seem to be in use

GSASII.GSASIIctrlGUI.**ReadOnlyTextCtrl**(**args*, ***kwargs*)

Create a read-only TextCtrl for display of constants This is probably not ideal as it mixes visual cues, but it does look nice. Addresses 4.2 bug where TextCtrl has no default size

class GSASII.GSASIIctrlGUI.**RefinementProgress**(*title='Residual'*, *message='All data Rw ='*, *maximum=101*, *parent=None*, *trialMode=False*, *seqLen=0*, *style=None*)

Defines a wrapper to place around wx.ProgressDialog to be used for showing refinement progress. At some point a better progress window should be created that keeps useful info on the screen such as some starting and current fit metrics, but for now all this adds is window defaults and a wx.Yield call during progress update calls.

Update(*value*, *newmsg=''*) → Tuple[bool, bool]

Updates the dialog, setting the progress bar to the new value and updating the message if new one is specified.

class GSASII.GSASIIctrlGUI.**SGMagSpinBox**(*parent*, *title*, *text*, *table*, *Cents*, *names*, *spins*, *ifGray*)

Special version of MessageBox that displays magnetic spin text

Show()

Use this method after creating the dialog to post it

class GSASII.GSASIIctrlGUI.**SGMessageBox**(*parent*, *title*, *text*, *table*, *spins=[]*)

Special version of MessageBox that displays space group & super space group text in two blocks

Show()

Use this method after creating the dialog to post it

GSASII.GSASIIctrlGUI.**SaveCite**(*prog*, *text*)

Save citation information as it is referenced so that all of it can be displayed in the About GSAS-II window

GSASII.GSASIIctrlGUI.**SaveConfigVars**(*vars*, *parent=None*)

Write the current config variable values to ~/.GSASII/config.ini

Params dict vars
a dictionary of variable settings and meanings as created in `GetConfigValsDocs()`. Most of the information gathered in GetConfigValsDocs is no longer used here.

Parameters
parent – wx.Frame object or None. No longer used.

Returns
True if unable to write the file, None otherwise

class GSASII.GSASIIctrlGUI.**ScrolledMultiEditor**(*parent*, *dictlst*, *elemlst*, *prelbl=[]*, *postlbl=[]*, *title='Edit items'*, *header=''*, *size=(300, 250)*, *CopyButton=False*, *ASCIIonly=False*, *minvals=[]*, *maxvals=[]*, *sizevals=[]*, *checkdictlst=[]*, *checkelemlst=[]*, *checklabel=''*)

Define a window for editing a potentially large number of dict- or list-contained values with validation for each item. Edited values are automatically placed in their source location. If invalid entries are provided, the TextCtrl is turned yellow and the OK button is disabled.

The type for each TextCtrl validation is determined by the initial value of the entry (int, float or string). Float values can be entered in the TextCtrl as numbers or also as algebraic expressions using operators + - / * () and **, in addition pi, sind(), cosd(), tand(), and sqrt() can be used, as well as appreviations s(), sin(), c(), cos(), t(), tan() and sq().

Parameters

- **parent** (*wx.Frame*) – name of parent window, or may be None
- **dictlst** (*tuple*) – a list of dicts or lists containing values to edit
- **elemlst** (*tuple*) – a list of keys/indices for items in dictlst. Note that elemlst must have the same length as dictlst, where each item in elemlst will will match an entry for an entry for successive dicts/lists in dictlst.
- **prelbl** (*tuple*) – a list of labels placed before the TextCtrl for each item (optional)
- **postlbl** (*tuple*) – a list of labels placed after the TextCtrl for each item (optional)
- **title** (*str*) – a title to place in the frame of the dialog
- **header** (*str*) – text to place at the top of the window. May contain new line characters.
- **size** (*wx.Size*) – a size parameter that dictates the size for the scrolled region of the dialog. The default is (300,250).
- **CopyButton** (*bool*) – if True adds a small button that copies the value for the current row to all fields below (default is False)
- **ASCIIonly** (*bool*) – if set as True will remove unicode characters from strings
- **minvals** (*list*) – optional list of minimum values for validation of float or int values. Ignored if value is None.
- **maxvals** (*list*) – optional list of maximum values for validation of float or int values. Ignored if value is None.
- **sizevals** (*list*) – optional list of wx.Size values for each input widget. Ignored if value is None.
- **checkdictlst** (*tuple*) – an optional list of dicts or lists containing bool values (similar to dictlst).
- **checkelemlst** (*tuple*) – an optional list of dicts or lists containing bool key values (similar to elemlst). Must be used with checkdictlst.
- **checklabel** (*string*) – a string to use for each checkbutton

Returns

the wx.Dialog created here. Use method .ShowModal() to display it.

Example for use of ScrolledMultiEditor:

```
dlg = <pkg>.ScrolledMultiEditor(frame,dictlst,elemlst,prelbl,postlbl,
                                header=header)
if dlg.ShowModal() == wx.ID_OK:
     for d,k in zip(dictlst,elemlst):
         print d[k]
```

Example definitions for dictlst and elemlst:

```
    dictlst = (dict1,list1,dict1,list1)
    elemlst = ('a', 1, 2, 3)

This causes items dict1['a'], list1[1], dict1[2] and list1[3] to be edited.
```

Note that these items must have int, float or str values assigned to them. The dialog will force these types to be retained. String values that are blank are marked as invalid.

ControlOKButton(*setvalue*)

Enable or Disable the OK button for the dialog. Note that this is passed into the ValidatedTxtCtrl for use by validators.

Parameters

setvalue (*bool*) – if True, all entries in the dialog are checked for validity. if False then the OK button is disabled.

class GSASII.GSASIIctrlGUI.**ScrolledStaticText**(*parent*, *label=''*, *delay=100*, *lbllen=15*, *dots=True*, ***kwargs*)

Fits a long string into a small space by scrolling it. Inspired by ActiveText.py from J Healey <rolfofsaxony@gmx.com> https://discuss.wxpython.org/t/activetext-rather-than-statictext/36370

Use examples:

```
frm = wx.Frame(None) # create a frame
ms = wx.BoxSizer(wx.VERTICAL)
text = 'this is a long string that will be scrolled'
ms.Add(G2G.ScrolledStaticText(frm,label=text))
txt = G2G.ScrolledStaticText(frm,label=text, lbllen=20)
smallfont = wx.SystemSettings.GetFont(wx.SYS_SYSTEM_FONT)
smallfont.SetPointSize(10)
txt.SetFont(smallfont)
ms.Add(txt)
ms.Add(G2G.ScrolledStaticText(frm,label=text,dots=False,delay=250,lbllen=20))
frm.SetSizer(ms)
```

Parameters

- **parent** (*w.Frame*) – Frame or Panel where widget will be placed
- **label** (*str*) – string to be displayed
- **delay** (*int*) – time between updates in ms (default is 100)
- **lbllen** (*int*) – number of characters to show (default is 15)
- **dots** (*bool*) – If True (default) ellipsis (...) are placed at the beginning and end of the string when any characters in the string are not shown. The displayed string length will thus be lbllen+6 most of the time
- **(other)** – other optional keyword parameters for the wx.StaticText widget such as size or style may be specified.

class GSASII.GSASIIctrlGUI.**SelectConfigSetting**(*parent*)

Dialog to select configuration variables and set associated values.

OnBoolSelect(*event*)

Respond to a change in a True/False variable

OnChange(*event=None*)

Check if anything been changed. Turn the save button on/off.

OnSave(*event*)

Write the config variables to ~/.GSASII/config.ini as the current settings

OnSelection()

show a selected variable and allow it to be changed

onSelColor(*event*)

Select a color from a menu

onSelDir(*event*)

Select a directory from a menu

onSelExec(*event*)

Select an executable file from a menu

GSASII.GSASIIctrlGUI.**SelectEdit1Var**(*G2frame*, *array*, *labelLst*, *elemKeysLst*, *dspLst*, *refFlgElem*)

Select a variable from a list, then edit it and select histograms to copy it to.

Parameters

- **G2frame** (*wx.Frame*) – main GSAS-II frame
- **array** (*dict*) – the array (dict or list) where values to be edited are kept
- **labelLst** (*list*) – labels for each data item
- **elemKeysLst** (*list*) – a list of lists of keys needed to be applied (see below) to obtain the value of each parameter
- **dspLst** (*list*) – list list of digits to be displayed (10,4) is 10 digits with 4 decimal places. Can be None.
- **refFlgElem** (*list*) – a list of lists of keys needed to be applied (see below) to obtain the refine flag for each parameter or None if the parameter does not have refine flag.

Example:

```
array = data
labelLst = ['v1','v2']
elemKeysLst = [['v1'], ['v2',0]]
refFlgElem = [None, ['v2',1]]

* The value for v1 will be in data['v1'] and this cannot be refined while,
* The value for v2 will be in data['v2'][0] and its refinement flag is data['v2
↪'][1]
```

GSASII.GSASIIctrlGUI.**SelectPkgInstall**(*event*)

Offer the user a chance to install Python packages needed by one or more importers. There might be times where something like this will be useful for other GSAS-II actions.

GSASII.GSASIIctrlGUI.**SelectSearchVars**(*G2frame*, *labelLst*, *keyDict*)

Get a sample parameter and a comment label from the user so we can search for that in the comments.

Returns

Selection, Key, where Selection is the parameter name to be set (from labelLst) and Key will be a value tag from keyDict or an equation object (from `GSASIIexprGUI.ExpressionDialog()`). Both will be None if command is cancelled.

`GSASII.GSASIIctrlGUI.`**`ShowCitations`**(*event*)

Show all work that GSAS-II users may wish to cite

`GSASII.GSASIIctrlGUI.`**`ShowHelp`**(*helpType*, *frame*, *helpMode=None*)

Called to bring up a web page for documentation.

class `GSASII.GSASIIctrlGUI.`**`ShowLSParms`**(*G2frame*, *title*, *parmDict*, *varyList*, *fullVaryList*, *Controls*, *size=(650, 430)*)

Create frame to show least-squares parameters

`DrawPanel`()

Draws the contents of the entire dialog. Called initially & when radio buttons are pressed

`repaintScrollTbl`()

Shows the selected variables in a ListCtrl

`GSASII.GSASIIctrlGUI.`**`ShowScrolledColText`**(*parent*, *txt*, *width=600*, *height=400*, *header='Warning info'*, *col1len=999*)

Simple code to display tabular information in a scrolled wx.Dialog window.

Lines ending with a colon (:) are centered across all columns and have a grey background. Lines beginning and ending with '**' are also are centered across all columns and are given a yellow background All other lines have columns split by tab (t) characters.

Parameters

- **`parent`** (*`wx.Frame`*) – parent window
- **`txt`** (*`str`*) – text to be displayed
- **`width`** (*`int`*) – lateral of window in pixels (defaults to 600)
- **`height`** (*`int`*) – vertical dimension of window in pixels (defaults to 400)
- **`header`** (*`str`*) – title to be placed on window

`GSASII.GSASIIctrlGUI.`**`ShowScrolledInfo`**(*parent*, *txt*, *width=600*, *height=400*, *header='Warning info'*, *buttonlist=None*)

Simple code to display possibly extensive error or warning text in a scrolled window.

Parameters

- **`parent`** (*`wx.Frame`*) – parent window for
- **`txt`** (*`str`*) – text to be displayed
- **`width`** (*`int`*) – lateral of window in pixels (defaults to 600)
- **`height`** (*`int`*) – vertical dimension of window in pixels (defaults to 400)
- **`header`** (*`str`*) – title to be placed on window
- **`buttonlist`** (*`list`*) – list of button Ids to show, or one or more pairs of values, where the first is a label to place on the button and the second is a routine that is called if the button is pressed. The default is None which places a single "Close" button that returns wx.ID_CANCEL

Returns

the wx Id for the selected button

Example:

```
res = ShowScrolledInfo(self.frame,msg,header='Please Note',buttonlist=[
        ('Open', lambda event: event.GetEventObject().GetParent().EndModal(wx.ID_
↪OK)),
        ('Skip', lambda event: event.GetEventObject().GetParent().EndModal(wx.ID_
↪CANCEL))
       ])
if res == wx.ID_OK:
    pass
```

GSASII.GSASIIctrlGUI.**ShowWebPage**(*URL*, *frame*, *browser=False*, *internal=False*)

Called to show a tutorial web page.

Parameters

- **URL** (*str*) – web page URL
- **frame** (*wx.Frame*) – parent window (or None)
- **browser** (*bool*) – If True, forces the page to be opened in a web browser, regardless of the Help_mode config setting.

class GSASII.GSASIIctrlGUI.**SingleFloatDialog**(*parent*, *title*, *prompt*, *value*, *limits=[0.0, 1.0]*, *fmt='%.5g'*)

Dialog to obtain a single float value from user

Parameters

- **parent** (*wx.Frame*) – name of parent frame
- **title** (*str*) – title string for dialog
- **prompt** (*str*) – string to tell user what they are inputing
- **value** (*str*) – default input value, if any
- **limits** (*list*) – upper and lower value used to set bounds for entry, use [None,None] for no bounds checking, [None,val] for only upper bounds, etc. Default is [0,1]. Values outside of limits will be ignored.
- **format** (*str*) – string to format numbers. Defaults to '%.5g'. Use '%d' to have integer input (but dlg.GetValue will still return a float).

Typical usage:

```
limits = (0,1)
dlg = G2G.SingleFloatDialog(G2frame,'New value','Enter new value for...',default,
↪limits)
if dlg.ShowModal() == wx.ID_OK:
    parm = dlg.GetValue()
dlg.Destroy()
```

ControlOKButton(*setvalue*)

Enable or Disable the OK button for the dialog. Note that this is passed into the ValidatedTxtCtrl for use by validators.

Parameters

setvalue (*bool*) – if True, all entries in the dialog are checked for validity. if False then the OK button is disabled.

class GSASII.GSASIIctrlGUI.**SingleIntDialog**(*parent*, *title*, *prompt*, *value*, *limits=[None, None]*)

Dialog to obtain a single int value from user

Parameters

- **parent** (*wx.Frame*) – name of parent frame
- **title** (*str*) – title string for dialog
- **prompt** (*str*) – string to tell user what they are inputing
- **value** (*str*) – default input value, if any
- **limits** (*list*) – upper and lower value used to set bounds for entries. Default is [None,None] – for no bounds checking; use [None,val] for only upper bounds, etc. Default is [0,1]. Values outside of limits will be ignored.

Typical usage:

```
limits = (0,None)  # allows zero or positive values only
dlg = G2G.SingleIntDialog(G2frame,'New value','Enter new value for...',default,
↪limits)
if dlg.ShowModal() == wx.ID_OK:
    parm = dlg.GetValue()
dlg.Destroy()
```

class GSASII.GSASIIctrlGUI.**SingleStringDialog**(*parent*, *title*, *prompt*, *value=''*, *size=(200, -1)*, *help=''*, *choices=None*)

Dialog to obtain a single string value from user

Parameters

- **parent** (*wx.Frame*) – name of parent frame
- **title** (*str*) – title string for dialog
- **prompt** (*str*) – string to tell use what they are inputting
- **value** (*str*) – default input value, if any
- **size** (*tuple*) – specifies default size and width for the text entry section of the dialog [default (200,-1)]. If the vertical size (the second number) is greater than 20 (~ a single line) then the textbox will allow inclusion of new-line characters. In single-line mode, return causes the dialog to close.
- **help** (*str*) – if supplied, a help button is added to the dialog that can be used to display the supplied help text/URL for setting this variable. (Default is '', which is ignored.)
- **choices** (*list*) – a set of strings that provide optional values that can be selected from; these can be edited if desired.

GetValue()

Use this method to get the value entered by the user :returns: string entered by user

Show()

Use this method after creating the dialog to post it :returns: True if the user pressed OK; False if the User pressed Cancel

class GSASII.GSASIIctrlGUI.**SortableLstCtrl**(*parent*)

Creates a read-only table with sortable columns. Sorting is done by clicking on a column label. A triangle facing up or down is added to indicate the column is sorted.

To use, the header is labeled using *PopulateHeader()*, then *PopulateLine()* is called for every row in table and finally *SetColWidth()* is called to set the column widths.

Parameters

parent (*wx.Frame*) – parent object for control

Example:

```
data = [
   (':0:sig-0', '30.7906', '30.7907', 'Pwd=SNAP066555: TOF profile term'),
   (':0:sig-1', '208.419', '208.419', 'Pwd=SNAP066555: TOF profile term'),
   (':0:Scale', '582006', '582006', 'Pwd=SNAP066555: Scale factor'),
   (':1:Scale', '592440', '592440', 'Pwd=SNAP06 (1): Scale factor'),
]
sortPanel = G2G.SortableLstCtrl(G2frame)
sortPanel.PopulateHeader([f'label{i}' for i in range(4)],4*[0])
for i,l in enumerate(data): sortPanel.PopulateLine(i,l)
for i in range(4): sortPanel.SetColWidth(i) # set width to automatic
sortPanel.SetColWidth(1,sortType='float') # sort 1st column by numeric value not␣
↪str
```

PopulateHeader(*header*, *justify*)

Defines the column labels

Parameters

- **header** (*list*) – a list of strings with header labels
- **justify** (*list*) – a list of int values where 0 causes left justification, 1 causes right justification, and -1 causes centering

PopulateLine(*key*, *data*)

Enters each row into the table

Parameters

- **key** (*int*) – a unique int value for each line, probably should be sequential
- **data** (*list*) – a list of strings for each column in that row

SetColWidth(*col*, *width=None*, *auto=True*, *minwidth=0*, *maxwidth=None*, *sortType='str'*)

Sets the column width.

Parameters

- **width** (*int*) – the column width in pixels
- **auto** (*bool*) – if True (default) and width is None (default) the width is set by the maximum width entry in the column
- **minwidth** (*int*) – used when auto is True, sets a minimum column width
- **maxwidth** (*int*) – used when auto is True, sets a maximum column width. Do not use with minwidth

SetInitialSortColumn(*col*, *ascending=True*)

Sets the initial column to be used for sorting when the table is first displayed. This method should be called after all PopulateLine calls are complete. The up or down arrow indicator will be displayed on the specified column.

Parameters

- **col** (*int*) – the column index (0-based) to sort by initially
- **ascending** (*bool*) – if True (default), sort in ascending order; if False, descending

GSASII.GSASIIctrlGUI.**StringSearchTemplate**(*parent*, *title*, *prompt*, *start*, *help=None*)

Dialog to obtain a single string value from user

Parameters

- **parent** (*wx.Frame*) – name of parent frame
- **title** (*str*) – title string for dialog
- **prompt** (*str*) – string to tell use what they are inputting
- **start** (*str*) – default input value, if any
- **help** (*str*) – if supplied, a help button is added to the dialog that can be used to display the supplied help text/URL for setting this variable. (Default is '', which is ignored.)

GSASII.GSASIIctrlGUI.**StripIndents**(*msg*, *singleLine=False*)

Strip unintended indentation from multiline strings. When singleLine is True, all newline are removed, but inserting "%%" into the string will cause a blank line to be inserted at that point and %t% will generate a new line and tab (to indent a line)

Parameters

- **msg** (*str*) – a string containing one or more lines of text. spaces or tabs following a newline are removed.
- **singleLine** (*bool*) – removes all newlines from the msg so that the text may be wrapped.

Returns

the string but reformatted

GSASII.GSASIIctrlGUI.**StripUnicode**(*string*, *subs='.'*)

Strip non-ASCII characters from strings

Parameters

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

Returns

a new string with only ASCII characters

class GSASII.GSASIIctrlGUI.**Table**(*data=[]*, *rowLabels=None*, *colLabels=None*, *types=None*)

Basic data table for use with GSgrid

AppendRows(*numRows=1*) → bool

Append additional rows at the end of the table.

CanGetValueAs(*row*, *col*, *typeName*) → bool

Returns true if the value of the given cell can be accessed as if it were of the specified type.

CanSetValueAs(*row*, *col*, *typeName*) → bool

Returns true if the value of the given cell can be set as if it were of the specified type.

GetColLabelValue(*col*) → str

Return the label of the specified column.

GetNumberCols() → int

Must be overridden to return the number of columns in the table.

GetNumberRows() → int

Must be overridden to return the number of rows in the table.

GetRowLabelValue(*row*) → str

Return the label of the specified row.

GetTypeName(*row*, *col*) → str

Returns the type of the value in the given cell.

GetValue(*row*, *col*) → Any

Must be overridden to implement accessing the table values as text.

InsertRows(*pos=0*, *numRows=1*) → bool

Insert additional rows into the table.

IsEmptyCell(*row*, *col*) → bool

May be overridden to implement testing for empty cells.

SetColLabelValue(*col*, *label*) → None

Exactly the same as SetRowLabelValue() but for columns.

SetRowLabelValue(*row*, *label*) → None

Set the given label for the specified row.

SetValue(*row*, *col*, *value*) → None

Must be overridden to implement setting the table values as text.

class GSASII.GSASIIctrlGUI.**ValidatedTxtCtrl**(*parent*, *loc*, *key*, *nDig=None*, *notBlank=True*, *xmin=None*, *xmax=None*, *OKcontrol=None*, *OnLeave=None*, *typeHint=None*, *CIFinput=False*, *exclLim=[False, False]*, *OnLeaveArgs={}*, *ASCIIonly=False*, ***kw*)

Create a TextCtrl widget that uses a validator to prevent the entry of inappropriate characters and changes color to highlight when invalid input is supplied. As valid values are typed, they are placed into the dict or list where the initial value came from. The type of the initial value must be int, float or str or None (see `key` and `typeHint`); this type (or the one in `typeHint`) is preserved. Values are processed and saved when Enter is pressed, when the mouse is moved to another control (leave window or focus is lost) or after a change and a delay of two seconds.

Float values can be entered in the TextCtrl as numbers or also as algebraic expressions using operators (+ - / * () and **), in addition pi, sind(), cosd(), tand(), and sqrt() can be used, as well as appreviations s, sin, c, cos, t, tan and sq.

Parameters

- **parent** (*wx.Panel*) – name of panel or frame that will be the parent to the TextCtrl. Can be None.
- **loc** (*dict/list*) – the dict or list with the initial value to be placed in the TextCtrl.
- **key** (*int/str*) – the dict key or the list index for the value to be edited by the TextCtrl. The `loc[key]` element must exist, but may have value None. If None, the type for the element is taken from `typeHint` and the value for the control is set initially blank (and thus invalid.) This is a way to specify a field without a default value: a user must set a valid value.

  If the value is not None, it must have a base type of int, float, str or unicode; the TextCrtl will be initialized from this value.

- **nDig** (*list*) – number of digits, places and optionally the format ([nDig,nPlc,fmt]) after decimal to use for display of float. The format is either 'f' (default) or 'g'. Alternately, None can be specified which causes numbers to be displayed with approximately 5 significant figures for floats. If this is specified, then `typeHint` = float becomes the default. (Default=None).
- **notBlank** (*bool*) – if True (default) blank values are invalid for str inputs.
- **xmin** (*number*) – minimum allowed valid value. If None (default) the lower limit is unbounded. NB: test in NumberValidator is val >= xmin not val > xmin
- **xmax** (*number*) – maximum allowed valid value. If None (default) the upper limit is unbounded NB: test in NumberValidator is val <= xmax not val < xmax
- **exclLim** (*list*) – if True exclude min/max value ([exclMin,exclMax]); (Default=[False,False])
- **OKcontrol** (*function*) – specifies a function or method that will be called when the input is validated. The called function is supplied with one argument which is False if the TextCtrl contains an invalid value and True if the value is valid. Note that this function should check all values in the dialog when True, since other entries might be invalid. The default for this is None, which indicates no function should be called.
- **OnLeave** (*function*) – specifies a function or method that will be called when the focus for the control is lost. The called function is supplied with (at present) three keyword arguments:
  - invalid: (*bool*) True if the value for the TextCtrl is invalid
  - value: (*int/float/str*) the value contained in the TextCtrl
  - tc: (*wx.TextCtrl*) the TextCtrl object

  The number of keyword arguments may be increased in the future should needs arise, so it is best to code these functions with a **kwargs argument so they will continue to run without errors

  The default for OnLeave is None, which indicates no function should be called.
- **typeHint** (*type*) – the value of typeHint should be int, float or str (or None). The value for this will override the initial type taken from value for the dict/list element `loc[key]` if not None and thus specifies the type for input to the TextCtrl. Defaults as None, which is ignored, unless `nDig` is specified in which case the default is float.
- **CIFinput** (*bool*) – for str input, indicates that only printable ASCII characters may be entered into the TextCtrl. Forces output to be ASCII rather than Unicode. For float and int input, allows use of a single '?' or '.' character as valid input.
- **OnLeaveArgs** (*dict*) – a dict with keyword args that are passed to the `OnLeave` function. Defaults to `{}`
- **ASCIIonly** (*bool*) – if set as True will remove unicode characters from strings
- **(other)** – other optional keyword parameters for the wx.TextCtrl widget such as size or style may be specified.

ChangeValue(*val*)

Place a value into the text widget and save it into the associated array element. Note that, unlike the stock wx.TextCtrl, val is expected to be in the form expected by the widget (float/int/str) rather than only str's. For float val values, the value is formatted when placed in the TextCtrl, but the supplied value is what is actually saved.

This routine does not trigger a wx.EVT_TEXT event. This is what should be used inside event callbacks, not *SetValue()*.

OnKeyDown(*event*)

Special callback for wx 2.9+ on Mac where backspace is not processed by validator

SaveBackgroundColor(*color*)

Use this color when valid rather than default

SetValue(*val*, *warn=True*)

Place a value into the text widget and save it into the associated array element. Note that, unlike the stock wx.TextCtrl, val is expected to be in the form expected by the widget (float/int/str) rather than only str's. For float val values, the value is formatted when placed in the TextCtrl, but the supplied value is what is actually saved.

This routine triggers a wx.EVT_TEXT event

ShowStringValidity(*previousInvalid=True*)

Check if input is valid. Anytime the input is invalid, call self.OKcontrol (if defined) because it is fast. If valid, check for any other invalid entries only when changing from invalid to valid, since that is slower.

Parameters

previousInvalid (*bool*) – True if the TextCtrl contents were invalid prior to the current change.

class GSASII.GSASIIctrlGUI.**VirtualVarBox**(*parent*)

This is used to construct the parameter list display for viewing least squares parameters

OnGetItemAttr(*item*) → ItemAttr

This function may be overridden in the derived class for a control with wxLC_VIRTUAL style.

OnGetItemText(*item*, *column*) → str

This function must be overridden in the derived class for a control with wxLC_VIRTUAL style.

OnRowRightClick(*event*, *row=None*)

This adds or removes a variable from the list of logged paremeters.

When a row is right-clicked this seems to be called twice, once event.GetIndex() of -1, then OnRowSelected is called and then this is called again, but where event.GetIndex() is the row that was selected.

OnRowSelected(*event*, *row=None*)

Creates an edit window when a parameter is selected

GSASII.GSASIIctrlGUI.**XformMatrix**(*panel*, *Trans*, *Uvec*, *Vvec*, *OnLeave=None*, *OnLeaveArgs=[]*)

Display a transformation matrix and two vectors

GSASII.GSASIIctrlGUI.**askQuestion**(*parent*, *question*, *title*)

Simple code to ask a Y/N question and get answer

GSASII.GSASIIctrlGUI.**askSaveDirectory**(*G2frame*)

Ask the user to supply a directory name. Path name is used as the starting point for the next export path search.

Returns

a directory name (str) or None if Cancel is pressed

GSASII.GSASIIctrlGUI.**askSaveFile**(*G2frame*, *defnam*, *extension*, *longFormatName*, *parent=None*)

Ask the user to supply a file name

Parameters

- **G2frame** (*wx.Frame*) – The main GSAS-II window
- **defnam** (*str*) – a default file name

- **extension** (*str*) – the default file extension beginning with a '.'
- **longFormatName** (*str*) – a description of the type of file
- **parent** (*wx.Frame*) – the parent window for the dialog. Defaults to G2frame.

Returns
a file name (str) or None if Cancel is pressed

GSASII.GSASIIctrlGUI.**findValsInNotebook**(*data*, *target*)

Pull a string of values from saved values in the GSAS-II notebook

GSASII.GSASIIctrlGUI.**getTextSize**(*txt*)

Get the size of the text string txt in points, returns (x,y)

GSASII.GSASIIctrlGUI.**gitCheckUpdates**(*G2frame*)

Used to update to the latest GSAS-II version, but checks for a variety of repository conditions that could make this process more complex. If there are uncommitted local changes, these changes must be cached or deleted first. If there are local changes that have been committed or a new branch has been created, the user (how obstensibly must know use of git) will probably need to do this manually. If GSAS-II has previously been regressed (using *gitSelectVersion()*), then this is noted as well.

When all is done, function GSASIIpath.gitStartUpdate() is called to actually perform the update.

GSASII.GSASIIctrlGUI.**gitSelectBranch**(*event*)

Pull in latest GSAS-II branches on origin server; Allow user to select a branch; checkout that branch and restart GSAS-II. Expected to be used by developers and by expert users only.

GSASII.GSASIIctrlGUI.**gitSelectVersion**(*G2frame*)

Used to regress to a previous GSAS-II version, checking first for a variety of repository conditions that could make this process more complex. If there are uncommitted local changes, these changes must be cached or deleted before a different version can be installed. If there are local changes that have been committed or a new branch has been created, the user (how obstensibly must know use of git) will probably need to do this manually. If GSAS-II has previously been regressed (using *gitSelectVersion()*), then this is noted as well.

When all is done, function GSASIIpath.gitStartUpdate() is called to actually perform the update.

GSASII.GSASIIctrlGUI.**gitSwitchMaster2Main**()

This is "patch" code to switch from the master branch to the main branch. At some point this will be made part of the update process in the master branch. This routine is not needed in the main branch.

Switching is a bit complicated as additional Python packages are needed and becuase the GSASII.py file gets renamed to G2.py so "shortcuts" need to be re-created to reference that.

class GSASII.GSASIIctrlGUI.**gitVersionSelector**(*parent=None*)

Dialog to allow a user to select a version of GSAS-II to install from a git repository

docCommit(*commit*)

Provides a string with information about a specific git commit.

Returns
a multi-line string

getVersion()

Gets the selected version that should be installed

Returns

returns one of three values:

- 0: if the newest version is selected, so that the installation should be updated rather than regressed

- None: if the currently installed version is selected, so that nothing need be done
- A hexsha string: the regressed version that should be selected.

class GSASII.GSASIIctrlGUI.**gpxFileSelector**(*parent*, *startdir='.'*, *multiple=False*, **args*, ***kwargs*)

Create a file selection widget for locating .gpx files as a modal dialog. Displays status information on selected files. After creating this use dlg.ShowModal() to wait for selection of a file. If dlg.ShowModal() returns wx.ID_OK, use dlg.Selection (multiple=False) to obtain the selected file or dlg.Selections (multiple=True) to obtain a list of multiple files.

Parameters

- **parent** (*wx.Frame*) – name of panel or frame that will be the parent to the dialog. Can be None.
- **startdir** (*path*) – Specifies the initial directory that is opened when the window is initially opened. Default is '.'
- **multiple** (*bool*) – if True, checkboxes are used to allow selection of multiple files. Default is False

DirSelected(*event=None*, **args*, ***kwargs*)

Respond to a directory being selected. List files found in fileBox and clear any selections. Also clear any reference to a timer.

FileSelected(*event*)

Respond to a file being selected (or checked in multiple mode)

displayGPXrtc(*result*, *fwp*)

Show info about selected file in a RichText display

GSASII.GSASIIctrlGUI.**haveGUI**()

Test if there is a GUI that can be accessed :returns: True if a GUI is available

GSASII.GSASIIctrlGUI.**helpLocDict = {}**

This is an index to the HTML anchors defined in the GSAS-II help files

GSASII.GSASIIctrlGUI.**makeContourSliders**(*G2frame*, *Ymax*, *PlotPatterns*, *newPlot*, *plottype*)

Create a non-modal dialog for sliders to set contour plot intensity thresholds.

GSASII.GSASIIctrlGUI.**patch_condarc**()

Comment out any references to "file:" locations in the .condarc file. These should not be there and cause problems.

class GSASII.GSASIIctrlGUI.**popupSelectorButton**(*parent*, *lbl*, *choices=None*, *selected=None*, *choiceDict=None*, *OnChange=None*, ***kw*)

Create a button that will invoke a menu with choices that can be selected. Do special stuff if the first item is "all"

TODO: It might be better to make this a wx.ComboCtrl if I can figure out how to make that work, or perhaps make that an option

Parameters

- **parent** (*wx.Frame*) – a panel or frame that is the parent to this button
- **lbl** (*str*) – a label for the button
- **choices** (*list*) – a list of str's with labels for the items in the menu
- **selected** (*list*) – a list of bool's that determine if the menu item is initial selected
- **OnChange** (*function*) – an optional function that is called after the menu is removed

- **others** – other keyword parameters are allowed. They will be passed to the OnChange routine.

Param
dict choiceDict: a dict with both choices and their values (selections). Use this or choices & selected, not both. If this is used, the values are set as radiobutton choices, only the most recent setting is selected.

popupSelector(*event*)

Show the menu and then get current values. Optionally call the OnChange routine.

GSASII.GSASIIctrlGUI.**setColorButton**(*parent*, *array*, *key*, *callback=None*, *callbackArgs=[]*)

Define a button for setting colors This bypasses the bug in wx4.1.x in ColourSelect

GSASII.GSASIIctrlGUI.**showUniqueCell**(*frame*, *cellSizer*, *row*, *cell*, *SGData=None*, *editAllowed=False*, *OnCellChange=None*)

function to put cell values into a GridBagSizer. First column (#0) is reserved for labels etc. if editAllowed is True, values are placed in a wx.TextCtrl and if needed two rows are used in the table.

GSASII.GSASIIctrlGUI.**skimGPX**(*fl*)

pull out fit information from a .gpx file quickly

Returns
dict with status info

GSASII.GSASIIctrlGUI.**updateNoticeDict = {4919: True}**

A dict with versions that should be noted. The value associated with the tag is if all older projects should show the warning, or only the first to be opened.

GSASII.GSASIIctrlGUI.**updateNotifier**(*G2frame*, *fileVersion*)

Posts an update notice when a a specially tagged GSAS-II version is seen for the first time. Versions to be tagged are set in global updateNoticeDict; version info is found in file versioninfo.txt.

Parameters

- **G2frame** (*wx.Frame*) – GSAS-II main window
- **fileVersion** (*int*) – version of GSAS-II used to create the current .gpx file

GSASII.GSASIIctrlGUI.**viewWebPage**(*parent*, *URL=''*, *size=(750, 450)*, *newFrame=False*, *HTML=''*)

Creates a child wx.Frame with an OS-managed web browser. The window is modeless, so it can be left open without affecting GSAS-II operations, but will be closed when GSAS-II is ended if a `parent` window is specified.

The web browser is filled with a supplied URL or HTML text. Reuses the previous window unless `newFrame` is set to True.

Parameters

- **parent** (*wx.Frame*) – name of main GSAS-II window (G2frame), if None a toplevel window is created (probably not a good idea).
- **URL** (*str*) – web page to be viewed. This is ignored if `HTML` (below) is specified, but argument `URL` is not optional.
- **size** (*wx.Size*) – initial size of Frame to be created. Defaults to (750,450).
- **newFrame** (*bool*) – When True, a new frame is opened even if the previously-used frame exists. Default is False.
- **HTML** (*str*) – HTML text of a web page to be displayed. If this is specified, the contents of the URL argument is ignored.

Returns
the wx.Frame object used to display the web page

7.2 GSASIImiscGUI: Misc I/O routines

Module with miscellaneous routines for input and output. Many are GUI routines to interact with user.

Includes support for image reading.

7.2.1 GSASIImiscGUI Classes & Routines

Misc routines for GUI-based input and output, including image reading follow.

This module contains quite a bit of older code that could use some attention, or possibly movement into other modules. It was previously called GSASIIIO.py which is why most modules reference it as G2IO.

GSASII.GSASIImiscGUI.**ExportPowder**(*G2frame*, *TreeName*, *fileroot*, *extension*, *hint=''*)

Writes a single powder histogram using the Export routines. This is used in `GSASIIimgGUI.AutoIntFrame()` only.

Parameters

- **G2frame** (*wx.Frame*) – the GSAS-II main data tree window
- **TreeName** (*str*) – the name of the histogram (PWDR …) in the data tree
- **fileroot** (*str*) – name for file to be written, extension ignored
- **extension** (*str*) – extension for file to be written (start with '.'). Must match a powder export routine that has a Writer object.
- **hint** (*str*) – a string that must match the export's format

GSASII.GSASIImiscGUI.**ExportPowderList**(*G2frame*)

Returns a list of extensions supported by *ExportPowder()* along with their descriptions (note that a extension may be repeated but descriptions are unique). This is used in `GSASIIimgGUI.AutoIntFrame()` only.

Parameters
G2frame (*wx.Frame*) – the GSAS-II main data tree window

GSASII.GSASIImiscGUI.**ExportSequential**(*G2frame*, *data*, *obj*, *exporttype*)

Used to export from every phase/dataset in a sequential refinement using a .Writer method for either projects or phases. Prompts to select histograms and for phase exports, which phase(s).

Parameters

- **G2frame** (*wx.Frame*) – the GSAS-II main data tree window
- **data** (*dict*) – the sequential refinement data object
- **obj** (*exporter*) – an exporter object
- **exporttype** (*str*) – indicates the type of export ('project' or 'phase')

GSASII.GSASIImiscGUI.**ExportSequentialFullCIF**(*G2frame*, *seqData*, *Controls*)

Handles access to CIF exporter a bit differently for sequential fits, as this is not accessed via the usual export menus

GSASII.GSASIImiscGUI.**FileDlgFixExt**(*dlg*, *file*)

this is needed to fix a problem in linux wx.FileDialog

GSASII.GSASIImiscGUI.**GetCheckImageFile**(*G2frame*, *treeId*)

Try to locate an image file if the project and image have been moved together. If the image file cannot be found, request the location from the user.

Parameters

- **G2frame** (*wx.Frame*) – main GSAS-II Frame and data object
- **treeId** (*wx.Id*) – Id for the main tree item for the image

Returns

Npix,imagefile,imagetag with (Npix) number of pixels, imagefile, if it exists, or the name of a file that does exist or False if the user presses Cancel and (imagetag) an optional image number

GSASII.GSASIImiscGUI.**GetPowderPeaks**(*fileName*)

Read powder peaks from a file

GSASII.GSASIImiscGUI.**IndexPeakListSave**(*G2frame*, *peaks*)

Save powder peaks from the indexing list

GSASII.GSASIImiscGUI.**LoadImage2Tree**(*imagefile*, *G2frame*, *Comments*, *Data*, *Npix*, *Image*)

Load an image into the tree. Saves the location of the image, as well as the ImageTag (where there is more than one image in the file), if defined.

GSASII.GSASIImiscGUI.**LogCellChanges**(*G2frame*)

Log varied cell parameters into the data tree notebook

GSASII.GSASIImiscGUI.**PeakListSave**(*G2frame*, *file*, *peaks*)

Save powder peaks to a data file

GSASII.GSASIImiscGUI.**ProjFileOpen**(*G2frame*, *showProvenance=True*)

Read a GSAS-II project file and load into the G2 data tree

GSASII.GSASIImiscGUI.**ProjFileSave**(*G2frame*)

Save a GSAS-II project file

GSASII.GSASIImiscGUI.**PutG2Image**(*filename*, *Comments*, *Data*, *Npix*, *image*)

Write an image as a python pickle - might be better as an .edf file?

GSASII.GSASIImiscGUI.**ReadImages**(*G2frame*, *imagefile*)

Read one or more images from a file and put them into the Tree using image importers. Called only in `AutoIntFrame.OnTimerLoop()`.

ToDo: Images are most commonly read in `GSASIIdataGUI.GSASII.OnImportGeneric()` which is called from `GSASIIdataGUI.GSASII.OnImportImage()` it would be good if these routines used a common code core so that changes need to be made in only one place.

Parameters

- **G2frame** (*wx.Frame*) – main GSAS-II Frame and data object.
- **imagefile** (*str*) – name of image file

Returns

a list of the id's of the IMG tree items created

GSASII.GSASIImiscGUI.**SaveIntegration**(*G2frame*, *PickId*, *data*, *Overwrite=False*)

Save image integration results as powder pattern(s)

GSASII.GSASIImiscGUI.**XYsave**(*G2frame*, *XY*, *labelX='X'*, *labelY='Y'*, *names=[]*)

Save XY table data

GSASII.GSASIImiscGUI.**mkParmDictfromTree**(*G2frame*, *sigDict=None*)

Load the GSAS-II refinable parameters from the tree into dict parmDict Updating refined values to those from the last cycle. Optionally compute the s.u. values for the parameters and place them in sigDict.

The actions in the routine are used in a number of places around the GSAS-II code where it would be "cleaner" to use this instead. Perhaps that will happen as code revisions are made. One example of this, `GSASIIfiles.ExportBaseclass.loadParmDict()`.

Parameters

- **G2frame** (*wx.Frame*) – a reference to the main GSAS-II window
- **sigDict** (*dict*) – a Python dict with sigma (s.u.) values for each parameter

Returns

parmDict, a dict with the value for all refined and most unrefined GSAS-II parameters used in the diffraction computations. This parmDict version has only values, as opposed to the version used in some parts of the code that has refinement flags and initial values as well.

GSASII.GSASIImiscGUI.**objectScan**(*data*, *tag*, *indexStack=[]*)

Recursively scan an object looking for unexpected data types. This is used in debug mode to scan .gpx files for objects we did not intend to be there.

GSASII.GSASIImiscGUI.**saveNewPhase**(*G2frame*, *phData*, *newData*, *phlbl*, *msgs*, *orgFilName*)

create a .gpx file from a structure from the BilbaoSite pseudosym site saved in newData

7.3 gltext: draw OpenGL text

Routines that render text on OpenGL without use of GLUT.

Code written by Christian Brugger & Stefan Hacker and distributed under GNU General Public License.

class gltext.**Text**(*text='Text'*, *font=None*, *font_size=8*, *foreground=wx.Colour(-1, -1, -1, 255)*, *centered=False*)

A simple class for using System Fonts to display text in an OpenGL scene. The Text adds a global Cache of already created text elements to TextElement's base functionality so you can save some memory and increase speed

property centered

Display the text centered

draw_text(*position=wx.RealPoint(0.0, 0.0)*, *scale=1.0*, *rotation=0*)

position (wx.RealPoint) - x/y Position to draw in scene scale (float) - Scale rotation (int) - Rotation in degree

Draws the text to the scene

property font

Font of the object

property font_size

Font size

property foreground

Color/Overlay bitmap of the text

getTextElement()

Returns the text element bound to the Text class

getTexture()

Returns the texture of the bound TextElement

getTexture_size()

Returns a texture size tuple

setCentered(*value*, *reinit=True*)

value (bool) - New centered value reinit (bool) - Create a new texture

Sets a new value for 'centered'

setFont(*value*, *reinit=True*)

value (bool) - New Font reinit (bool) - Create a new texture

Sets a new font

setFont_size(*value*, *reinit=True*)

value (bool) - New font size reinit (bool) - Create a new texture

Sets a new font size

setForeground(*value*, *reinit=True*)

value (bool) - New centered value reinit (bool) - Create a new texture

Sets a new value for 'centered'

setText(*value*, *reinit=True*)

value (bool) - New Text reinit (bool) - Create a new texture

Sets a new text

property text

Text of the object

property text_element

TextElement bound to this class

property texture

Texture of bound TextElement

property texture_size

Size of the used texture

class gltext.**TextElement**(*text=''*, *font=None*, *foreground=wx.Colour(-1, -1, -1, 255)*, *centered=False*)

A simple class for using system Fonts to display text in an OpenGL scene

bind()

Increase refcount

property centered

Is text centered

createTexture()

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

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

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

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

deleteTexture()

Deletes the OpenGL texture object

draw_text(*position=wx.RealPoint(0.0, 0.0)*, *scale=1.0*, *rotation=0*)

position (wx.RealPoint) - x/y Position to draw in scene scale (float) - Scale rotation (int) - Rotation in degree

Draws the text to the scene

property font

Font of the object

property foreground

Color of the text

isBound()

Return refcount

property owner_cnt

Owner count

release()

Decrease refcount

property text

Text of the object

property texture

Used texture

property texture_size

Size of the used texture

CHAPTER

EIGHT

GSAS-II GUI COMPONENTS

These modules are used to create different parts of the GSAS-II graphical user interface (GUI).

8.1 GSASIIdataGUI: Main GUI for GSAS-II

Module that defines GUI routines and classes for the main GUI Frame (window) and the main routines that define the GSAS-II tree panel and much of the data editing panel.

8.1.1 GSASIIdataGUI Classes & Routines

Routines for main GUI wx.Frame follow.

GSASII.GSASIIdataGUI.**FindPhaseItem**(*G2frame*)

Finds the Phase item in the tree. If not present it adds one also adding 'Hist/Phase' if config var SeparateHistPhaseTreeItem is set as True.

class GSASII.GSASIIdataGUI.**G2DataWindow**(*parent*)

Create the GSAS-II data window as well as sets up the menus for each window. There will be one instance of this in the GSAS-II app named as `G2frame.dataWindow`.

This creates two panels, where the inner one is the data object in this class, which is scrolled. The outer one (`G2frame.dataWindow.outer`) uses all space in the appropriate part of the window, but defines a sizer at the top and bottom of the window that can be used to place information that will not be scrolled. The inner one is the `G2frame.dataWindow` object.

Note that before any items are to be placed in either of these panels, one should call:

```
G2frame.dataWindow.ClearData()
```

This deletes the contents of the three main sizers used in the panels. Do not delete the sizers for the unscrolled regions at the top and bottom of the outer panel, as they cannot be [easily?] regenerated if deleted.

The sizer for the scrolled panel should be not be reused, though some earlier code may do that.

After the contents of the data window have been created, a call is made to:

```
G2frame.dataWindow.SetDataSize()
```

this ensures that the window's scroll bars are placed properly. Initial GUI creation for the contents of dataWindow is done in *SelectDataTreeItem()*, which is invoked when a selection is made in the data tree selection. This may places items into the dataWindow, but more commonly calls other routines tht call that.

Routines that are called multiple times to redraw the contents of the data window should call *ClearData()* and *SetDataSize()* at the beginning and end of the GUI code, respectively, to clear contents and complete the layout.

When placing a widget in the sizer that has its own scrolling e.g. `GSASIIctrlGUI.GSNoteBook` (anything else?) that one widget should be placed in the scrolledpanel sizer using:

```
mainSizer =  wx.BoxSizer(wx.VERTICAL)
G2frame.dataWindow.SetSizer(mainSizer)
mainSizer.Add(G2frame.<obj>,1,wx.EXPAND)
```

so that it consumes the full size of the panel and so that the NoteBook widget does the scrolling.

For other uses, one will likely place a bunch of widgets and (other [sub-]sizers) into the master sizer. In this case, DO NOT use wx.EXPAND, as this will result in the widget resizing/repositioning as the window resizes. Possible exceptions might be for widgets going into a fixed-size panel that is inside the dataWindow (probably not being done). A call to Sizer.Fit(dataWindow) will do bad things, though a call to SubSizer.Fit(dataWindow.subpanel) could make sense.

Use of the unscrolled top sizer: `G2DataWindow.topBox` provides access to a Horizontal wx.BoxSizer, where GUI objects can be placed. The parent for these objects should be `G2DataWindow.topPanel`. For the unscrolled bottom region of the window, use `G2DataWindow.bottomBox` and `G2DataWindow.bottomPanel` as parent. Sample code:

```
topSizer = G2frame.dataWindow.topBox
parent = G2frame.dataWindow.topPanel
topSizer.Add(wx.StaticText(parent,label='Indexing tools'),0,WACV)
topSizer.Add((-1,-1),1,wx.EXPAND)
topSizer.Add(G2G.HelpButton(parent,helpIndex=G2frame.dataWindow.helpKey))
```

Menus: The same core menu items are used in all menu bars (defined in *`PrefillDataMenu()`* and `PostfillDataMenu`, but different items may be added, depending on what data tree item and in some cases (phases +?) window tab. Menu definitions are performed in `:meth:`_initMenus()`. Menus that are needed at all times in GSAS-II are created there with a call sich as:

```
self.ConstraintMenu = wx.MenuBar()
```

but to reduce the time needed to start GSAS-II initially, most menus are created "on demand". This is done by defining a routine (named here as `_makemenu()`) and the above definition is replaced with:

```
self.ConstraintMenu = _makemenu
```

The code that causes a menubar to be displayed (*`SetDataMenuBar()`*) checks to see if the menubar has been already been created, if so it is displayed, if not the function (the appropriate one of many `_makemenu()` routines) is called. This creates and displays the menu.

Note, if there is a problem, a call like

> wx.CallAfter(G2frame.phaseDisplay.SendSizeEvent)

might be needed. There are some calls to G2frame.dataWindow.SendSizeEvent() or G2frame.dataWindow.outer.SendSizeEvent() that may be doing the same thing.

`ClearData`()

Initializes the contents of the dataWindow panel

`PostfillDataMenu`(*empty=False*)

Add the help menu to the menus associated with data tree items.

`PrefillDataMenu`(*menu*, *empty=False*)

Create the "standard" part of data frame menus & add the dataWindow menu headings This menu duplicates the tree menu, but adds an extra help command for the current data item and a separator.

SetDataSize()

Sizes the contents of the dataWindow panel and sets up for response to change in size of window.

SetStatusWidth()

Make the left-side of the status bar track the size of the data tree

class GSASII.GSASIIdataGUI.**GSASII**(*parent*)

Define the main GSAS-II frame and its associated menu items.

Parameters

parent – reference to parent application

AddSimulatedPowder(*ttArr*, *intArr*, *HistName*, *Lam1*, *Lam2*)

Create a PWDR entry for a computed powder pattern

AddToNotebook(*text*, *typ=None*, *TimeStamp=True*)

Add entry to Notebook tree item

CheckNotebook()

Make sure the data tree has the minimally expected controls.

DoSequentialProjExport(*event*)

Export a sequential project

duplicates part of GSASIIseqGUI.DoSequentialExport

EnableRefineCommand()

Check that phases are connected to histograms - if so then Data/Remove Histogram is enabled

ErrorDialog(*title*, *message*, *parent=None*, *wtype=4*)

Display an error message

ExitMain(*event*)

Called if exit selected or the main window is closed record last position of data & plot windows to config NB: not called if console window closed

ExpandAll(*event*)

Expand all tree items or those of a single type

FillMainMenu(*menubar*, *addhelp=True*)

Define contents of the main GSAS-II menu for the (main) data tree window. For the mac, this is also called for the data item windows as well so that the main menu items are data menu as well.

GetFileList(*fileType*, *skip=None*)

Get list of file names containing a particular string; can skip one of known GSAS-II id param: fileType str: any string within a file name param: skip int:default=None, a GSAS-II assigned id of a data item to skip in collecting the names returns: list of file names from GSAS-II tree returns: str name of file optionally skipped Appears unused, but potentially useful. Note routine of same name in GSASIIpwdGUI; it does not have the skip option

GetHKLFdatafromTree(*HKLFname*)

Returns single crystal data from GSASII tree

Parameters

HKLFname (*str*) – a single crystal histogram name as obtained from GSASIIstruct.GetHistogramNames()

Returns

HKLFdata = single crystal data list of reflections

GetHistogramNames(*hType*)

Returns a list of histogram names found in the GSASII data tree Note routine *GSASIIstrIO.GetHistogramNames()* also exists to get same info from GPX file.

Parameters
hType (*str*) – list of histogram types

Returns
list of histogram names

GetHistogramNamesID(*hType*)

Returns a list of histogram names found in the GSASII data tree and a lookup table of their Id values. Should replace GetHistogramNames since that will not be much faster (and there may be real speed gains from caching the Ids rather than keep searching for them).

N.B routine *GSASIIstrIO.GetHistogramNames()* also exists to get same info, but from GPX file.

Parameters
hType (*str*) – list of histogram types

Returns
list of histogram names and a dict of histogram Ids keyed by histogram name.

GetHistogramTypes()

Returns a list of histogram types found in the GSASII data tree

Returns
list of histogram types

GetPWDRdatafromTree(*PWDRname*)

Returns powder data from GSASII tree

Parameters
PWDRname (*str*) – a powder histogram name as obtained from `GSASIIstruct.GetHistogramNames()`

Returns
PWDRdata = powder data dictionary with Powder data arrays, Limits, Instrument Parameters, Sample Parameters

GetPhaseData()

Returns a dict with defined phases. Note routine *GSASIIstrIO.GetPhaseData()* also exists to get same kind of info from GPX file & put in a list of 9 items.

GetPhaseInfofromTree(*Used=False*)

Get the phase names and their rId values, also the histograms referenced in each phase.

Parameters
Used (*bool*) – if Used is True, only histograms that are referenced in the histogram are returned

Returns

(phaseRIdList, usedHistograms) where

- phaseRIdList is a list of random Id values for each phase
- usedHistograms is a dict where the keys are the phase names and the values for each key are a list of the histogram names used in each phase.

GetPhaseNames()

Returns a list of defined phases. Note routine *GSASIIstrIO.GetPhaseNames()* also exists to get same info from GPX file.

GetPowderIparm(*rd*, *prevIparm*, *lastIparmfile*, *lastdatafile*)

Open and read an instrument parameter file for a data file Returns the list of parameters used in the data tree

Parameters

- **rd** (*obj*) – the raw data (histogram) data object.
- **prevIparm** (*str*) – not used
- **lastIparmfile** (*str*) – Name of last instrument parameter file that was read, or a empty string.
- **lastdatafile** (*str*) – Name of last data file that was read.

Returns

a list of two dicts, the first containing instrument parameters and the second used for TOF lookup tables for profile coeff.

GetTreeItemsList(*item*)

returns a list of all GSAS-II tree items

GetUsedHistogramsAndPhasesfromTree()

Returns all histograms that are found in any phase and any phase that uses a histogram. This also assigns numbers to used phases and histograms by the order they appear in the file. Note routine *GSASIIstrIO.GetUsedHistogramsAndPhases()* also exists to get same info from GPX file.

Returns

(Histograms,Phases)

- Histograms = dictionary of histograms as {name:data,...}
- Phases = dictionary of phases that use histograms

MakeLSParmDict(*seqHist=None*)

Load all parameters used for computation from the tree into a dict of paired values [value, refine flag]. Note that this is different than the parmDict used in the refinement, which only has values.

Note that similar things are done in *GSASIIfiles.ExportBaseclass.loadParmDict()* (from the tree) and *GSASIIstrMain.Refine()* and *GSASIIstrMain.SeqRefine()* (from a GPX file).

Parameters

seqHist (*dict*) – defines a specific histogram to be loaded for a sequential refinement, if None (default) all are loaded. Note: at present this parameter is not used anywhere.

Returns

(parmDict,varyList) where:

- parmDict is a dict with values and refinement flags for each parameter and
- varyList is a list of variables (refined parameters).

MoveTreeItems(*event*)

Move tree items of a single type to the end of the tree

OnAddPhase(*event*)

Add a new, empty phase to the tree. Called by Data/Add Phase menu

OnClusterAnalysis(*event*)

Setsup cluster analysis & make tree entry

OnColMetaTest(*event*)

Test the .par/.*lbls pair for contents

OnDataDelete(*event*)

Delete one or more histograms from data tree. Called by the Data/DeleteData menu

OnDataTreeSelChanged(*event*)

Called when a data tree item is selected. May be called on item deletion as well.

OnDeletePhase(*event*)

Delete one or more phases from the tree. Called by Data/Delete Phase menu. Also delete this phase from Reflection Lists for each PWDR histogram; removes the phase from restraints and deletes any constraints with variables from the phase. If any deleted phase is marked as Used in a histogram, a more rigorous "deep clean" is done and histogram refinement results are cleared, as well as the covariance information and all plots are deleted

OnDeleteSequential(*event*)

Delete any sequential results table. Called by the Data/Delete sequential results menu

OnDummyPowder(*event*)

Called in response to Import/Powder Data/Simulate menu item to create a Dummy powder diffraction data set.

Reads an instrument parameter file and then gets input from the user

OnExportHKL(*event*)

Exports a PWDR reflection list as a text file

OnExportMTZ(*event*)

exports MTZ file from macromoleculat Reflection Lists in multiple histograms

OnExportPDF(*event*)

Save S(Q), G(R),… as selected by user

OnExportPeakList(*event*)

Exports a PWDR peak list as a text file

OnExpressionCalc(*event*)

Compute an arbitrary expression (supplied by user) as well as the (statistical) standard uncertainty on that expression.

Uses the *GSASIIexprGUI.ExpressionDialog* to obtain an expression which is evaluated using the *GSASIIobj.ExpressionObj* capability. Then the derivative of the expression is computed numerically for every parameter in the covariance matrix. Finally the derivative list is used to find the s.u. on the expression using Ted Prince's method.

OnFileBrowse(*event*)

Gets a GSAS-II .gpx project using the GPX browser, in response to the File/"Open Project browser" menu button

OnFileClose(*event*)

Clears the data tree in response to the File/New Project menu button. User is given option to save the project.

OnFileOpen(*event*, *filename=None*, *askSave=True*)

Gets a GSAS-II .gpx project file in response to the File/Open Project menu button

OnFileReopen(*event*)

Creates a dialog box showing previously opened GSAS-II projects & offers to open one called by File/Reopen recent… menu item

OnFileReread(*event*)

reread the current GPX file; no questions asked – no save for development purposes.

OnFileSave(*event*)

Save the current project in response to the File/Save Project menu button

OnFileSaveas(*event*)

Save the current project with a new name in response to the File/Save as menu button. The current project then has this new name

OnGPXtreeItemActivated(*event*)

Called when a tree item is activated

OnGPXtreeItemCollapsed(*event*)

Called when a tree item is collapsed - all children will be collapsed

OnGPXtreeItemDelete(*event*)

Called when a tree item is deleted, inhibit the next tree item selection action

OnGPXtreeItemExpanded(*event*)

Called when a tree item is expanded

OnGPXtreeKeyDown(*event*)

Allows stepping through the tree with the up/down arrow keys

OnImageSum(*event*)

Sum together image data

OnImportGeneric(*reader*, *readerlist*, *label*, *multiple=False*, *usedRanIdList=[]*, *Preview=True*, *load2Tree=False*, *filename=None*)

Used for all imports from GUI, including Phases, datasets, images…

N.B. The code here is largely duplicated in `GSASIIscriptable:import_generic()` so any changes made here need to be duplicated there, too.

Called from *GSASII.OnImportPhase()*, *GSASII.OnImportImage()*, *GSASII.OnImportSfact()*, *GSASII.OnImportPowder()*, *GSASII.OnImportSmallAngle()* and :meth:'GSASII.OnImportReflectometry`

Uses reader_objects subclassed from *GSASIIobj.ImportPhase*, *GSASIIobj.ImportStructFactor*, *GSASIIobj.ImportPowderData*, *GSASIIobj.ImportSmallAngleData* *GSASIIobj.ImportReflectometryData* or *GSASIIobj.ImportImage*. If a specific reader is specified, only that method will be called, but if no reader is specified, every one that is potentially compatible (by file extension) will be tried on the file(s) selected in the Open File dialog.

Parameters

- **reader** (*reader_object*) – This will be a reference to a particular object to be used to read a file or None, if every appropriate reader should be used.
- **readerlist** (*list*) – a list of reader objects appropriate for the current read attempt. At present, this will be either self.ImportPhaseReaderlist, self.ImportSfactReaderlist self.ImportPowderReaderlist or self.ImportImageReaderlist (defined in _init_Imports from the files found in the path), but in theory this list could be tailored. Used only when reader is None.

- **label** (*str*) – string to place on the open file dialog: Open *label* input file
- **multiple** (*bool*) – True if multiple files can be selected in the file dialog. False is default. At present True is used only for reading of powder data.
- **usedRanIdList** (*list*) – an optional list of random Ids that have been used and should not be reused
- **Preview** (*bool*) – indicates if a preview of the file should be shown. Default is True, but set to False for image files which are all binary.
- **load2Tree** (*bool*) – indicates if the file should be loaded into the data tree immediately (used for images only). True only when called from *OnImportImage()*; causes return value to change to a list of True values rather than reader objects.
- **filename** (*str*) – a filename, used only for debugging

Returns

a list of reader objects (rd_list) that were able to read the specified file(s). This list may be empty.

OnImportImage(*event*)

Called in response to an Import/Image/… menu item to read an image from a file. Like all the other imports, dict self.ImportMenuId is used to look up the specific reader item associated with the menu item, which will be None for the last menu item, which is the "guess" option where all appropriate formats will be tried.

A reader object is filled each time an image is read.

N.B. The code here is largely duplicated in `GSASIIscriptable` so any changes made here need to be duplicated there, too.

OnImportPDF(*event*)

Called in response to an Import/PDF G(R) Data/… menu item to read a PDF G(R) data set. dict self.ImportMenuId is used to look up the specific reader item associated with the menu item, which will be None for the last menu item, which is the "guess" option where all appropriate formats will be tried.

N.B. The code here is largely duplicated in `GSASIIscriptable` so any changes made here need to be duplicated there, too.

OnImportPhase(*event*)

Called in response to an Import/Phase/… menu item to read phase information. dict self.ImportMenuId is used to look up the specific reader item associated with the menu item, which will be None for the last menu item, which is the "guess" option where all appropriate formats will be tried.

N.B. The code here is largely duplicated in `GSASIIscriptable` so any changes made here need to be duplicated there, too.

OnImportPowder(*event*)

Called in response to an Import/Powder Data/… menu item to read a powder diffraction data set. dict self.ImportMenuId is used to look up the specific reader item associated with the menu item, which will be None for the last menu item, which is the "guess" option where all appropriate formats will be tried.

Also reads an instrument parameter file for each dataset.

N.B. The code here is largely duplicated in `GSASIIscriptable` so any changes made here need to be duplicated there, too.

OnImportReflectometry(*event*)

Called in response to an Import/Reflectometry Data/… menu item to read a reflectometry data set. dict self.ImportMenuId is used to look up the specific reader item associated with the menu item, which will

be None for the last menu item, which is the "guess" option where all appropriate formats will be tried. Reflectometry data is presumed to be in QIE form for x-rays of neutrons

N.B. This code is not yet duplicated in `GSASIIscriptable`.

OnImportSfact(*event*)

Called in response to an Import/Structure Factor/… menu item to read single crystal datasets. dict self.ImportMenuId is used to look up the specific reader item associated with the menu item, which will be None for the last menu item, which is the "guess" option where all appropriate formats will be tried.

N.B. The code here is largely duplicated in `GSASIIscriptable` so any changes made here need to be duplicated there, too.

OnImportSmallAngle(*event*)

Called in response to an Import/Small Angle Data/… menu item to read a small angle diffraction data set. dict self.ImportMenuId is used to look up the specific reader item associated with the menu item, which will be None for the last menu item, which is the "guess" option where all appropriate formats will be tried. Small angle data is presumed to be as QIE form for either x-rays or neutrons

N.B. This code is not yet duplicated in `GSASIIscriptable`.

OnLeBail(*event*)

Do a 1 cycle LeBail refinement with no other variables; usually done upon initialization of a LeBail refinement either single or sequentially

OnMakePDFs(*event*)

Sets up PDF data structure filled with defaults; if found chemical formula is inserted so a default PDF can be made.

OnNewGSASII(*event*)

Gets a GSAS-II .gpx project file in response to the File/Open new window menu button.

OnPlotDelete(*event*)

Delete one or more plots from plot window. Called by the Data/DeletePlots menu

OnPowderFPA(*event*)

Perform FPA simulation/peak fitting

OnPreferences(*event*)

Edit the GSAS-II configuration variables

OnPwdrSum(*event*)

Sum or Average together powder data(?)

OnReadPowderPeaks(*event*)

Bound to menu Data/Read Powder Peaks

OnRefine(*event*)

Perform a single refinement or a sequential refinement (depending on controls setting) Called from the Calculate/Refine menu.

OnRefinePartials(*event*)

Computes and saves the intensities from each phase for each powder histogram. Do a 0 cycle fit with no variables to pickle intensities for each phase into a file. Not for sequential fits. Sets Controls['PhasePartials'] to a file name to trigger save of info in *GSASIIstrMath.getPowderProfile()* and then clear that.

OnRenameData(*event*)

Renames an existing histogram. Called by Data/Rename Phase menu. Must be used before a histogram is used in a phase.

OnRunAbsorb(*event*)

Run Absorb

OnRunFprime(*event*)

Run Fprime

OnRunPlotXNFF(*evnt*)

Run PlotXNFF

OnSaveMultipleImg(*event*)

Select and save multiple image parameter and mask files

OnSavePartials(*event*)

Saves partials as a csv file

OnSeqRefine(*event*)

Perform a sequential refinement. Called from self.OnRefine (Which is called from the Calculate/Refine menu)

OnShowLSParms(*event*)

Displays a window showing all parameters in the refinement. Called from the Calculate/View LS Parms menu.

This could potentially be sped up by loading only the histogram that is needed for a sequential fit.

OpenPowderInstprm(*instfile*)

Read a GSAS-II (new) instrument parameter file

Parameters

instfile (*str*) – name of instrument parameter file

PreviewFile(*filename*)

utility to confirm we have the right file

ReadPowderInstprm(*instLines*, *bank*, *rd*)

Read contents of a GSAS-II (new) .instprm instrument parameter file similar to G2pwdGUI.OnLoad. Uses *GSASIIfiles.ReadInstprm()* to actually read the file. If instprm file has multiple banks (where each has header #Bank n: …, and bank is supplied as None here, this routine (in GUI) uses a dialog for selection. Note that multibank .instprm files are made by a "Save all profile" command in Instrument Parameters.

Parameters

- **instLines** (*list*) – contents of GSAS-II parameter file as a list of str; N.B. lines can be concatenated with ';'
- **bank** (*int*) – bank number to use when instprm file has values for multiple banks (noted by headers of '#BANK n:…'.). This is ignored for instprm files without those headers. If bank is None with multiple banks, a selection window is shown.
- **rd** (*GSASIIobj.ImportPowder*) – The reader object that will be read from. Sample parameters are placed here.

Returns

Either an instrument parameter dict if OK, or an Error message (str) if read failed

ReadPowderIparm(*instfile*, *bank*, *databanks*, *rd*)

Read a GSAS (old) instrument parameter file

Parameters

- **instfile** (*str*) – name of instrument parameter file
- **bank** (*int*) – the bank number read in the raw data file

- **databanks** (*int*) – the number of banks in the raw data file. If the number of banks in the data and instrument parameter files agree, then the sets of banks are assumed to match up and bank is used to select the instrument parameter file. If not and not TOF, the user is asked to make a selection.
- **rd** (*obj*) – the raw data (histogram) data object. This sets rd.instbank.

ResetPlots()

This reloads the current tree item, often drawing a plot. It also refreshes any plots that have registered a refresh routine (see G2plotNB.RegisterRedrawRoutine) and deletes all plots that have not been refreshed and require one (see G2plotNB.SetNoDelete).

SaveTreeSetting()

Save the current selected tree item by name (since the id will change)

SetDataSize()

this routine is a placeholder until all G2frame.SetDataSize calls are replaced by G2frame.dataWindow.SetDataSize

SetTitleByGPX()

Set the title for the two window frames

StartProject()

Opens a GSAS-II project file & selects the 1st available data set to display (PWDR, HKLF, REFD or SASD)

class SumDialog(*parent*, *title*, *text*, *dataType*, *data*, *dataList*, *Limits=None*)

Used to sum images or powder patterns. Allows user to supply scale factor(s) when summing data.

TODO: clean up the scrolling & resize on this dialog. Can we use the copy down button from CIF? TODO: move it to GSASIIctrlGUI? (BHT) this class should not be in the middle of the (already too complex) GSASIImain class.

OnFilter(*event*)

Read text from filter control and select entries that match.

onChar(*event*)

Respond to keyboard events in the Filter box

clearProject()

Initializes the data tree etc.

init_vars()

initialize default values for GSAS-II "global" variables (saved in main Frame, G2frame)

reloadFromGPX(*rtext=None*, *Rvals={}*)

Deletes current data tree & reloads it from GPX file (after a refinement.) Done after events are completed to avoid crashes.

Parameters

rtext (*str*) – string info from caller to be put in Notebook after reload

testSeqRefineMode()

Returns the list of histograms included in a sequential refinement or an empty list if a standard (non-sequential) refinement. Also sets Menu item status depending on mode

GSASII.GSASIIdataGUI.**GSASIImain**(*application*)

Start up the GSAS-II GUI

GSASII.GSASIIdataGUI.**GUIpatches**()

Misc fixes that only needs to be done when running a GUI

GSASII.GSASIIdataGUI.**GetDisplay**(*pos*)

Gets display number (0=main display) for window position (pos). If pos outside all displays returns None

GSASII.GSASIIdataGUI.**GetGPXtreeDataNames**(*G2frame*, *dataTypes*)

Finds all items in tree that match a 4 character prefix

Parameters

- **G2frame** (*wx.Frame*) – Data tree frame object
- **dataTypes** (*list*) – Contains one or more data tree item types to be matched such as ['IMG '] or ['PWDR','HKLF']

Returns

a list of tree item names for the matching items

GSASII.GSASIIdataGUI.**GetGPXtreeItemId**(*G2frame*, *parentId*, *itemText*)

Find the tree item that matches the text in itemText starting with parentId

Parameters

- **G2frame** (*wx.Frame*) – Data tree frame object
- **parentId** (*wx.TreeItemId*) – tree item to start search with
- **itemText** (*str*) – text for tree item

class GSASII.GSASIIdataGUI.**HKLSortDialog**(*parent*, *data*)

HKL transformation & sort dialog - no merge

Parameters

- **parent** (*wx.Frame*) – reference to parent frame (or None)
- **data** – HKLF data

class GSASII.GSASIIdataGUI.**MergeDialog**(*parent*, *data*)

HKL transformation & merge dialog

Parameters

- **parent** (*wx.Frame*) – reference to parent frame (or None)
- **data** – HKLF data

GSASII.GSASIIdataGUI.**SelectDataTreeItem**(*G2frame*, *item*, *oldFocus=None*)

Called from GSASIIdataGUI.GSASII.OnDataTreeSelChanged() when a item is selected on the tree. Also called from GSASII.OnGPXtreeEndDrag, OnAddPhase – might be better to select item, triggering the the bind to SelectDataTreeItem

Also Called in GSASIIphsGUI.UpdatePhaseData by OnTransform callback.

GSASII.GSASIIdataGUI.**SetDataMenuBar**(*G2frame*, *menu=None*)

Attach the appropriate menu (a wx.MenuBar object) for the selected data tree item to the system's menu bar.

To speed startup of the main window, most menu bars are not created at startup of the program, instead, the menu variable is instead initially defined with a reference to routine that is called to create the menu bar. This routine should overwrite the variable that points to the menu bar (so that the routine is called only once) and it should call GSASII.SetMenuBar() since the name of the created MenuBar object is not available here.

Note that there are some data tree items that do not need their own customized menu bars, for these this routine can be called without a value for the menu argument. This causes the standard, uncustomized, menubar to be used.

GSASII.GSASIIdataGUI.**ShowVersions**()

Show the versions of all GUI-required Python packages, etc.

GSASII.GSASIIdataGUI.**TestOldVersions**()

Test the versions of required Python packages, etc. Returns a non-empty text string if there are problems.

GSASII.GSASIIdataGUI.**UpdateComments**(*G2frame*, *data*)

Place comments into the data window

GSASII.GSASIIdataGUI.**UpdateControls**(*G2frame*, *data*)

Edit overall GSAS-II controls in main Controls data tree entry

GSASII.GSASIIdataGUI.**UpdateNotebook**(*G2frame*, *data*)

Called when the data tree notebook entry is selected. Allows for editing of the text in that tree entry

GSASII.GSASIIdataGUI.**UpdatePWHKPlot**(*G2frame*, *kind*, *item*)

Called when the PWDR & HKLF histogram main tree entry is called. Displays the histogram weight factor, refinement statistics for the histogram and the range of data for a simulation.

Also invokes a plot of the histogram.

GSASII.GSASIIdataGUI.**compareVersions**(*version1*, *version2*)

Compare two version strings ("x", "x.y", "x.y.z") Note that '3.' matches '3.1', and '3.0' matches '3.0.1' but '3.0.0' does not match '3.0.1'

Returns
0 if the versions match, -1 if version1 < version2, or 1 if version1 > version2

GSASII.GSASIIdataGUI.**convVersion**(*version*)

Convert a version string ("x", "x.y", "x.y.z") into a series of ints.

Returns
[i0, i1, i2] where None is used if a value is not specified and 0 is used if a field cannot be parsed.

GSASII.GSASIIdataGUI.**readFromFile**(*reader*)

Define a caption for a file import menu item

GSASII.GSASIIdataGUI.**versionDict = {'badVersionWarn': {'matplotlib': ['3.1', '3.2'], 'numpy': ['1.16.0'], 'wx': ['4.1']}, 'tooNewUntested': {'Python': '3.14', 'wx': '4.3'}, 'tooNewWarn': {'wx': '4.2.5'}, 'tooOld': {'Python': '2.7', 'matplotlib': '2.'}, 'tooOldWarn': {'Python': '3.6', 'matplotlib': '3.2.99', 'wx': '3.99'}}**

Variable versionDict is used to designate versions of packages that should generate warnings or error messages.

- `versionDict['tooOld']` is a dict with module versions that are too old and are known to cause serious errors
- `versionDict['tooOldWarn']` is a dict with module versions that are significantly out of date and should be updated, but will probably function OK.
- `versionDict['badVersionWarn']` is a dict of with lists of package versions that are known to have bugs. One should select an older or newer version of the package.
- `versionDict['tooNewUntested']` is a dict with module versions that have not been tested but there is no reason to expect problems
- `versionDict['tooNewWarn']` is a dict with module versions that have not been tested but there are concerns that problems may occur.

Packages/versions to be avoided

- Python:
- We are no longer supporting Python <=2.7 and <=3.6. Jan. 2023: We will soon start removing code that is specific to Python 2.7.
- A problem has been noted with wx4.0.7.post2 with Python 3.10 that we can't yet duplicate (2/4/22).
- We anticipate that Python 3.10+ will flag code that previously worked fine, because it reports errors where we pass a floating point number to a wxpython routine that expects a int value. We are fixing these as we learn about them.
- wxPython:
- <=2.x.x: while most of GSAS-II has been written to be compatible with older versions of wxpython, we are now testing with version 4.0+ only.
- wxpython 3.0 is pretty similar to 4.0, but we did see some issues with Python 3.x.
- wxpython 4.1 has some serious internal bugs with Python 3.10+ so we recommend 4.2+ for compatibility with newer Python versions.
- 4.2.0 has a problem on MacOS where buttons w/default size are not displayed properly. (see https://github.com/wxWidgets/Phoenix/issues/2319). Worked around (mostly?) in our code.
- 4.2.5 has a problem on MacOS where FileDialog is failing in SaveAs.
- Matplotlib:
  - 1.x: there have been significant API changes since these versions and significant graphics errors will occur.
  - 3.1.x and 3.2.x: these versions have a known bug for plotting 3-D surfaces, such as microstrain vs crystal axes. The plots may appear distorted as the lengths of x, y & z will not be constrained as equal. Preferably use 3.0.x as 3.3.x is not fully tested.
  - between 3.3.x vs 3.6.x there seems to be a change in how 3d graphics are handled; we seem to have this fixed, but not sure how <3.3 will work. Since 3.1 & 3.2 have problems; warn w/mpl <3.3.0
- numpy:
  - 1.16.0: produces .gpx files that are not compatible with older version numpy versions. This is a pretty outmoded version; upgrade.

8.2 *GSASIIseqGUI: Sequential Results GUI*

Module that defines GUI routines and classes for the various sequential result GUI Frames (window). Also defines GUI routines for Cluster Analysis results.

Note that there are seven types of sequential results that GSAS-II can produce and all are displayed/analyzed with the code in this module. They vary by title so that a project can hold one result of each type without a naming collision:

- Rietveld (Title: Sequential results)
- PDF (Title: Sequential PDFfit2 results)
- Peak fit (Title: Sequential peak fit results)
- Small angle (Title: Sequential SASD fit results)
- Reflectometry (Title: Sequential REFD results)

- Image (strain) (Title: Sequential strain fit results)
- Image (calibration) (Title: Sequential image calibration results)

8.2.1 GSASIIseqGUI Classes & Routines

Routines for Sequential Results & Cluster Analysis dataframes follow.

GSASII.GSASIIseqGUI.**ExportSequentialImages**(*G2frame*, *histNames*, *outdir*, *dpi='figure'*)

Used to create plot images as PNG for each fit in the sequential results table. For PWDR entries only. Used in Cinema: D-S to create refinement thumbnails.

Parameters

- **G2frame** (*wx.Frame*) – reference to main GSAS-II frame
- **histNames** (*list*) – a list of the tree name entries corresponding to each row in the table
- **outdir** (*str*) – Name of a directory (path). Files created by this routine are placed here.
- **dpi** (*int*) – dots per inch for the output. Defaults to the screen resolution

Returns

a list of the files created by this routine

GSASII.GSASIIseqGUI.**UpdateSeqResults**(*G2frame*, *data*, *prevSize=None*)

Called when any data tree entry is selected that has 'Sequential' in the name to show results from any sequential analysis.

Parameters

- **G2frame** (*wx.Frame*) – main GSAS-II data tree windows
- **data** (*dict*) – a dictionary containing the following items:
  - 'histNames' - list of histogram names in order as processed by Sequential Refinement
  - 'varyList' - list of variables - identical over all refinements in sequence note that this is the original list of variables, prior to processing constraints.
  - 'variableLabels' – a dict of labels to be applied to each parameter (this is created as an empty dict if not present in data).
  - keyed by histName - dictionaries for all data sets processed, which contains:
    - 'variables'- result[0] from leastsq call
    - 'varyList' - list of variables passed to leastsq call (not same as above)
    - 'sig' - esds for variables
    - 'covMatrix' - covariance matrix from individual refinement
    - 'title' - histogram name; same as dict item name
    - 'newAtomDict' - new atom parameters after shifts applied
    - 'newCellDict' - refined cell parameters after shifts to A0-A5 from Dij terms applied'

8.3 *Phase GUI files*

Routines create the GUI for display of phase information in the data display window when a phase is selected. Phase information is stored in one or more *Phase Tree Item* objects. Note that there are functions that respond to some tabs in the phase GUI in other modules (such as GSASIIddata). This is large enough that it has been split into three files.

8.3.1 GSASIIphsGUI Classes & Routines

Main routine here is *UpdatePhaseData()*, which displays the phase information (called from GSASIIdataGUI:SelectDataTreeItem()).

Other top-level routines are: GetSpGrpfromUser() (called locally only); *FindBondsDraw()* and *FindBondsDrawCell()* (called locally and in GSASIIplot); *SetPhaseWindow()* (called locally and in GSASIIddataGUI and GSASIIrestrGUI, multiple locations) to control scrolling.

Routines for Phase dataframes follow.

class GSASII.GSASIIphsGUI.**AddHatomDialog**(*parent*, *Neigh*, *phase*)

H atom addition dialog. After ShowModal() returns, the results are found in dict self.data, which is accessed using *GetData()*.

Parameters

- **parent** (*wx.Frame*) – reference to parent frame (or None)
- **Neigh** (*dict*) – a dict of atom names with list of atom name, dist pairs for neighboring atoms
- **phase** (*dict*) – a dict containing the phase as defined by *Phase Tree Item*

Draw(*Neigh*, *phase*)

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

GetData()

Returns the values from the dialog

OnOk(*event*)

Called when the OK button is pressed

GSASII.GSASIIphsGUI.**CheckAddHKLF**(*G2frame*, *data*)

GUI actions associated with linking a Phase to a HKLF histogram.

This gets called in two routines named OnHklfAdd (one inside GSASIIphsGUI.UpdatePhaseData() and the other inside GSASIIddataGUI.MakeHistPhaseWin()).

class GSASII.GSASIIphsGUI.**DIFFaXcontrols**(*parent*, *ctrls*, *parms=None*)

Solicit items needed to prepare DIFFaX control.dif file

GSASII.GSASIIphsGUI.**FindBondsDraw**(*data*)

Generally used routine where cell is from data

GSASII.GSASIIphsGUI.**FindBondsDrawCell**(*data*, *cell*)

uses numpy & masks - very fast even for proteins! allows different cell as input from seq. refinements

GSASII.GSASIIphsGUI.**FindCoordination**(*ind*, *data*, *neighborArray*, *coordsArray*, *cmx=0*, *targets=None*)

Find atoms coordinating atom ind, speed-up version. This only searches to atoms already added to the Draw Array, though we might want to search to all atoms in the asymmetric unity (which would mean searching against atomsAll, but would also require a reformat of atom entry to match difference in format between atoms and drawatoms.

GSASII.GSASIIphsGUI.**FindCoordinationByLabel**(*data*)

Map out molecular connectivity by determining the atoms bonded to each atom, by label. The atoms bonded to each atom in the asymmetric unit is determined and returned in a dict. Works best

class GSASII.GSASIIphsGUI.**RotationDialog**(*parent*)

Get Rotate & translate matrix & vector - currently not used needs rethinking - possible use to rotate a group of atoms about some vector/origin + translation

GSASII.GSASIIphsGUI.**SetDrawingDefaults**(*drawingData*)

Add required items into data['drawing'] array if not present. This does not add all the items in SetupDrawingData, but it seems that this is not a problem. Perhaps the two routines could be combined?

GSASII.GSASIIphsGUI.**SetPhaseWindow**(*phasePage*, *mainSizer=None*, *Scroll=0*)

Finish off processing for all items going into a phase notebook page This connects the sizer to the Panel/ScrolledWindow that is assigned as the notebook's page for a tab.

Note that a wx.ScrolledWindow is used for most tab pages, with the exception of Atoms, drawAtoms, G2frame.MapPeaks and G2frame.PawleyRefl, where a wx.Panel is used with a single Grid inside. This allows the grid to handle scrolling.

When a wx.ScrolledWindows is used, scrolling is turned on here. The optional Scroll parameter is used to restore the scroll position to the previous position so that the window can be redrawn without disruption.

class GSASII.GSASIIphsGUI.**SphereEnclosure**(*parent*, *general*, *drawing*, *indx*)

Add atoms within sphere of enclosure to drawing

Parameters

- **parent** (*wx.Frame*) – reference to parent frame (or None)
- **general** – general data (includes drawing data)
- **atoms** – drawing atoms data
- **indx** – list of selected atoms (may be empty)

class GSASII.GSASIIphsGUI.**SymOpDialog**(*parent*, *SGData*, *New=True*, *ForceUnit=False*)

Class to select a symmetry operator

class GSASII.GSASIIphsGUI.**TransformDialog**(*parent*, *phase*, *Trans=array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])*, *Uvec=array([0., 0., 0.])*, *Vvec=array([0., 0., 0.])*, *ifMag=False*, *BNSlatt=''*)

Phase transformation X' = M*(X-U)+V

Parameters

- **parent** (*wx.Frame*) – reference to parent frame (or None)
- **phase** – parent phase data

#NB: commonNames & commonTrans defined in GSASIIdataGUI = G2gd

GSASII.GSASIIphsGUI.**UpdatePhaseData**(*G2frame*, *Item*, *data*)

Create the data display window contents when a phase is clicked on in the main (data tree) window. Called only from `GSASIIdataGUI.SelectDataTreeItem()`, which in turn is called from `GSASIIdataGUI.GSASII.OnDataTreeSelChanged()` when a Phase tree item is selected. This creates all tabs on the page and fills their contents. Routine OnPageChanged is called each time a tab is pressed and updates the contents of the tab's page.

Parameters

- **G2frame** (*wx.frame*) – the main GSAS-II frame object
- **Item** (*wx.TreeItemId*) – the tree item that was selected
- **data** (*dict*) – all the information on the phase in a dictionary

class GSASII.GSASIIphsGUI.**UseMagAtomDialog**(*parent*, *Name*, *Atoms*, *atCodes*, *atMxyz*, *ifMag=True*, *ifOK=False*, *ifDelete=False*)

Get user selected magnetic atoms after cell transformation

GSASII.GSASIIphsGUI.**VoidMap**(*data*, *aMax=1*, *bMax=1*, *cMax=1*, *gridspacing=0.25*, *probeRadius=0.5*, *aMin=0*, *bMin=0*, *cMin=0*)

Compute points where there are no atoms within probeRadius A. All atoms in the Atoms list are considered, provided their occupancy is non-zero.

Parameters

- **data** (*dict*) – Phase data array
- **aMax** (*float*) – Maximum along the *a* direction (fractional units). Defaults to 1.
- **bMax** (*float*) – Maximum along the *b* direction (fractional units). Defaults to 1.
- **cMax** (*float*) – Maximum along the *c* direction (fractional units). Defaults to 1.
- **gridspacing=.25** (*float*) – Approximate spacing of points (fractional units). Defaults to 1.
- **,probeRadius=.5** (*float*)
- **aMin** (*float*) – Minimum along the *a* direction (fractional units). Defaults to 0.
- **bMin** (*float*) – Minimum along the *b* direction (fractional units). Defaults to 0.
- **cMin** (*float*) – Minimum along the *c* direction (fractional units). Defaults to 0.

GSASII.GSASIIphsGUI.**checkPDFfit**(*G2frame*)

Checks to see if PDFfit2 is available and can be imported. PDFfit2 can be installed in a separate Python interpreter (saved in the pdffit2_exec config variable). If this is defined, no attempt is made to check that it actually runs. Otherwise, if diffpy.PDFfit has been installed with conda/pip, it is checked if the install command.

Returns

False if PDFfit2 cannot be run/accessed. True if it appears it can be run.

GSASII.GSASIIphsGUI.**getAtomRadii**(*data*)

Get radii for atoms, using generalData['DisAglCtls']['BondRadii'] to override generalData['BondRadii'] when present. Fix to make sure that all elements in generalData are present in DisAglCtls.

GSASII.GSASIIphsGUI.**getAtomSelections**(*AtmTbl*, *cn=0*, *action='action'*, *includeView=False*, *ask=True*)

get selected atoms from table or ask user if none are selected

Parameters

- **AtmTbl** (*list*) – atom or draw atom table
- **cn** (*int*) – atom name position
- **action** (*str*) – description for prompt, when needed
- **includeView** (*bool*) – if True, the viewpoint is included as an option in the selection dialog

Returns

indx (list) selected atoms from indices in table. If includeView is True, indx can contain index n (where there are n atoms in table). This is indicates the viewpoint.

GSASII.GSASIIphsGUI.**getPawleydRange**(*G2frame*, *data*)

find d-space range in used histograms

GSASII.GSASIIphsGUI.**makeIsoNewPhase**(*phData*, *cell*, *atomList*, *sglbl*, *sgnum*)

create a new phase from a supergroup structure generated by ISOCIF

GSASII.GSASIIphsGUI.**renamePhaseName**(*G2frame*, *data*, *phaseItem*, *generalData*, *newName*)

Called to rename the phase. Updates the tree and items that reference the file name.

GSASII.GSASIIphsGUI.**saveIsoNewPhase**(*G2frame*, *phData*, *newPhase*, *orgFilName*)

save the new phase generated by ISOCIF created in *makeIsoNewPhase()* into a GSAS-II project (.gpx) file

GSASII.GSASIIphsGUI.**showFinalResults**(*G2frame*, *msgs*, *pagelist*, *fileList*)

Show a summary with info from a search of supergroups in OnSuperSearch() (in *UpdatePhaseData()*)

GSASII.GSASIIphsGUI.**updateAddRBorientText**(*G2frame*, *testRBObj*, *Bmat*, *ifSlide=True*)

Update all origin/orientation text on the Add RB panel or on main RB Models page in response to Alt+mouse movement

8.3.2 GSASIIphsGUI2 Classes & Routines

Routines for Phase dataframes follow. Only a few Update routines are here all others are in GSASIIphsGUI.py

GSASII.GSASIIphsGUI2.**UpdateDysnomia**(*G2frame*, *data*)

Present the controls for running Dysnomia

GSASII.GSASIIphsGUI2.**UpdateISODISTORT**(*G2frame*, *data*, *Scroll=0*)

Setup ISODISTORT and present the results. Allow selection of a distortion model for PDFfit or GSAS-II structure refinement as a cif file produced by ISODISTORT. Allows manipulation of distortion mode displacements selection their refinement for this new phase.

GSASII.GSASIIphsGUI2.**UpdateLayerData**(*G2frame*, *data*, *Scroll=0*)

Present the contents of the Phase/Layers tab for stacking fault simulation

GSASII.GSASIIphsGUI2.**UpdateWavesData**(*G2frame*, *data*, *Scroll=0*)

Create the Wave Data tab for modulated structures

8.3.3 GSASIIrmcGUI Classes & Routines

Routines for Phase/RMC follow. Only the UpdateRMC routine is here. All others are in GSASIIphsGUI.py

GSASII.GSASIIrmcGUI.**UpdateRMC**(*G2frame*, *data*)

Present the controls for running fullrmc, RMCProfile or PDFfit

8.4 *GSASIIddataGUI: Phase Diffraction Data GUI*

Module to create the GUI for display of HAP items (where there is an entry for each histogram & phase). This is shown when the Phase "Data" tab is selected or may appear as if in a separate data tree item (see SeparateHistPhaseTreeItem in config.py).

8.4.1 GSASIIddataGUI Classes & Routines

Routines for Data tab in Phase dataframe follows.

GSASII.GSASIIddataGUI.**MakeHistPhaseWin**(*G2frame*)

Display Phase/Data (HAP) info from a Hist/Phase tree item. Used when the HAP info is shown as if a top-level entry in the data tree (see the SeparateHistPhaseTreeItem config option). This code is not used when the HAP info is shown in its original location, the Data tab in the Phase info.

GSASII.GSASIIddataGUI.**UpdateDDData**(*G2frame*, *DData*, *data*, *hist=''*, *Scroll=0*)

Display the Diffraction Data associated with a phase (items where there is a value for each histogram and phase) Used in the Phase/Data tab or the Hist/Phase tree entry

Parameters

- **G2frame** (*wx.frame*) – the main GSAS-II frame object

- **DData** (*wx.ScrolledWindow*) – notebook page to be used for the display
- **data** (*dict*) – all the information on the phase in a dictionary
- **hist** (*str*) – histogram name
- **Scroll** (*int*) – previous scroll position

8.5 *GSASIIElemGUI: GUI to select and delete element lists*

Module to select elements from a periodic table and to delete an element from a list of selected elements.

8.5.1 GSASIIElemGUI Classes & Routines

Routines for Periodic table wx.Frame follow.

class GSASII.GSASIIElemGUI.**DeleteElement**(*parent*, *choice*)

Delete element from selected set widget

ElButton(*name*, *pos*)

Needs a doc string

class GSASII.GSASIIElemGUI.**PickElement**(*parent*, *oneOnly=False*, *ifNone=False*, *ifMag=False*, *ifOrbs=False*, *multiple=False*, *ifVaQ=True*)

Makes periodic table widget for picking element. Modes: oneOnly if True element symbols are provided, otherwise select valence ifNone if True show None button ifMag if True present magnetic scatters only ifOrbs if True present orbital form actors only multiple if True multiple elements can be selected ifVaQ if True show Va & Q buttons

ElButton(*name*, *pos*, *tip*, *color*)

Creates an element button widget

class GSASII.GSASIIElemGUI.**PickElements**(*parent*, *list*)

Makes periodic table widget for picking elements - caller maintains element list

8.6 *GSASIIconstrGUI: Constraint GUI routines*

GUI routines to define constraints and rigid bodies.

8.6.1 GSASIIconstrGUI Classes & Routines

Constraints and rigid bodies GUI routines follow.

GSASII.GSASIIconstrGUI.**CheckAllScalePhaseFractions**(*G2frame*, *refine=True*)

Check if scale factor and all phase fractions are refined without a constraint for all used histograms, if so, offer the user a chance to create a constraint on the sum of phase fractions

Returns

False if refinement should be continued

GSASII.GSASIIconstrGUI.**CheckConstraints**(*G2frame*, *Phases*, *Histograms*, *data*, *newcons=[]*, *reqVaryList=None*, *seqhst=None*, *seqmode='use-all'*)

Load constraints & check them for errors.

N.B. Equivalences based on symmetry (etc.) are generated by running GSASIIstrIO.GetPhaseData().

When reqVaryList is included (see WarnConstraintLimit) then parameters with limits are checked against constraints and a warning is shown.

GSASII.GSASIIconstrGUI.**CheckScalePhaseFractions**(*G2frame*, *hist*, *histograms*, *phases*, *Constraints*)

Check if scale factor and all phase fractions are refined without a constraint for histogram hist, if so, offer the user a chance to create a constraint on the sum of phase fractions

class GSASII.GSASIIconstrGUI.**ConstraintDialog**(*parent*, *title*, *text*, *data*, *separator='*'*, *varname=''*, *varyflag=False*)

Window to edit Constraint values

class GSASII.GSASIIconstrGUI.**DragableRBGrid**(*parent*, *rb*, *onChange=None*)

Simple grid implentation for display of rigid body positions.

Parameters

- **parent** – frame or panel where grid will be placed
- **rb** (*dict*) – dict with atom labels, types and positions
- **onChange** (*function*) – a callback used every time a value in rb is changed.

OnRowMove(*evt*)

called when a row move needs to take place

completeEdits()

complete any outstanding edits

class GSASII.GSASIIconstrGUI.**G2BoolEditor**

Substitute for wx.grid.GridCellBoolEditor except toggles grid items immediately when opened, updates grid & table contents after every item change

ApplyEdit(*row*, *col*, *grid*)

Save the value into the table, and create event. Called after EndEdit(), BeginEdit and onCheckSet.

BeginEdit(*row*, *col*, *grid*)

Prepares the edit control by loading the initial value from the table (toggles it since you would not click on it if you were not planning to change it), buts saves the original, pre-change value. Makes change to table immediately. Saves the info needed to make updates in self.saveVals. Sets the focus.

Clone()

required

Create(*parent*, *id*, *evtHandler*)

Create the editing control (wx.CheckBox) when cell is opened for edit

Destroy()

final cleanup

EndEdit(*row*, *col*, *grid*, *oldVal=None*)

End editing the cell. This is supposed to return None if the value has not changed, but I am not sure that actually works.

Reset()

Reset the value in the control back to its starting value.

SetSize(*rect*)

Set position/size the edit control within the cell's rectangle.

StartingClick()

This seems to be needed for BeginEdit to work properly

onCheckSet(*event*)

Callback used when checkbox is toggled. Makes change to table immediately (creating event)

class GSASII.GSASIIconstrGUI.**RBDataTable**(*rb*, *onChange*)

A Table to support *DragableRBGrid*

GetColLabelValue(*col*) → str

Return the label of the specified column.

GetNumberCols() → int

Must be overridden to return the number of columns in the table.

GetNumberRows() → int

Must be overridden to return the number of rows in the table.

GetRowLabelValue(*row*) → str

Return the label of the specified row.

GetValue(*row*, *col*) → Any

Must be overridden to implement accessing the table values as text.

IsEmptyCell(*row*, *col*) → bool

May be overridden to implement testing for empty cells.

SetValue(*row*, *col*, *value*) → None

Must be overridden to implement setting the table values as text.

GSASII.GSASIIconstrGUI.**ShowIsoDistortCalc**(*G2frame*, *phase=None*)

Compute how NewVar constraints (particularly ISODISTORT mode values) change the dependent variables. The initial constraint values are computed from the settings of the dependent variables. After use the settings can be saved, but if not any changes made here are discarded.

This should work with any type of ISODISTORT mode, as well as New Var constraints created by the user. The window is displayed as a non-modal dialog, so that the plot window remains active.

Called in response to the "Show New Var modes" command in the Compute menu of the "Atoms" or "Draw Atoms" tabs in phases or in Constraints "Edit Constr." menu. Also, the Operations/"Show relationships" menu item in the phase ISODISTORT tab, which should be enabled only when Phase['ISODISTORT'] is defined.

GSASII.GSASIIconstrGUI.**ShowIsoModes**(*G2frame*, *phase*)

Show details about the ISODISTORT mode and the displacements they translate to. Called from the ISODISTORT phase tab only.

GSASII.GSASIIconstrGUI.**TransConstraints**(*G2frame*, *oldPhase*, *newPhase*, *Trans*, *Vec*, *atCodes*)

Add constraints for new magnetic phase created via transformation of old nuclear one NB: A = [G11,G22,G33,2*G12,2*G13,2*G23]

GSASII.GSASIIconstrGUI.**UpdateConstraints**(*G2frame*, *data*, *selectTab=None*, *Clear=False*)

Called when Constraints tree item is selected. Displays the constraints in the data window

GSASII.GSASIIconstrGUI.**UpdateRigidBodies**(*G2frame*, *data*)

Called when Rigid bodies tree item is selected. Displays the rigid bodies in the data window

8.7 GSASIIrestrGUI: Restraint GUI routines

GUI Routines used to define restraints.

8.7.1 GSASIIrestrGUI Classes & Routines

Restraint GUI routines follow.

GSASII.GSASIIrestrGUI.**GetSelectedRows**(*widget*, *lbl='edit'*, *G2frame=None*)

Returns a list of selected rows. Rows can be selected, blocks of cells or individual cells can be selected. The column for selected cells is ignored.

GSASII.GSASIIrestrGUI.**UpdateRestraints**(*G2frame*, *data*, *phaseName*)

Respond to selection of the Restraints item on the data tree

8.8 GSASIIimgGUI: Image GUI

GUI Routines used to control image display and processing

8.8.1 GSASIIimgGUI Classes & Routines

Image GUI routines follow.

class GSASII.GSASIIimgGUI.**AutoIntFrame**(*G2frame*, *PollTime=30.0*)

Creates a wx.Frame window for the Image AutoIntegration. The intent is that this will be used as a non-modal dialog window.

Implements a Start button that morphs into a pause and resume button. This button starts a processing loop that is repeated every `PollTime()` seconds.

Parameters

- **G2frame** (*wx.Frame*) – main GSAS-II frame
- **PollTime** (*float*) – frequency in seconds to repeat calling the processing loop. (Default is 30.0 seconds.)

EnableButtons(*flag*)

Relabels and enable/disables the buttons at window bottom when auto-integration is running

IntegrateImage(*img*, *useTA=None*, *useMask=None*)

Integrates a single image. Ids for created PWDR entries (more than one is possible) are placed in G2frame.IntgOutList

OnPause()

Respond to Pause, changes text on button/Status line, if needed Stops timer self.Pause should already be True

OnTimerLoop(*event*)

A method that is called every `PollTime()` seconds that is used to check for new files and process them. Integrates new images. Also optionally sets up and computes PDF. This is called only after the "Start" button is pressed (then its label reads "Pause").

ResetFromTable(*dist*)

Sets integration parameters based on values from the lookup table

SetSourceDir(*event*)

Use a dialog to get a directory for image files

ShowMatchingFiles(*value*, *invalid=False*, ***kwargs*)

Find and show images in the tree and the image files matching the image file directory (self.params['readdir']) and the image file filter (self.params['filter']) and add this information to the GUI list box

StartLoop()

Prepare to start autointegration timer loop. Save current Image params for use in future integrations also label the window so users understand what is being used

checkPDFprm(*ShowContents=False*)

Read in the PDF (.pdfprm) parameter file and check for problems. If ShowContents is True, a formatted text version of some of the file contents is returned. If errors are found, the return string will contain the string "Error:" at least once.

GSASII.GSASIIimgGUI.**CleanupMasks**(*data*)

If a mask creation is not completed, an empty mask entry is created in the masks array. This cleans them out. It is called when the masks page is first loaded and before saving them or after reading them in. This should also probably be done before they are used for integration.

GSASII.GSASIIimgGUI.**DefineEvaluator**(*dlg*)

Creates a function that provides interpolated values for a given distance value

GSASII.GSASIIimgGUI.**GetImageZ**(*G2frame*, *data*, *newRange=False*)

Gets image & applies dark, background & flat background corrections.

Parameters

- **G2frame** (*wx.Frame*) – main GSAS-II frame
- **data** (*dict*) – Image Controls dictionary

Returns

array sumImg: corrected image for background/dark/flat back

class GSASII.GSASIIimgGUI.**ImgIntLstCtrl**(*parent*, *ID*, *pos=wx.Point(-1, -1)*, *size=(1000, 200)*, *style=0*)

Creates a custom ListCtrl for editing Image Integration parameters

FillList(*parms*)

Places the current parms into the table

OnDouble(*evt*)

respond to a double-click

class GSASII.GSASIIimgGUI.**IntegParmTable**(*parent*, *parms=None*, *IMfileList=None*, *readFileList=None*)

Creates a dialog window with a table of integration parameters. `ShowModal()` will return wx.ID_OK if the process has been successful. In this case, *`DefineEvaluator()`* should be called to obtain a function that creates a dictionary with interpolated parameter values.

ReadFiles(*files*)

Reads a list of .imctrl files or a single .imtbl file

ReadImageParmTable()

Reads possibly edited values from the ListCtrl table and returns a list of values for each column.

GSASII.GSASIIimgGUI.**ReadControls**(*filename*)

read an image controls (.imctrl) file

GSASII.GSASIIimgGUI.**ReadMask**(*filename*)

Read a mask (.immask) file

GSASII.GSASIIimgGUI.**Read_imctrl**(*imctrl_file*)

Read an image control file and record control parms into a dict, with some simple type conversions

GSASII.GSASIIimgGUI.**UpdateImageControls**(*G2frame*, *data*, *masks*, *useTA=None*, *useMask=None*, *IntegrateOnly=False*)

Shows and handles the controls on the "Image Controls" data tree entry

GSASII.GSASIIimgGUI.**UpdateMasks**(*G2frame*, *data*)

Shows and handles the controls on the "Masks" data tree entry

GSASII.GSASIIimgGUI.**UpdateStressStrain**(*G2frame*, *data*)

Shows and handles the controls on the "Stress/Strain" data tree entry

GSASII.GSASIIimgGUI.**displayPhase**(*G2frame*, *phList*, *panel*, *gsizer*, *headers*, *RefreshPlot*)

Fills the scrolled panel with the potential phases to display and their plot options

GSASII.GSASIIimgGUI.**initPhaseOpts**(*phases*)

Make sure that the options for display of partials are all defined

GSASII.GSASIIimgGUI.**selectPhase**(*G2frame*, *calList*, *RefreshPlot*)

Display a dialog with a list of avaliable phases

Parameters

- **G2frame** – main GSAS-II window
- **calList** (*list*) – a list of phases and calibrants that can be selected
- **RefreshPlot** (*function*) – a callable routine that will redisplay the image

GSASII.GSASIIimgGUI.**testColumnMetadata**(*G2frame*)

Test the column-oriented metadata parsing, as implemented at 1-ID, by showing results when using a .par and .lbls pair.

- Select a .par file, if more than one in selected dir.
- Select the .*lbls file, if more than one matching .par file.
- Parse the .lbls file, showing errors if encountered; loop until errors are fixed.
- Search for an image or a line in the .par file and show the results when interpreted

See `GSASIIfiles.readColMetadata()` for more details.

8.9 *GSASIIpwdGUI: Powder Pattern GUI routines*

Used to define GUI controls for the routines that interact with the powder histogram (PWDR) data tree items.

8.9.1 GSASIIpwdGUI Classes & Routines

GUI routines for PWDR datadree subitems follow.

GSASII.GSASIIpwdGUI.**CopyPlotCtrls**(*G2frame*)

Global copy: Copy plot controls from current histogram to others.

GSASII.GSASIIpwdGUI.**CopySelectedHistItems**(*G2frame*)

Global copy: Copy items from current histogram to others.

GSASII.GSASIIpwdGUI.**GetFileBackground**(*G2frame*, *xye*, *background*, *scale=True*)

Select a background file to subtract from PWDR pattern param: xye list [npts,6] of PWDR pattern param: background PWDR file to be used as background param: scale bool:=True if scale mult included in background & apply it returns: list background to subtract

GSASII.GSASIIpwdGUI.**GetFileList**(*G2frame*, *fileType*)

Get list of file names containing a particular string param: fileType str: any string within a file name returns: list of file names from GSAS-II tree Note routine of same name in GSASIIdataGUI; it has a skip option

GSASII.GSASIIpwdGUI.**GetHistsLikeSelected**(*G2frame*)

Get the histograms that match the current selected one: The histogram prefix and data type (PXC etc.), the number of wavelengths and the instrument geometry (Debye-Scherrer etc.) must all match. The current histogram is not included in the list.

Parameters

G2frame (*wx.Frame*) – pointer to main GSAS-II data tree

GSASII.GSASIIpwdGUI.**GetPhasesforHistogram**(*G2frame*, *histoName*)

Returns phases (if any) associated with provided Histogram Returns a list of phase dicts

GSASII.GSASIIpwdGUI.**IsHistogramInAnyPhase**(*G2frame*, *histoName*)

Tests a Histogram to see if it is linked to any phases. Returns the name of the first phase where the histogram is used.

GSASII.GSASIIpwdGUI.**OptimizePDF**(*G2frame*, *data*, *showFit=True*, *maxCycles=5*)

Optimize the PDF to minimize the difference between G(r) and the expected value for low r (-4 pi r #density).

class GSASII.GSASIIpwdGUI.**RDFDialog**(*parent*)

Display controls for generating RDF plot in Background

GSASII.GSASIIpwdGUI.**RefineCell**(*G2frame*)

Refine a unit cell, called by OnRefine

GSASII.GSASIIpwdGUI.**SetCopyNames**(*histName*, *dataType*, *addNames=[]*)

Determine the items in the sample parameters that should be copied, depending on the histogram type and the instrument type.

GSASII.GSASIIpwdGUI.**SetLattice**(*controls*)

impose constraints on lattice constaints and determine the Bravias lattice index (ibrav) as used in cellGUIlist

GSASII.GSASIIpwdGUI.**SetupSampleLabels**(*histName*, *dataType*, *histType*)

Setup a list of labels and number formatting for use in labeling sample parameters.

Parameters

- **histName** (*str*) – Name of histogram, ("PWDR …")
- **dataType** (*str*)

class GSASII.GSASIIpwdGUI.**SubCellsDialog**(*parent*, *title*, *controls*, *SGData*, *items*, *phaseDict*, *ifPick=False*)

Display magnetic subcell space group information from selection in Unit Cells table of results from k-SUBGROUPSMAG

GSASII.GSASIIpwdGUI.**TestAtoms**(*phase*, *magAtms*, *SGData*, *Uvec*, *Trans*, *maxequiv=100*, *maximal=False*)

Tests atoms for substructure equivalents param: phase GSAS-II phase object param: magAtms list: atom objects param: SGData dict: GSAS-II space group object param: Uvec array: Translation U vector param: Trans array: Transformation matrix params: maxequiv int:maximum number of atoms with moments to consider param: maximal bool:=True if maximal subgroups only are allowed returns: unique atoms (if any)

GSASII.GSASIIpwdGUI.**TestMagAtoms**(*phase*, *magAtms*, *SGData*, *Uvec*, *Trans*, *allmom*, *maxequiv=100*, *maximal=False*)

Tests substructure magnetic atoms for magnetic site symmetry param: phase GSAS-II phase object param: magAtms list:magnetic atom objects param: SGData dict: GSAS-II space group object param: Uvec array: Translation U vector param: Trans array: Transformation matrix param: allmom bool: =True if all atoms must have moments

allowed params: maxequiv int:maximum number of atoms with moments to consider param: maximal bool:=True if maximal subgroups only are allowed returns: unique magnetic atoms (if any)

GSASII.GSASIIpwdGUI.**UpdateBackground**(*G2frame*, *data*)

respond to selection of PWDR background data tree item.

GSASII.GSASIIpwdGUI.**UpdateIndexPeaksGrid**(*G2frame*, *data*)

respond to selection of PWDR Index Peak List data tree item.

GSASII.GSASIIpwdGUI.**UpdateInstrumentGrid**(*G2frame*, *data*)

respond to selection of PWDR/SASD/REFD/HKLF Instrument Parameters data tree item.

GSASII.GSASIIpwdGUI.**UpdateLimitsGrid**(*G2frame*, *data*, *datatype*)

respond to selection of PWDR Limits data tree item. Allows setting of limits and excluded regions in a PWDR data set

GSASII.GSASIIpwdGUI.**UpdateModelsGrid**(*G2frame*, *data*)

respond to selection of SASD Models data tree item.

GSASII.GSASIIpwdGUI.**UpdatePDFGrid**(*G2frame*, *data*)

respond to selection of PWDR PDF data tree item.

GSASII.GSASIIpwdGUI.**UpdatePeakGrid**(*G2frame*, *data*)

respond to selection of PWDR powder peaks data tree item.

GSASII.GSASIIpwdGUI.**UpdateREFDModelsGrid**(*G2frame*, *data*)

respond to selection of REFD Models data tree item.

GSASII.GSASIIpwdGUI.**UpdateReflectionGrid**(*G2frame*, *data*, *HKLF=False*, *Name=''*)

respond to selection of PWDR or HKLF Reflections data tree item by displaying a table of reflections in the data window.

Note that this is used for Single Xtal even though in pwdGUI.

GSASII.GSASIIpwdGUI.**UpdateSampleGrid**(*G2frame*, *data*)

respond to selection of PWDR/SASD Sample Parameters data tree item.

GSASII.GSASIIpwdGUI.**UpdateSubstanceGrid**(*G2frame*, *data*)

respond to selection of SASD/REFD Substance data tree item.

GSASII.GSASIIpwdGUI.**UpdateUnitCellsGrid**(*G2frame*, *data*, *callSeaResSelected=False*, *New=False*, *showUse=False*)

respond to selection of PWDR Unit Cells data tree item.

Parameters

- **G2frame** (*wx.Frame*) – Main GSAS-II window
- **data** (*dict*) – contents of "Unit Cells List" data tree item
- **callSeaResSelected** (*bool*) – when True, selects first entry in UnitCellsTable search results table
- **New** (*bool*)
- **showUse** (*bool*) – when showUse is False (default) the Show flag is cleared in all search tables. When True, and there is a True value for Show, the flag is set for that in the grid and the row is scrolled into view. This is currently implemented for indexing (Cell Search Results) only.

GSASII.GSASIIpwdGUI.**addAutoBack**(*G2frame*, *data*, *xydata*)

Create a new histogram for the computed auto background and place as the fixed background histogram

class GSASII.GSASIIpwdGUI.**autoBackground**(*G2frame*, **args*, ***kwargs*)

Create a file selection widget for setting background with pybaselines, as requested by James Feng.

Parameters

G2frame (*wx.Frame*) – reference to the main GSAS-II frame.

GSASII.GSASIIpwdGUI.**computePDF**(*G2frame*, *data*)

Calls GSASIIpwd.CalcPDF() to compute the PDF and put into the data tree array. Called from OnComputePDF and OnComputeAllPDF and OnComputeAllPDF in GSASIIimgGUI.py

8.10 *GSASIIexprGUI: Expression Handling*

This module defines a class for defining an expression in terms of values in a parameter dictionary via a wx.Dialog. The dialog creates a GSASIIexprGUI.GSASII.ExpressionObj which is used to evaluate the expression against a supplied parameter dictionary.

The expression is parsed to find variables used in the expression and then the user is asked to assign parameters from the dictionary to each variable.

Default expressions are read from file DefaultExpressions.txt using GSASIIpath.LoadConfigFile().

8.10.1 GSASIIexprGUI Classes & Routines

Routines for users to input Python expressions used within GSAS-II computations follow.

class GSASII.GSASIIexprGUI.**AngleDialog**(*parent*, *Phases*, *parmDict*, *exprObj=None*, *header='Select an angle for table'*, *wintitle='Select angle'*, *VarLabel=None*, *depVarDict=None*, *ExtraButton=None*, *usedVars=[]*)

A wx.Dialog that allows a user to select a bond angle to be evaluated. What needs to be done here? Need phase info for atom 0. Select phase 1. Select 1st atom from dialog 2. Find neighbors & select two from dialog 3. Set up angle equation & save it - has to look like result from Show in above ExpressionDialog Use existing angle & esd calculate routines

Draw()

paints the angle dialog window

class GSASII.GSASIIexprGUI.**BondDialog**(*parent*, *Phases*, *parmDict*, *exprObj=None*, *header='Select a bond for table'*, *wintitle='Select bond'*, *VarLabel=None*, *depVarDict=None*, *ExtraButton=None*, *usedVars=[]*)

A wx.Dialog that allows a user to select a bond length to be evaluated. What needs to be done here? Need phase info for atoms 0. Select phase 1. Select 1st atom from dialog 2. Find neighbors & select one from dialog 3. Set up distance equation & save it - has to look like result from Show in above ExpressionDialog Use existing bond & esd calculate routines

Draw()

paints the distance dialog window

class GSASII.GSASIIexprGUI.**ExpressionDialog**(*parent*, *parmDict*, *exprObj=None*, *header='Enter restraint expression here'*, *wintitle='Expression Editor'*, *fit=True*, *VarLabel=None*, *depVarDict=None*, *ExtraButton=None*, *usedVars=[]*, *wildCard=True*)

A wx.Dialog that allows a user to input an arbitrary expression to be evaluated and possibly minimized.

To do this, the user assigns a new (free) or existing GSAS-II parameter to each parameter label used in the expression. The free parameters can optionally be designated to be refined. For example, is an expression is used such as:

```
'A*np.exp(-B/C)'
```

then A, B and C can each be assigned as Free parameter with a user-selected value or to any existing GSAS-II variable in the parameter dictionary. As the expression is entered it is checked for validity.

After the *ExpressionDialog* object is created, use *Show()* to run it and obtain a `GSASIIobj.ExpressionObj` object with the user input.

Parameters

- **parent** (*wx.Frame*) – The parent of the Dialog. Can be None, but better is to provide the name of the Frame where the dialog is called.
- **parmDict** (*dict*) – a dict with defined parameters and their values. Each value may be a list with parameter values and a refine flag or may just contain the parameter value (non-float/int values in dict are ignored)
- **exprObj** – a `GSASIIobj.ExpressionObj` object with an expression and label assignments or None (default)
- **wintitle** (*str*) – String placed on title bar of dialog; defaults to "Expression Editor"
- **header** (*str*) – String placed at top of dialog to tell the user what they will do here; default is "Enter restraint expression here"
- **fit** (*bool*) – determines if the expression will be used in fitting (default=True). If set to False, refinement flags are not shown and Free parameters are not offered as an assignment option.
- **VarLabel** (*str*) – an optional variable label to include before the expression input. Ignored if None (default)
- **depVarDict** (*list*) – a dict of choices for the dependent variable to be fitted to the expression and their values. Ignored if None (default).
- **ExtraButton** (*list*) – a list with two terms that define [0]: the label for an extra button and [1] the callback routine to be used when the button is pressed. The button will only be enabled when the OK button can be used (meaning the equation/expression is valid). The default is None, meaning this will not be used.
- **usedVars** (*list*) – defines a list of previously used variable names. These names will not be reused as defaults for new free variables. (The default is an empty list).
- **wildCard** (*bool*) – If True (default), histogram names are converted to wildcard values, as is appropriate for the sequential refinement table

CheckVars()

Check that appropriate variables are defined for each symbol used in `self.expr`

Returns

a text error message or None if all needed input is present

GetDepVar()

Returns the name of the dependent variable, when depVarDict is used.

OnChar(*event*)

Called as each character is entered. Cancels any running timer and starts a new one. The timer causes a check of syntax after 2 seconds without input. Disables the OK button until a validity check is complete.

OnChoice(*event*)

Respond to a selection of a variable type for a label in an expression

OnChoiceButton(*event*)

Respond to a selection of a variable type for a label in an expression

OnDepChoice(*event*)

Respond to a selection of a variable type for a label in an expression

OnValidate(*event*)

Respond to a press of the Validate button or when a variable is associated with a label (in *OnChoice()*)

Repaint(*exprObj*)

Redisplay the variables and continue the validation

RestartTimer()

Cancels any running timer and starts a new one. The timer causes a check of syntax after 2 seconds unless there is further input. Disables the OK button until a validity check is complete.

SelectG2var(*sel*, *var*, *parmList*)

Offer a selection of a GSAS-II variable.

Parameters

sel (*int*) – Determines the type of variable to be selected. where 1 is used for Phase variables, and 2 for Histogram/Phase vars, 3 for Histogram vars and 4 for Global vars.

Returns

a variable name or None (if Cancel is pressed)

Show(*mode=True*)

Call to use the dialog after it is created.

Returns

None (On Cancel) or a new `ExpressionObj`

depVarDict

dict for dependent variables

dependentVar

name for dependent variable selection, when depVarDict is specified

expr

Expression as a text string

exprVarLst

A list containing the variables utilized in the current expression. Placed into a `GSASIIobj.ExpressionObj` object when the dialog is closed with "OK", saving any changes.

parmDict

A copy of the G2 parameter dict (parmDict) except only numerical values are included and only the value (not the vary flag, if present) is included.

setEvalResult(*msg*)

Show a string in the expression result area

showError(*msg1*, *msg2=''*, *msg3=''*)

Show an error message of 1 to 3 sections. The second section is shown in an equally-spaced font.

Parameters

- **msg1** (*str*) – msg1 is shown in a the standard font
- **msg2** (*str*) – msg2 is shown in a equally-spaced (wx.MODERN) font
- **msg3** (*str*) – msg3 is shown in a the standard font

usedVars
variable names that have been used and should not be reused by default

varName
Name assigned to each variable

varRefflag
Refinement flag for a variable (Free parameters only)

varSelect
A dict that shows the variable type for each label found in the expression.

- If the value is None or is not defined, the value is not assigned.
- If the value is 0, then the varible is "free" – a new refineable parameter.
- Values above 1 determine what variables will be shown when the option is selected.

varValue
Value for a variable (Free parameters only)

GSASII.GSASIIexprGUI.**IndexParmDict**(*parmDict*, *wildcard*)
Separate the parameters in parmDict into list of keys by parameter type.

Parameters

- **parmDict** (*dict*) – a dict with GSAS-II parameters
- **wildcard** (*bool*) – True if wildcard versions of parameters should be generated and added to the lists

Returns
a dict of lists where key 1 is a list of phase parameters, 2 is histogram/phase parms, 3 is histogram parms and 4 are global parameters

GSASII.GSASIIexprGUI.**LoadDefaultExpressions**()
Read a configuration file with default expressions from all files named DefaultExpressions.txt found in the path. Duplicates are removed and expressions are sorted alphabetically

8.11 *GSASIIfpaGUI: Fundamental Parameters Routines*

This module contains GUI routines to accept Fundamental Parameters Approach (FPA) input used to run the NIST XRD Fundamental Parameters Code, computes a set of peaks with that code and fits profile terms to the peaks. Also allows for plotting the convolutors generated by that code.

8.11.1 GSASIIfpaGUI Classes & Routines

NIST XRD Fundamental Parameters interface routines follow:

GSASII.GSASIIfpaGUI.**BBPSDDetector = [('SiPSD_th2_angular_range', 3.0, 'Angular range observed by PSD (degrees 2Theta)')]**
Additional FPA dict entries used in *FillParmSizer()* needed for Bragg Brentano instruments with linear (1-D) Si PSD detectors.

GSASII.GSASIIfpaGUI.**BBPointDetector = [('receiving_slit_width', 0.2, 'Width of receiving slit (mm)')]**

Additional FPA dict entries used in *FillParmSizer()* needed for Bragg Brentano instruments with point detectors.

GSASII.GSASIIfpaGUI.**BraggBrentanoParms = [('divergence', 0.5, 'Bragg-Brentano divergence angle (degrees)'), ('soller_angle', 2.0, 'Soller slit axial divergence (degrees)'), ('Rs', 220, 'Diffractometer radius (mm)'), ('filament_length', 12.0, 'X-ray tube line focus length (mm)'), ('sample_length', 12.0, 'Illuminated sample length in axial direction (mm)'), ('receiving_slit_length', 12.0, 'Length of receiving slit in axial direction (mm)'), ('LAC_cm', 0.0, 'Linear absorption coef. adjusted for packing density (cm-1)'), ('sample_thickness', 1.0, 'Depth of sample (mm)'), ('convolution_steps', 8, 'Number of Fourier-space bins per two-theta step'), ('source_width', 0.04, 'Tube filament width, in projection at takeoff angle (mm)'), ('tube-tails_L-tail', -1.0, 'Left-side tube tails width, in projection (mm)'), ('tube-tails_R-tail', 1.0, 'Right-side tube tails width, in projection (mm)'), ('tube-tails_rel-I', 0.001, 'Tube tails fractional intensity (no units)')]**

FPA dict entries used in *FillParmSizer()*. Tuple contains a dict key, a default value and a description. These are the parameters needed for all Bragg Brentano instruments

GSASII.GSASIIfpaGUI.**DetMode = 'BBpoint'**

The type of detector, either 'BBpoint' for Bragg-Brentano point detector or or 'BBPSD' (linear) position sensitive detector

GSASII.GSASIIfpaGUI.**FillParmSizer**()

Create a list of input items for the parameter section of the input window, sets default values when not set and displays them in the scrolledpanel prmPnl.

GSASII.GSASIIfpaGUI.**IBmono = False**

set to True if an incident beam monochromator is in use

GSASII.GSASIIfpaGUI.**IBmonoParms = [('src_mono_mm', 119, 'Distance from xray line source to monochromator crystal (mm)'), ('focus_mono_mm', 217, 'Distance from monochromator crystal to focus slit (mm)'), ('passband_mistune', -0.145, 'Offset from the tuning of the IBM to the center of the reference line of the spectrum, in units of its bandwidth'), ('mono_src_proj_mn', 51, 'Projection width of line-focus xray source on the monochromator along beam direction (microns), sets bandwidth'), ('passband_shoulder', 0.087, 'Width of transition region from flat top to tails, in units of the bandwidth'), ('two_theta_mono', 27.27, 'The full diffraction angle of the IBM crystal, e.g. double 2theta-Bragg for the mono (deg)'), ('mono_slit_attenuation', 0.03, 'Attenuation of Cu K alpha2 lines due to focal slit')]**

Additional FPA dict entries used in *FillParmSizer()*, needed for Incident Beam Monochromator

GSASII.GSASIIfpaGUI.**MakeSimSizer**(*G2frame*, *dlg*)

Create a GUI to get simulation with parameters for Fundamental Parameters fitting.

Parameters

dlg (*wx.Window*) – Frame or Dialog where GUI will appear

Returns

a sizer with the GUI controls

GSASII.GSASIIfpaGUI.**MakeTopasFPASizer**(*G2frame*, *FPdlg*, *SetButtonStatus*)

Create a GUI with parameters for the NIST XRD Fundamental Parameters Code. Parameter input is modeled after Topas input parameters.

Parameters

- **G2frame** (*wx.Frame*) – main GSAS-II window
- **FPdlg** (*wx.Window*) – Frame or Dialog where GUI will appear
- **SetButtonStatus** – a callback function to call to see what buttons in this windows can be enabled. Called with done=True to trigger closing the parent window as well.

Returns
a sizer with the GUI controls

GSASII.GSASIIfpaGUI.**NISTparms = {}**

Parameters in a nested dict, with an entry for each concolutor. Entries in those dicts have values in SI units (of course). NISTparms can be can be input directly or can be from created from *parmDict* by *XferFPAsettings()*

GSASII.GSASIIfpaGUI.**SetCu2Wave**()

Set the parameters to the two-line Cu K alpha 1+2 spectrum

GSASII.GSASIIfpaGUI.**SetCu6wave**()

Set the emission parameters to the NIST six-line Cu K alpha spectrum

GSASII.GSASIIfpaGUI.**SetMonoWave**()

Eliminates the short-wavelength line from the six-line Cu K alpha spectrum when incident beam mono; resets it to 6 if no mono

GSASII.GSASIIfpaGUI.**XferFPAsettings**(*InpParms*)

convert Topas-type parameters to SI units for NIST and place in a dict sorted according to use in each convoluter

Parameters
InpParms (*dict*) – a dict with Topas-like parameters, as set in *MakeTopasFPASizer()*

Returns
a nested dict with global parameters and those for each convolution

GSASII.GSASIIfpaGUI.**doFPAcalc**(*NISTpk*, *ttArr*, *twotheta*, *calcwid*, *step*)

Compute a single peak using a NIST profile object

Parameters

- **NISTpk** (*object*) – a peak profile computational object from the NIST XRD Fundamental Parameters Code, typically established from a call to SetupFPAcalc()
- **ttArr** (*np.Array*) – an evenly-spaced grid of two-theta points (degrees)
- **twotheta** (*float*) – nominal center of peak (degrees)
- **calcwid** (*float*) – width to perform convolution (degrees)
- **step** (*float*) – step size

GSASII.GSASIIfpaGUI.**parmDict = {'int': {0: 0.58384351, 1: 0.2284605, 2: 0.11258773, 3: 0.07077796, 4: 0.0043303, 5: 0.00208613}, 'lwidth': {0: 0.436, 1: 0.487, 2: 0.63, 3: 0.558, 4: 2.93, 5: 2.93}, 'wave': {0: 1.5405925, 1: 1.5443873, 2: 1.5446782, 3: 1.5410769, 4: 1.53471, 5: 1.53382}}**

Parameter dict used for reading Topas-style values. These are converted to SI units and placed into *NISTparms*

GSASII.GSASIIfpaGUI.**setupFPAcalc**()

Create a peak profile object using the NIST XRD Fundamental Parameters Code.

Returns
a profile object that can provide information on each convolution or compute the composite peak shape.

GSASII.GSASIIfpaGUI.**simParms = {}**

Parameters to set range for pattern simulation

GSASII.GSASIIfpaGUI.**writeNIST**(*filename*)

Write the NIST FPA terms into a JSON-like file that can be reloaded in _onReadFPA

8.12 *fprime: compute f' & f"*

This module contains GUI routines to calculate and plot real and resonant X-ray scattering factors to 250keV, based on Fortran program of Cromer & Liberman corrected for Kissel & Pratt energy term; Jensen term not included (D. T. Cromer and D. A. Liberman, Acta Cryst. (1981). A37, 267-268.)

8.12.1 fprime Classes & Routines

main Fprime routines Copyright: 2008, Robert B. Von Dreele (Argonne National Laboratory)

class GSASII.fprime.**Fprime**(*parent*)

Creates a frame where input for absorption calculation is supplied

CalcFPPS()

generate set of f' & f" curves for selected elements does constant delta-lambda/lambda steps over defined range

OnABOUTItems0Menu(*event*)

Displays the About window

UpDateFPlot(*Wave*, *rePlot=True*)

Plot f' & f" vs wavelength 0.05-3.0A

8.13 *Absorb: Compute X-ray Absorption*

This module contains GUI routines to calculate X-ray absorption factors to 250keV for cylindrical powder samples with a specified chemical composition and density. Uses same approach as in `fprime` (D. T. Cromer and D. A. Liberman, Acta Cryst. (1981). A37, 267-268.)

8.13.1 Absorb Classes & Routines

main Absorb routines Copyright: 2009, Robert B. Von Dreele (Argonne National Laboratory)

class GSASII.Absorb.**Absorb**(*parent*)

Creates a frame where input for absorption calculation is supplied

CalcFPPS()

generate f" curves for selected elements does constant delta-lambda/lambda steps over defined range

OnABOUTItems0Menu(*event*)

Displays the About window

SetSize(*x*, *y*, *width*, *height*, *sizeFlags=SIZE_AUTO*) → None

SetSize(*rect*) → None

SetSize(*size*) → None

SetSize(*width*, *height*) → None

Sets the size of the window in pixels.

UpDateAbsPlot(*Wave*, *rePlot=True*)

Plot mu vs wavelength 0.05-3.0A

8.14 *PlotXNFF: Show Form Factors*

8.14.1 *PlotXNFF: Check x-ray & neutron form factor computation*

Use this to check form factors used in x-ray & neutron scattering

class GSASII.PlotXNFF.**PlotXNFF**(*parent*)

CHAPTER

NINE

GSAS-II DATA STORAGE MODULES

These modules are used to provide a place to find constants that are used in various locations.

9.1 GSASIIdata: Ramachandran Parameters

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

9.2 ElementTable: Periodic Table Data

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

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

9.3 FormFactors: Scattering Data

Contains atomic scattering factors from "New Analytical Scattering Factor Functions for Free Atoms and Ions for Free Atoms and Ions", D. Waasmaier & A. Kirfel, *Acta Cryst.* **(1995).** A51, 416-413.

Also, tabulated coefficients for calculation of Compton Cross Section as a function of sin(theta)/lambda from "Analytic Approximations to Incoherently Scattered X-Ray Intensities", H. H. M. Balyuzi, *Acta Cryst.* **(1975).** A31, 600.

9.4 ImageCalibrants: Calibration Standards

GSASII powder calibrants in dictionary `ImageCalibrants.Calibrants` containing substances commonly used for powder calibrations for image data.

Each entry in `ImageCalibrants` consists of:

```
 'key':([Bravais num,],[space group,],[(a,b,c,alpha,beta,gamma),],no. lines skipped,
↪(dmin,pixLimit,cutOff),(absent list))

* See below for Bravais num assignments.
* The space group may be an empty string.
* The absent list is optional; it gives indices of lines that have no intensity␣
↪despite being allowed - see the Si example below; counting begins at zero
```

As an example:

```
'LaB6  SRM660a':([2,],['',][(4.1569162,4.1569162,4.1569162,90,90,90),],0,(1.0,10,10.
↪)),
```

For calibrants that are mixtures, the "Bravais num" and "(a,b,…)" values are repeated, as in this case:

```
'LaB6 & CeO2':([2,0],['','']  [(4.1569,4.1569,4.1569,90,90,90),(5.4117,5.4117,5.4117,
↪90,90,90)], 0, (1.0,2,1.)),
```

Note that Si has reflections (the 4th, 11th,…) that are not extinct by symmetry but still have zero intensity. These are supplied in the final list:

```
'Si':([0,],['F d 3 m'],[(5.4311946,5.4311946,5.4311946,90,90,90),],0,(1.,10,10.),(3,
↪10,13,20,23,26,33,35,40,43)),
```

Note, the Bravais numbers are:

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

9.4.1 User-Defined Calibrants

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

```
Calibrants={
  'LaB6 skip 2 lines':([2,],['',],[(4.1569162,4.1569162,4.1569162,90,90,90),],2,(1.0,
↪10,10),()),
}
```

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

9.5 *atmdata: Table of atomic data*

The entries here are:

XrayFF: a dict of form factor coefficients

AtmSize: atom Sizes, bond radii, angle radii, H-bond radii

AtmBlens: atom masses & neutron scattering length (b,b'), sig(incoh) @ 1A

MagFF: neutron magnetic form factor coeff: M for j<0> & N for j<2>

Sources:

Exponential scattering factor curve coeficients, Cromer and Waber(1971) Int. Tables Vol. IV. Delta f' and delta f" terms calcd via D.T. Cromer & D.A. Liberman (1981), Acta Cryst. A37, 267-268. Atomic weights from CRC 56th Edition

Electron scattering factor curve coefficients from International Tables for Crystallography, Vol. C, Ch 4. Sect. 4.3.2.

Neutron scattering lengths & abs. cross sections from H. Rauch & W. Waschowski, Neutron Data Booklet, 2003. More revent values cited in table AtmBlens below. X-ray <j0> & <j2> coeff. from Intl. Tables for Cryst, Vol. C 5-d <j0> & <j2> from Kobayashi K, Nagao T, Ito M. Acta Crystallogr A67, 473-480 (2011)

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

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

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

Orbital form factors from fitting Table 2.2D in International Tables for x-ray Crystallography Vol IV by Cromer & Waber (1974)

9.6 *defaultIparms: Table of instrument parameters*

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

For a new entry:

Append a useful name to defaultIparms_lbl.

Append the list of lines to defaultIparms.

defaultIparm_lbl: defines a list of labels.

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

CHAPTER

TEN

GSAS-II STRUCTURE SUBMODULES

These modules are used to perform structure-related (mostly intensity) computations that are needed for refinement and related computations.

10.1 GSASIIstrMain: main structure routine

10.1.1 GSASIIstrMain Classes & Routines

`GSASIIstrMain` routines, used for refinement, or refinement-related computations (e.g. distance & angle computations), are found below.

These routines are most commonly called when working from a .gpx file, but may sometimes be called from the GUI. These routines expect that all needed input will have been read from the file/tree and are passed to thes routines here as arguments. The data tree is never accessed directly in this module and no GUI modules should be imported here.

`GSASII.GSASIIstrMain.`**`AllPrmDerivs`**(*Controls*, *Histograms*, *Phases*, *restraintDict*, *rigidbodyDict*, *parmDict*, *varyList*, *calcControls*, *pawleyLookup*, *symHold*, *dlg=None*)

Computes the derivative of the fitting function (total Chi**2) with respect to every parameter in the parameter dictionary (parmDict) by applying shift below the parameter value as well as above.

Returns

a dict with the derivatives keyed by variable number. Derivatives are a list with three values: evaluated over 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.

`GSASII.GSASIIstrMain.`**`BestPlane`**(*PlaneData*)

Needs a doc string

`GSASII.GSASIIstrMain.`**`CheckLeBail`**(*Phases*)

Check if there is a LeBail extraction in any histogram

Returns

True if there is at least one LeBail flag turned on, False otherwise

`GSASII.GSASIIstrMain.`**`DisAglTor`**(*DATData*)

Needs a doc string

`GSASII.GSASIIstrMain.`**`DoLeBail`**(*GPXfile*, *dlg=None*, *cycles=10*, *refPlotUpdate=None*, *seqList=None*)

Fit LeBail intensities without changes to any other refined parameters. This is a stripped-down version of `Refine()` that does not perform any refinement cycles

Parameters

- **`GPXfile`** (*str*) – G2 .gpx file name

- **dlg** (*wx.ProgressDialog*) – optional progress window to update. Default is None, which means no calls are made to this.
- **cycles** (*int*) – Number of LeBail cycles to perform
- **refPlotUpdate** (*function*) – Optional routine used to plot results. Default is None, which means no calls are made to this.
- **seqList** (*list*) – List of histograms to be processed. Default is None which means that all used histograms in .gpx file are processed.

GSASII.GSASIIstrMain.**DoNoFit**(*GPXfile*, *key*)

Compute the diffraction pattern with no refinement of parameters.

TODO: At present, this will compute intensities all diffraction patterns in the project, but this likely can be made faster by dropping all the histograms except key from Histograms.

Parameters

- **GPXfile** (*str*) – G2 .gpx file name
- **key** (*str*) – name of histogram to be computed

Returns

the computed diffraction pattern for the selected histogram

GSASII.GSASIIstrMain.**PrintDistAngle**(*DisAglCtls*, *DisAglData*, *out=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='utf-8'>*)

Print distances and angles

Parameters

- **DisAglCtls** (*dict*) – contains distance/angle radii usually defined using `GSASIIctrlGUI.DisAglDialog()`
- **DisAglData** (*dict*) – contains phase data: Items 'OrigAtoms' and 'TargAtoms' contain the atoms to be used for distance/angle origins and atoms to be used as targets. Item 'SGData' has the space group information (see *Space Group object*)
- **out** (*file*) – file object for output. Defaults to sys.stdout.

GSASII.GSASIIstrMain.**Refine**(*GPXfile*, *dlg=None*, *makeBack=True*, *refPlotUpdate=None*, *newLeBail=False*, *allDerivs=False*)

Global refinement – refines to minimize against all histograms. This can be called in one of three ways, from `GSASIIdataGUI.GSASII.OnRefine()` in an interactive refinement, where dlg will be a wx.ProgressDialog, or non-interactively from `GSASIIscriptable.G2Project.refine()` or from *do_refine()*, where dlg will be None.

GSASII.GSASIIstrMain.**RefineCore**(*Controls*, *Histograms*, *Phases*, *restraintDict*, *rigidbodyDict*, *parmDict*, *histDict1*, *varyList*, *calcControls*, *pawleyLookup*, *ifSeq*, *printFile*, *dlg*, *refPlotUpdate=None*)

Core optimization routines, shared between SeqRefine and Refine

Returns

5-tuple of ifOk (bool), Rvals (dict), result, covMatrix, sig

GSASII.GSASIIstrMain.**ReportProblems**(*result*, *Rvals*, *varyList*)

Create a message based results from the refinement

GSASII.GSASIIstrMain.**RetDistAngle**(*DisAglCtls*, *DisAglData*, *dlg=None*, *dmin=0.5*)

Compute and return distances and angles

Parameters

- **DisAglCtls** (*dict*) – contains distance/angle radii usually defined using `GSASIIctrlGUI.DisAglDialog()`. Example:

```
{'Name': 'Example',
'Factors': [0.85, 0.85],
'AtomTypes': ['Co', 'C', 'N', 'O', 'H'],
'BondRadii': [2.2, 1.12, 1.08, 1.09, 0.5],
'AngleRadii': [1.25, 0.92, 0.88, 0.89, 0.98]}
```

- **DisAglData** (*dict*) – contains phase & refinement data:
  - 'OrigAtoms' and 'TargAtoms' contain the atoms to be used for distance/angle origins and atoms to be used as targets.
  - 'OrigIndx' contains the index numbers for the Origin atoms.
  - 'SGData' has the space group information (see *Space Group object*)
  - 'pId' has the phase id
  - 'Cell' has the unit cell parameters and cell volume
  - 'covData' has the contents of Covariance data tree item

  Added for use with rigid bodies:
  - 'RBlist' has the index numbers for atoms in a rigid body
  - 'rigidbodyDict' the contents of the main Rigid Body data tree item
  - 'Phases' has the phase information for all used phases in the data tree. Only the current phase is needed, but this is easy.
  - 'parmDict' is the GSAS-II parameter dict
- **dlg** – an optional wx progress window that is updated as search progresses
- **dmin** (*float*) – distances less than this distance are not reported as unrealistic (defaults to 0.5 A)

Returns

AtomLabels, DistArray, AngArray where:

AtomLabels is a dict of atom labels, keys are the atom number

DistArray is a dict keyed by the origin atom number where the value is a list of distance entries. The value for each distance is a list containing:

0) the target atom number (int);

1) the unit cell offsets added to x,y & z (tuple of int values);

2) the symmetry transformation, which includes the symmetry operator number, any centering, if a center of symmetry was applied;

3) an interatomic distance in A (float);

4) an uncertainty on the distance in A or 0.0 (float).

AngArray is a dict keyed by the origin (central) atom number where the value is a list of angle entries. The value for each angle entry consists of three values:

0) a distance item reference for one neighbor atom (int);

1) a distance item reference for the second neighbor atom (int);

2) a angle, uncertainty pair; the s.u. may be zero (degrees, tuple of two floats).

The AngArray distance reference items refer directly to the index of the items in the DistArray item which gives the distance from the central atom and the applied symmetry transformation for the two target atoms.

GSASII.GSASIIstrMain.**SeqRefine**(*GPXfile*, *dlg*, *refPlotUpdate=None*)

Perform a sequential refinement – cycles through all selected histgrams, one at a time

GSASII.GSASIIstrMain.**do_refine**(**args*)

Called to run a refinement when this module is executed

GSASII.GSASIIstrMain.**dropOOBvars**(*varyList*, *parmDict*, *sigDict*, *Controls*, *parmFrozenList*)

Find variables in the parameters dict that are outside the ranges (in parmMinDict and parmMaxDict) and set them to the limits values. Add any such variables into the list of frozen variable (parmFrozenList). Returns a list of newly frozen variables, if any.

GSASII.GSASIIstrMain.**phaseCheck**(*phaseVary*, *Phases*, *histogram*)

Removes unused parameters from phase varylist if phase not in histogram for seq refinement removes vars in "Fix FXU" and "FixedSeqVars" here

10.2 *GSASIIstrMath - structure math routines*

10.2.1 GSASIIstrMath Classes & Routines

GSASIIstrMath routines, found below, used to support refinement-related computations. These routines are used primarily in GSASIIstrMain and GSASIIstrIO, but also in a few other routines in other locations: The GSASIIfiles.ExportBaseclass.loadParmDict() routine accesses routine *computeRBsu()*, GSASIIdataGUI.GSASII.OnExpressionCalc() accesses *ApplyRBModels()* and in module testDeriv routines *errRefine()* and *dervRefine()* are accessed a several places.

The routines here are most commonly called when working from a .gpx file, but may sometimes be called from the GUI. These routines expect that all needed input will have been read from the file/tree and are passed to the routines as arguments. The data tree is never accessed directly here.

GSASII.GSASIIstrMath.**ApplyRBModelDervs**(*dFdvDict*, *parmDict*, *rigidbodyDict*, *Phase*)

Computes rigid body derivatives w/r to RB params; N.B.: there are none for Spin RBs

GSASII.GSASIIstrMath.**ApplyRBModels**(*parmDict*, *Phases*, *rigidbodyDict*, *Update=False*)

Takes RB info from RBModels in Phase and RB data in rigidbodyDict along with current RB values in parmDict & modifies atom contents (fxyz & Uij) of parmDict Takes RB parameters from parmDict and rigid body desriptions from rigidbodyDict and atomic information from Phases to compute parmDict values.

Parameters

- **parmDict** (*dict*) – parameter dict. This is updated by this routine.
- **Phases** (*dict*) – nested dict with information on all phases (from data tree)
- **rigidbodyDict** (*dict*) – dict with information on all rigid bodies (from data tree)
- **Update** (*bool*) – if True, the rigidbodyDict is updated with parameters in parmDict. (Default: False.)

Returns

a list of parameters that are set by this routine

GSASII.GSASIIstrMath.**ApplyXYZshifts**(*parmDict*, *varyList*)

takes atom x,y,z shift and applies it to corresponding atom x,y,z value

Parameters

- **parmDict** (*dict*) – parameter dictionary
- **varyList** (*list*) – list of variables (not used!)

Returns

newAtomDict - dictionary of new atomic coordinate names & values; key is parameter shift name

GSASII.GSASIIstrMath.**Dict2Values**(*parmdict*, *varylist*)

Use before call to leastsq to setup list of values for the parameters in parmdict, as selected by key in varylist

GSASII.GSASIIstrMath.**GetAbsorb**(*refl*, *im*, *hfx*, *calcControls*, *parmDict*)

Needs a doc string

GSASII.GSASIIstrMath.**GetAbsorbDerv**(*refl*, *im*, *hfx*, *calcControls*, *parmDict*)

Needs a doc string

GSASII.GSASIIstrMath.**GetAtomFXU**(*pfx*, *calcControls*, *parmDict*)

Needs a doc string

GSASII.GSASIIstrMath.**GetAtomSSFXU**(*pfx*, *calcControls*, *parmDict*)

Needs a doc string

GSASII.GSASIIstrMath.**GetFobsSq**(*Histograms*, *Phases*, *parmDict*, *calcControls*)

Compute the observed structure factors for Powder histograms and store in reflection array Multiprocessing support added

GSASII.GSASIIstrMath.**GetHStrainShift**(*refl*, *im*, *SGData*, *phfx*, *hfx*, *calcControls*, *parmDict*)

Computes the shifts in peak position due to the Hydrostatic strain (HStrain, Dij terms).

GSASII.GSASIIstrMath.**GetHStrainShiftDerv**(*refl*, *im*, *SGData*, *phfx*, *hfx*, *calcControls*, *parmDict*)

Computes the derivatives due to the shifts in peak position from Hydrostatic strain (HStrain, Dij terms).

GSASII.GSASIIstrMath.**GetIntensityCorr**(*refl*, *im*, *uniq*, *G*, *g*, *pfx*, *phfx*, *hfx*, *SGData*, *calcControls*, *parmDict*)

Needs a doc string

GSASII.GSASIIstrMath.**GetIntensityDerv**(*refl*, *im*, *wave*, *uniq*, *G*, *g*, *pfx*, *phfx*, *hfx*, *SGData*, *calcControls*, *parmDict*)

Needs a doc string

GSASII.GSASIIstrMath.**GetNewCellParms**(*parmDict*, *varyList*)

Compute unit cell tensor terms from varied Aij and Dij values. Terms are included in the dict only if Aij or Dij is varied.

GSASII.GSASIIstrMath.**GetPrefOri**(*uniq*, *G*, *g*, *phfx*, *hfx*, *SGData*, *calcControls*, *parmDict*)

March-Dollase preferred orientation correction

GSASII.GSASIIstrMath.**GetPrefOriDerv**(*refl*, *im*, *uniq*, *G*, *g*, *phfx*, *hfx*, *SGData*, *calcControls*, *parmDict*)

Needs a doc string

GSASII.GSASIIstrMath.**GetPwdrExt**(*refl*, *im*, *pfx*, *phfx*, *hfx*, *calcControls*, *parmDict*)

Needs a doc string

GSASII.GSASIIstrMath.**GetPwdrExtDerv**(*refl*, *im*, *pfx*, *phfx*, *hfx*, *calcControls*, *parmDict*)

Needs a doc string

GSASII.GSASIIstrMath.**GetReflPos**(*refl*, *im*, *wave*, *A*, *pfx*, *hfx*, *phfx*, *calcControls*, *parmDict*)

Needs a doc string

GSASII.GSASIIstrMath.**GetReflPosDerv**(*refl*, *im*, *wave*, *A*, *pfx*, *hfx*, *phfx*, *calcControls*, *parmDict*)

Needs a doc string

GSASII.GSASIIstrMath.**GetSampleSigGam**(*refl*, *im*, *wave*, *G*, *GB*, *SGData*, *hfx*, *phfx*, *calcControls*, *parmDict*)

Computes the sample-dependent Lorentzian & Gaussian peak width contributions from size & microstrain parameters

Parameters

wave (*float*) – wavelength for CW data, 2-theta for EDX data

GSASII.GSASIIstrMath.**GetSampleSigGamDerv**(*refl*, *im*, *wave*, *G*, *GB*, *SGData*, *hfx*, *phfx*, *calcControls*, *parmDict*)

Computes the derivatives on sample-dependent Lorentzian & Gaussian peak widths contributions from size & microstrain parameters

Parameters

wave (*float*) – wavelength for CW data, 2-theta for EDX data

GSASII.GSASIIstrMath.**HessRefine**(*values*, *HistoPhases*, *parmDict*, *histDict1*, *varylist*, *calcControls*, *pawleyLookup*, *dlg*)

Loop over histograms and compute derivatives of the fitting model (M) with respect to all parameters. For each histogram, the Jacobian matrix, dMdv, with dimensions (n by m) where n is the number of parameters and m is the number of data points *in the histogram*. The (n by n) Hessian is computed from each Jacobian and it is returned. This routine is used when refinement derivatives are selected as "analytic Hessian" in Controls.

Returns

Vec,Hess where Vec is the least-squares vector and Hess is the Hessian

GSASII.GSASIIstrMath.**MagStructureFactor2**(*refDict*, *G*, *hfx*, *pfx*, *SGData*, *calcControls*, *parmDict*)

Compute neutron magnetic structure factors for all h,k,l for phase puts the result, F^2, in each ref[8] in refList operates on blocks of 100 reflections for speed input:

Parameters

- **refDict** (*dict*) – where 'RefList' list where each ref = h,k,l,it,d,... 'FF' dict of form factors - filed in below
- **G** (*np.array*) – reciprocal metric tensor
- **pfx** (*str*) – phase id string
- **SGData** (*dict*) – space group info. dictionary output from SpcGroup
- **calcControls** (*dict*)
- **ParmDict** (*dict*)

Returns

copy of new refList - used in calculating numerical derivatives

GSASII.GSASIIstrMath.**MagStructureFactorDerv**(*refDict*, *G*, *hfx*, *pfx*, *SGData*, *calcControls*, *parmDict*)

Compute nonmagnetic structure factor derivatives on blocks of reflections in magnetic structures - for powders/nontwins only input:

Parameters

- **refDict** (*dict*) – where 'RefList' list where each ref = h,k,l,it,d,... 'FF' dict of form factors - filled in below

- **G** (*np.array*) – reciprocal metric tensor
- **hfx** (*str*) – histogram id string
- **pfx** (*str*) – phase id string
- **SGData** (*dict*) – space group info. dictionary output from SpcGroup
- **calcControls** (*dict*)
- **parmDict** (*dict*)

Returns
dict dFdvDict: dictionary of derivatives

GSASII.GSASIIstrMath.**MagStructureFactorDerv2**(*refDict*, *G*, *hfx*, *pfx*, *SGData*, *calcControls*, *parmDict*)

Compute magnetic structure factor derivatives numerically - for powders/nontwins only input:

Parameters

- **refDict** (*dict*) – where 'RefList' list where each ref = h,k,l,it,d,... 'FF' dict of form factors - filled in below
- **G** (*np.array*) – reciprocal metric tensor
- **hfx** (*str*) – histogram id string
- **pfx** (*str*) – phase id string
- **SGData** (*dict*) – space group info. dictionary output from SpcGroup
- **calcControls** (*dict*)
- **parmDict** (*dict*)

Returns
dict dFdvDict: dictionary of magnetic derivatives

GSASII.GSASIIstrMath.**MakeSpHarmFF**(*HKL*, *Amat*, *Bmat*, *SHCdict*, *Tdata*, *hType*, *FFtables*, *ORBtables*, *BLtables*, *FF*, *SQ*, *ifDeriv=False*)

Computes hkl dependent form factors & derivatives from spinning rigid bodies

Parameters

- **HKL** (*array*) – reflection hkl set to be considered
- **Bmat** (*array*) – inv crystal to Cartesian transfomation matrix
- **SHCdict** (*dict*) – RB spin/deformation data
- **Tdata** (*array*) – atom type info
- **hType** (*str*) – histogram type
- **FFtables** (*dict*) – x-ray form factor tables
- **ORBtables** (*dict*) – x-ray orbital form factor tables
- **BLtables** (*dict*) – neutron scattering lengths
- **FF** (*array*) – form factors - will be modified by adding the spin/deformation RB spherical harmonics terms
- **SQ** (*array*) – 1/4d^2 for the HKL set
- **ifDeriv** (*bool*) – True if dFF/dcoff to be returned

Returns
dict dFFdS of derivatives if ifDeriv = True

GSASII.GSASIIstrMath.**SCExtinction**(*ref*, *im*, *phfx*, *hfx*, *pfx*, *calcControls*, *parmDict*, *varyList*)

Single crystal extinction function; returns extinction & derivatives

GSASII.GSASIIstrMath.**SHPOcal**(*refl*, *im*, *g*, *phfx*, *hfx*, *SGData*, *calcControls*, *parmDict*)

spherical harmonics preferred orientation (cylindrical symmetry only)

GSASII.GSASIIstrMath.**SHPOcalDerv**(*refl*, *im*, *g*, *phfx*, *hfx*, *SGData*, *calcControls*, *parmDict*)

spherical harmonics preferred orientation derivatives (cylindrical symmetry only)

GSASII.GSASIIstrMath.**SHTXcal**(*refl*, *im*, *g*, *pfx*, *hfx*, *SGData*, *calcControls*, *parmDict*)

Spherical harmonics texture

GSASII.GSASIIstrMath.**SHTXcalDerv**(*refl*, *im*, *g*, *pfx*, *hfx*, *SGData*, *calcControls*, *parmDict*)

Spherical harmonics texture derivatives

GSASII.GSASIIstrMath.**SStructureFactor**(*refDict*, *G*, *hfx*, *pfx*, *SGData*, *SSGData*, *calcControls*, *parmDict*)

Compute super structure factors for all h,k,l,m for phase - no twins puts the result, F^2, in each ref[9] in refList works on blocks of 32 reflections for speed input:

Parameters

- **refDict** (*dict*) – where 'RefList' list where each ref = h,k,l,m,it,d,... 'FF' dict of form factors - filed in below
- **G** (*np.array*) – reciprocal metric tensor
- **pfx** (*str*) – phase id string
- **SGData** (*dict*) – space group info. dictionary output from SpcGroup
- **calcControls** (*dict*)
- **ParmDict** (*dict*)

GSASII.GSASIIstrMath.**SStructureFactorDerv**(*refDict*, *im*, *G*, *hfx*, *pfx*, *SGData*, *SSGData*, *calcControls*, *parmDict*)

Compute super structure factor derivatives for all h,k,l,m for phase - no twins Only Fourier component are done analytically here input:

Parameters

- **refDict** (*dict*) – where 'RefList' list where each ref = h,k,l,m,it,d,... 'FF' dict of form factors - filled in below
- **im** (*int*) – = 1 (could be eliminated)
- **G** (*np.array*) – reciprocal metric tensor
- **hfx** (*str*) – histogram id string
- **pfx** (*str*) – phase id string
- **SGData** (*dict*) – space group info. dictionary output from SpcGroup
- **SSGData** (*dict*) – super space group info.
- **calcControls** (*dict*)
- **ParmDict** (*dict*)

Returns
dict dFdvDict: dictionary of derivatives

GSASII.GSASIIstrMath.**SStructureFactorDerv2**(*refDict*, *im*, *G*, *hfx*, *pfx*, *SGData*, *SSGData*, *calcControls*, *parmDict*)

Compute super structure factor derivatives for all h,k,l,m for phase - no twins input:

Parameters

- **refDict** (*dict*) – where 'RefList' list where each ref = h,k,l,m,it,d,... 'FF' dict of form factors - filled in below
- **im** (*int*) – = 1 (could be eliminated)
- **G** (*np.array*) – reciprocal metric tensor
- **hfx** (*str*) – histogram id string
- **pfx** (*str*) – phase id string
- **SGData** (*dict*) – space group info. dictionary output from SpcGroup
- **SSGData** (*dict*) – super space group info.
- **calcControls** (*dict*)
- **ParmDict** (*dict*)

Returns

dict dFdvDict: dictionary of derivatives

GSASII.GSASIIstrMath.**SStructureFactorDervTw**(*refDict*, *im*, *G*, *hfx*, *pfx*, *SGData*, *SSGData*, *calcControls*, *parmDict*)

Needs a doc string

GSASII.GSASIIstrMath.**SStructureFactorTw**(*refDict*, *G*, *hfx*, *pfx*, *SGData*, *SSGData*, *calcControls*, *parmDict*)

Compute super structure factors for all h,k,l,m for phase - twins only puts the result, F^2, in each ref[8+im] in refList works on blocks of 32 reflections for speed input:

Parameters

- **refDict** (*dict*) – where 'RefList' list where each ref = h,k,l,m,it,d,... 'FF' dict of form factors - filed in below
- **G** (*np.array*) – reciprocal metric tensor
- **pfx** (*str*) – phase id string
- **SGData** (*dict*) – space group info. dictionary output from SpcGroup
- **calcControls** (*dict*)
- **ParmDict** (*dict*)

GSASII.GSASIIstrMath.**StructureFactor2**(*refDict*, *G*, *hfx*, *pfx*, *SGData*, *calcControls*, *parmDict*)

Compute structure factors for all h,k,l for phase puts the result, F^2, in each ref[8] in refList operates on blocks of 100 reflections for speed input:

Parameters

- **refDict** (*dict*) – where 'RefList' list where each ref = h,k,l,it,d,... 'FF' dict of form factors - filed in below
- **G** (*np.array*) – reciprocal metric tensor
- **pfx** (*str*) – phase id string
- **SGData** (*dict*) – space group info. dictionary output from SpcGroup

- **calcControls** (*dict*)
- **ParmDict** (*dict*)

GSASII.GSASIIstrMath.**StructureFactorDerv2**(*refDict*, *G*, *hfx*, *pfx*, *SGData*, *calcControls*, *parmDict*)

Compute structure factor derivatives on blocks of reflections - for powders/nontwins only faster than StructureFactorDerv - correct for powders/nontwins!! input:

Parameters

- **refDict** (*dict*) – where 'RefList' list where each ref = h,k,l,it,d,... 'FF' dict of form factors - filled in below
- **G** (*np.array*) – reciprocal metric tensor
- **hfx** (*str*) – histogram id string
- **pfx** (*str*) – phase id string
- **SGData** (*dict*) – space group info. dictionary output from SpcGroup
- **calcControls** (*dict*)
- **parmDict** (*dict*)

Returns

dict dFdvDict: dictionary of derivatives

GSASII.GSASIIstrMath.**StructureFactorDervTw2**(*refDict*, *G*, *hfx*, *pfx*, *SGData*, *calcControls*, *parmDict*)

Compute structure factor derivatives on blocks of reflections - for twins only faster than StructureFactorDervTw input:

Parameters

- **refDict** (*dict*) – where 'RefList' list where each ref = h,k,l,it,d,... 'FF' dict of form factors - filled in below
- **G** (*np.array*) – reciprocal metric tensor
- **hfx** (*str*) – histogram id string
- **pfx** (*str*) – phase id string
- **SGData** (*dict*) – space group info. dictionary output from SpcGroup
- **calcControls** (*dict*)
- **parmDict** (*dict*)

Returns

dict dFdvDict: dictionary of derivatives

GSASII.GSASIIstrMath.**Values2Dict**(*parmdict*, *varylist*, *values*)

Use after call to leastsq to update the parameter dictionary with values corresponding to keys in varylist

GSASII.GSASIIstrMath.**calcMassFracs**(*varyList*, *covMatrix*, *Phases*, *hist*, *hId*)

Compute mass fractions and their uncertainties for all phases in a histogram. Computed using the covariance matrix, along with the derivatives for the mass fraction equations.

Parameters

- **varyList** (*list*) – list of varied parameters
- **covMatrix** (*np.array*) – covariance matrix, order of rows and columns must match varyList
- **Phases** (*dict*) – data structure (from tree or .gpx) with all phase information

- **hist** (*str*) – name of selected histogram
- **hId** (*int*) – number of current histogram

Returns
valDict,sigDict which contain the mass fraction values and sigmas, keyed by "ph:h:WgtFrac"

GSASII.GSASIIstrMath.**computeRBsu**(*parmDict*, *Phases*, *rigidbodyDict*, *covMatrix*, *CvaryList*, *Csig*)

Computes s.u. values for atoms in rigid bodies

Parameters

- **parmDict** (*dict*) – parameter dict. This is not changed by this routine.
- **Phases** (*dict*) – nested dict with information on all phases (from data tree)
- **rigidbodyDict** (*dict*) – dict with information on all rigid bodies (from data tree)
- **covMatrix** (*np.array*) – covariance matrix (length NxN)
- **CvaryList** (*np.array*) – list of refined parameters (length N)
- **Csig** (*np.array*) – s.u. values for items in CvaryList (length N)

Returns
a dict with s.u. values for parameters that are generated by Rigid Bodies. Will be an empty dict if there are no RBs in use.

GSASII.GSASIIstrMath.**dervHKLF**(*Histogram*, *Phase*, *calcControls*, *varylist*, *parmDict*, *rigidbodyDict*)

Loop over reflections in a HKLF histogram and compute derivatives of the fitting model (M) with respect to all parameters. Independent and dependant dM/dp arrays are returned to either dervRefine or HessRefine.

Returns

GSASII.GSASIIstrMath.**dervRefine**(*values*, *HistoPhases*, *parmDict*, *histDict1*, *varylist*, *calcControls*, *pawleyLookup*, *dlg*)

Loop over histograms and compute derivatives of the fitting model (M) with respect to all parameters. Results are returned in a Jacobian matrix (aka design matrix) of dimensions (n by m) where n is the number of parameters and m is the number of data points. This can exceed memory when m gets large. This routine is used when refinement derivatives are selected as "analtytic Jacobian" in Controls.

Returns
Jacobian numpy.array dMdv for all histograms concatinated

GSASII.GSASIIstrMath.**errRefine**(*values*, *HistoPhases*, *parmDict*, *histDict1*, *varylist*, *calcControls*, *pawleyLookup*, *dlg=None*)

Computes the point-by-point discrepancies between every data point in every histogram and the observed value. Used in the Jacobian, Hessian & numeric least-squares to compute function

Returns
an np array of differences between observed and computed diffraction values.

GSASII.GSASIIstrMath.**getPowderProfile**(*parmDict*, *histDict1*, *x*, *varylist*, *Histogram*, *Phases*, *calcControls*, *pawleyLookup*, *histogram=None*)

Computes the powder pattern for a histogram based on contributions from all used phases

GSASII.GSASIIstrMath.**getPowderProfileDerv**(*args*)

Computes the derivatives of the computed powder pattern with respect to all refined parameters. Used for single processor & Multiprocessor versions

GSASII.GSASIIstrMath.**penaltyDeriv**(*pNames*, *pVal*, *HistoPhases*, *calcControls*, *parmDict*, *varyList*)

Compute derivatives on user-supplied and built-in restraint (penalty) functions

where pNames is list of restraint labels

Returns

array pDerv: partial derivatives by variable# in varList and restraint# in pNames (pDerv[variable#][restraint#])

GSASII.GSASIIstrMath.**penaltyFxn**(*HistoPhases*, *calcControls*, *parmDict*, *varyList*)

Compute user-supplied and built-in restraint functions

10.3 *GSASIIstrIO: structure I/O routines*

10.3.1 GSASIIstrIO Classes & Routines

GSASIIstrIO routines, used for refinement to read from GPX files and print to the .LST file. Used for refinements and in G2scriptable.

This file should not contain any wxpython references as this must be used in non-GUI settings.

GSASII.GSASIIstrIO.**GPXBackup**(*GPXfile*, *makeBack=True*)

makes a backup of the specified .gpx file

Parameters

- **GPXfile** (*str*) – full .gpx file name
- **makeBack** (*bool*) – if True (default), the backup is written to a new file; if False, the last backup is overwritten

Returns

the name of the backup file that was written

GSASII.GSASIIstrIO.**GetAllPhaseData**(*GPXfile*, *PhaseName*)

Returns the entire dictionary for PhaseName from GSASII gpx file

Parameters

- **GPXfile** (*str*) – full .gpx file name
- **PhaseName** (*str*) – phase name

Returns

phase dictionary or None if PhaseName is not present

GSASII.GSASIIstrIO.**GetControls**(*GPXfile*)

Returns dictionary of control items found in GSASII gpx file

Parameters

GPXfile (*str*) – full .gpx file name

Returns

dictionary of control items

GSASII.GSASIIstrIO.**GetFprime**(*controlDict*, *Histograms*)

Needs a doc string

GSASII.GSASIIstrIO.**GetFullGPX**(*GPXfile*)

Returns complete contents of GSASII gpx file. Used in GSASIIscriptable.LoadDictFromProjFile().

Parameters
GPXfile (*str*) – full .gpx file name

Returns

Project,nameList, where

- Project (dict) is a representation of gpx file following the GSAS-II tree structure for each item: key = tree name (e.g. 'Controls', 'Restraints', etc.), data is dict
- nameList (list) has names of main tree entries & subentries used to reconstruct project file

GSASII.GSASIIstrIO.**GetHistogramData**(*Histograms*, *Print=True*, *pFile=None*)

needs a doc string

GSASII.GSASIIstrIO.**GetHistogramNames**(*GPXfile*, *hTypes*)

Returns a list of histogram names found in a GSAS-II .gpx file that match specifed histogram types. Names are returned in the order they appear in the file.

Parameters

- **GPXfile** (*str*) – full .gpx file name
- **hTypes** (*str*) – list of histogram types

Returns
list of histogram names (types = PWDR & HKLF)

GSASII.GSASIIstrIO.**GetHistogramPhaseData**(*Phases*, *Histograms*, *Controls={}*, *Print=True*, *pFile=None*, *resetRefList=True*)

Loads the HAP histogram/phase information into dicts

Parameters

- **Phases** (*dict*) – phase information
- **Histograms** (*dict*) – Histogram information
- **Print** (*bool*) – prints information as it is read
- **pFile** (*file*) – file object to print to (the default, None causes printing to the console)
- **resetRefList** (*bool*) – Should the contents of the Reflection List be initialized on loading. The default, True, initializes the Reflection List as it is loaded.

Returns
(hapVary,hapDict,controlDict) * hapVary: list of refined variables * hapDict: dict with refined variables and their values * controlDict: dict with fixed parameters

GSASII.GSASIIstrIO.**GetHistograms**(*GPXfile*, *hNames*)

Returns a dictionary of histograms found in GSASII gpx file

Parameters

- **GPXfile** (*str*) – full .gpx file name
- **hNames** (*str*) – list of histogram names

Returns
dictionary of histograms (types = PWDR & HKLF)

GSASII.GSASIIstrIO.**GetPawleyConstr**(*SGLaue*, *PawleyRef*, *im*, *pawleyVary*)

needs a doc string

GSASII.GSASIIstrIO.**GetPhaseData**(*PhaseData*, *RestraintDict={}*, *rbIds={}*, *Print=True*, *pFile=None*, *seqHistName=None*, *symHold=None*)

Setup the phase information for a structural refinement, used for regular and sequential refinements, optionally printing information to the .lst file (if Print is True). Used as part of refinements but also to generate information on refinement settings. Can be used with dicts from data tree or read from the GPX file. Note that this routine shares a name with routine G2frame.GetPhaseData() (`GSASIIdata.GSASII.GetPhaseData()`) that instead returns the phase dict(s) from the tree.

Parameters

- **PhaseData** (*dict*) – the contents of the Phase tree item (may be read from .gpx file) with information on all phases
- **RestraintDict** (*dict*) – an optional dict with restraint information
- **rbIds** (*dict*) – an optional dict with rigid body information
- **Print** (*bool*) – a flag that determines if information will be formatted and printed to the .lst file
- **pFile** (*file*) – a file object (created by open) where print information is sent when Print is True
- **seqHistName** (*str*) – will be None, except for sequential fits. For sequential fits, this can be the name of the current histogram or 'All'. If a histogram name is supplied, only the phases used in the current histogram are loaded. If 'All' is specified, all phases are loaded (used for error checking).
- **symHold** (*list*) – if not None (None is the default) the names of parameters held due to symmetry are placed in this list even if not varied. (Used in G2constrGUI and for parameter impact estimates in AllPrmDerivs).

Returns

lots of stuff: Natoms,atomIndx,phaseVary,phaseDict,pawleyLookup, FFtables,EFtables,ORBtables,BLtables,MFtables,maxSSwave (see code for details).

GSASII.GSASIIstrIO.**GetPhaseNames**(*GPXfile*)

Returns a list of phase names found under 'Phases' in GSASII gpx file

Parameters

GPXfile (*str*) – full .gpx file name

Returns

list of phase names

GSASII.GSASIIstrIO.**GetRestraints**(*GPXfile*)

Read the restraints from the GPX file. Throws an exception if not found in the .GPX file

GSASII.GSASIIstrIO.**GetRigidBodies**(*GPXfile*)

Read the rigid body models from the GPX file

GSASII.GSASIIstrIO.**GetRigidBodyModels**(*rigidbodyDict*, *Print=True*, *pFile=None*)

Get Rigid body info from tree entry and print it to .LST file Adds variables and dict items for vector RBs, but for Residue bodies this is done in *GetPhaseData()*.

GSASII.GSASIIstrIO.**GetSeqResult**(*GPXfile*)

Returns the sequential results table information from a GPX file. Called at the beginning of `GSASIIstrMain.SeqRefine()`

Parameters

GPXfile (*str*) – full .gpx file name

Returns
a dict containing the sequential results table

GSASII.GSASIIstrIO.**GetUsedHistogramsAndPhases**(*GPXfile*)

Returns all histograms that are found in any phase and any phase that uses a histogram. This also assigns numbers to used phases and histograms by the order they appear in the file.

Parameters
GPXfile (*str*) – full .gpx file name

Returns

(Histograms,Phases)

- Histograms = dictionary of histograms as {name:data,...}
- Phases = dictionary of phases that use histograms

GSASII.GSASIIstrIO.**IndexGPX**(*GPXfile*, *read=False*)

Create an index to a GPX file, optionally the file into memory. The byte size of the GPX file is saved. If this routine is called again, and if this size does not change, indexing is not repeated since it is assumed the file has not changed (this can be overriden by setting read=True).

Parameters
GPXfile (*str*) – full .gpx file name

Returns

Project,nameList if read=, where

- Project (dict) is a representation of gpx file following the GSAS-II tree structure for each item: key = tree name (e.g. 'Controls', 'Restraints', etc.), data is dict
- nameList (list) has names of main tree entries & subentries used to reconstruct project file

GSASII.GSASIIstrIO.**PrintISOmodes**(*pFile*, *Phases*, *parmDict*, *sigDict*)

Prints the values for the ISODISTORT modes into the project's .lst file after a refinement.

GSASII.GSASIIstrIO.**PrintIndependentVars**(*parmDict*, *varyList*, *sigDict*, *PrintAll=False*, *pFile=None*)

Print the values and uncertainties on the independent parameters

GSASII.GSASIIstrIO.**ReadCheckConstraints**(*GPXfile*, *seqHist=None*, *Histograms=None*, *Phases=None*)

Load constraints and related info and return any error or warning messages This is done from the GPX file rather than the tree.

Parameters

- **GPXfile** (*str*) – specifies the path to a .gpx file.
- **seqHist** (*str*) – specifies a histogram to be loaded for a sequential refinement. If None (default) all are loaded.
- **Histograms** (*dict*) – output from *GetUsedHistogramsAndPhases()*, can optionally be supplied to save time for sequential refinements
- **Phases** (*dict*) – output from *GetUsedHistogramsAndPhases()*, can optionally be supplied to save time for sequential refinements

GSASII.GSASIIstrIO.**ReadConstraints**(*GPXfile*, *seqHist=None*)

Read the constraints from the GPX file and interpret them

called in *ReadCheckConstraints()*, `GSASIIstrMain.Refine()` and `GSASIIstrMain.SeqRefine()`.

GSASII.GSASIIstrIO.**SaveUpdatedHistogramsAndPhases**(*GPXfile*, *Histograms*, *Phases*, *RigidBodies*, *CovData*, *parmFrozen*)

Save phase and histogram information into "pseudo-gpx" files. The phase information is overwritten each time this is called, but histogram information is appended after each sequential step.

Parameters

- **GPXfile** (*str*) – full .gpx file name
- **Histograms** (*dict*) – dictionary of histograms as {name:data,...}
- **Phases** (*dict*) – dictionary of phases that use histograms
- **RigidBodies** (*dict*) – dictionary of rigid bodies
- **CovData** (*dict*) – dictionary of refined variables, varyList, & covariance matrix
- **parmFrozen** (*dict*) – dict with frozen parameters for all phases and histograms (specified as str values)

GSASII.GSASIIstrIO.**SaveUsedHistogramsAndPhases**(*GPXfile*, *Histograms*, *Phases*, *RigidBodies*, *CovData*, *parmFrozenList*, *makeBack=True*)

Updates gpxfile from all histograms that are found in any phase and any phase that used a histogram. Also updates rigid body definitions. This is used for non-sequential fits, but not for sequential fitting.

Parameters

- **GPXfile** (*str*) – full .gpx file name
- **Histograms** (*dict*) – dictionary of histograms as {name:data,...}
- **Phases** (*dict*) – dictionary of phases that use histograms
- **RigidBodies** (*dict*) – dictionary of rigid bodies
- **CovData** (*dict*) – dictionary of refined variables, varyList, & covariance matrix
- **parmFrozenList** (*list*) – list of parameters (as str) that are frozen due to limits; converted to GSASIIobj.G2VarObj objects.
- **makeBack** (*bool*) – True if new backup of .gpx file is to be made; else use the last one made

GSASII.GSASIIstrIO.**SetHistogramData**(*parmDict*, *sigDict*, *Histograms*, *calcControls*, *Print=True*, *pFile=None*, *seq=False*)

Shows histogram data after a refinement

GSASII.GSASIIstrIO.**SetHistogramPhaseData**(*parmDict*, *sigDict*, *Phases*, *Histograms*, *calcControls*, *Print=True*, *pFile=None*, *covMatrix=[]*, *varyList=[]*)

Updates parmDict with HAP results from refinement and prints a summary if Print is True

GSASII.GSASIIstrIO.**SetISOmodes**(*parmDict*, *sigDict*, *Phases*, *pFile=None*)

After a refinement, sets the values for the ISODISTORT modes into the parameter and s.u. dicts. Also, in the case of a non-sequential refinement, prints them into the project's .lst file.

Parameters

- **parmDict** (*dict*) – parameter dict
- **sigDict** (*dict*) – s.u. (uncertainty) dict
- **Phases** (*dict*) – Phase info from tree/.gpx
- **pFile** (*file*) – file for .lst info or None (None for sequential fits)

GSASII.GSASIIstrIO.**SetPhaseData**(*parmDict*, *sigDict*, *Phases*, *RBIds*, *covData*, *RestraintDict=None*, *pFile=None*)

Called after a refinement to transfer parameters from the parameter dict to the phase(s) information read from a GPX file. Also prints values to the .lst file

GSASII.GSASIIstrIO.**SetRigidBodyModels**(*parmDict*, *sigDict*, *rigidbodyDict*, *pFile=None*)

needs a doc string

GSASII.GSASIIstrIO.**SetSeqResult**(*GPXfile*, *Histograms*, *SeqResult*)

Places the sequential results information into a GPX file after a refinement has been completed. Called at the end of `GSASIIstrMain.SeqRefine()`

Parameters

GPXfile (*str*) – full .gpx file name

GSASII.GSASIIstrIO.**SetupSeqSavePhases**(*GPXfile*)

Initialize the files used to save intermediate results from sequential fits.

GSASII.GSASIIstrIO.**ShowBanner**(*pFile=None*)

Print authorship, copyright and citation notice

GSASII.GSASIIstrIO.**ShowControls**(*Controls*, *pFile=None*, *SeqRef=False*, *preFrozenCount=0*)

Print controls information

GSASII.GSASIIstrIO.**SummRestraints**(*restraintDict*)

Summarize number of Restraints in a single line

GSASII.GSASIIstrIO.**WriteRBObjPOAndSig**(*pfx*, *rbfx*, *rbsx*, *parmDict*, *sigDict*)

Cribbed version of PrintRBObjPOAndSig but returns lists of strings. Moved so it can be used in ExportCIF

GSASII.GSASIIstrIO.**WriteRBObjSHCAndSig**(*pfx*, *rbfx*, *rbsx*, *parmDict*, *sigDict*, *SHC*)

Cribbed version of PrintRBObjTorAndSig but returns lists of strings. Moved so it can be used in ExportCIF

GSASII.GSASIIstrIO.**WriteRBObjTLSAndSig**(*pfx*, *rbfx*, *rbsx*, *TLS*, *parmDict*, *sigDict*)

Cribbed version of PrintRBObjTLSAndSig but returns lists of strings. Moved so it can be used in ExportCIF

GSASII.GSASIIstrIO.**WriteRBObjTorAndSig**(*pfx*, *rbsx*, *parmDict*, *sigDict*, *nTors*)

Cribbed version of PrintRBObjTorAndSig but returns lists of strings. Moved so it can be used in ExportCIF

GSASII.GSASIIstrIO.**WriteResRBModel**(*RBModel*)

Write description of a residue rigid body. Code shifted from PrintResRBModel to make usable from G2export_CIF

GSASII.GSASIIstrIO.**WriteVecRBModel**(*RBModel*, *sigDict={}*, *irb=None*)

Write description of a vector rigid body. Code shifted from PrintVecRBModel to make usable from G2export_CIF

GSASII.GSASIIstrIO.**cellFill**(*pfx*, *SGData*, *parmDict*, *sigDict*)

Returns the filled-out reciprocal cell (A) terms and their uncertainties from the parameter and sig dictionaries.

Parameters

- **pfx** (*str*) – parameter prefix ("n::", where n is a phase number)
- **SGdata** (*dict*) – a symmetry object
- **parmDict** (*dict*) – a dictionary of parameters
- **sigDict** (*dict*) – a dictionary of uncertainties on parameters

Returns

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

GSASII.GSASIIstrIO.**cellVary**(*pfx*, *SGData*)

Creates equivalences for a phase based on the Laue class. Returns a list of A tensor terms that are non-zero.

GSASII.GSASIIstrIO.**fmtESD**(*varname*, *SigDict*, *fmtcode*, *ndig=None*, *ndec=None*)

Format an uncertainty value as requested, but surround the number by () if the parameter is set by an equivalence or by [] if the parameter is set by an constraint

Parameters

fmtcode (*str*) – can be a single letter such as 'g' or 'f', or a format string, such as '%10.5f'. For the latter, leave ndig and ndec as None.

GSASII.GSASIIstrIO.**getBackupName**(*GPXfile*, *makeBack*)

Get the name for the backup .gpx file name

Parameters

- **GPXfile** (*str*) – full .gpx file name
- **makeBack** (*bool*) – if True the name of a new file is returned, if False the name of the last file that exists is returned

Returns

the name of a backup file

GSASII.GSASIIstrIO.**gpxSize = -1**

Global variables used in *IndexGPX()* to see if file has changed (gpxSize) and to index where to find each 1st-level tree item in the file.

CHAPTER

ELEVEN

GSASIIMAPVARS: PARAM CONSTRAINTS

11.1 Summary/Contents

Module to implements algebraic contraints, parameter redefinition and parameter simplification contraints.

Section Contents

11.2 Externally-Accessible Routines

To define a set of constrained and unconstrained relations, one defines a list of dictionary defining constraint parameters and their values, a list of fixed values for each constraint and a list of parameters to be varied. In addition, one uses `StoreEquivalence()` to define parameters that are equivalent. See the *Constraints Processing section* for details on how processing of constraints is done.

routine	explanation
InitVars()	This is used to clear out all defined previously defined constraint information
StoreEquivalence()	Implements parameter redefinition. This should be called for every set of equivalence relationships. Use *StoreEquivalence()* before calling *GenerateConstraints()*
ProcessConstraints()	Initially constraints of all types are maintained in lists of dict entries that are stored in the data tree, with parameters are stored as `G2VarObj` objects so that they can be resolved if the phase/histogram order changes. *ProcessConstraints()* processes this list of dict entries, separating the "Equivalence", "Hold", "Const" and "New Var" entries for subsequent use. See the *Constraint Reorganization* section for more details.
EvaluateMultipliers()	Convert any string-specified (formula-based) multipliers to numbers. Call this before using *GenerateConstraints()*. At present only values in dict for phase (atom/cell) parameters are used to evaluate multipliers containint formulae, but this could be changed if needed.
GenerateConstraints()	Generates the internally-used tables from constraints and equivalences. Checks for internal consistency and repairs problems where possible. See the *Constraint Checking and Grouping* and *Equivalence Checking* sections for more details.
Map2Dict()	To determine values for any parameters created in this module, call Map2Dict. This will not apply contraints.
Dict2Map()	To apply the constraints and equivalences, call this. It takes values from the new independent parameters and constraints, and applies them to the parameter dict.
Dict2Deriv()	This determines derivatives on independent parameters from those on dependent ones.
ComputeDepESD()	Use ComputeDepESD to compute uncertainties on dependent variables.
VarRemapShow()	Use this to show a summary of the parameter remapping. Call after *GenerateConstraints()*.

11.3 Types of constraints

There are four ways to specify constraints, as listed below. Note that constraints are initially stored in the main section of the GSAS-II data tree under heading `Constraints`. This dict has four keys, 'Hist', 'HAP', 'Global', and 'Phase', each containing a list of constraints. An additional set of constraints are generated for each phase based on symmetry considerations by calling *GetPhaseData()* in module *GSASII.GSASIIstrIO*.

Note that in the constraints, as stored in the GSAS-II data tree, parameters are stored as `GSASIIobj.G2VarObj` objects, as these objects allow for changes in numbering of phases, histograms and atoms since *G2VarObj* objects use random Id's for references. When constraints are interpreted (in *ProcessConstraints()*), these references are resolved to the numbered objects by looking up random Id's so that the parameter object is converted to a string of form `ph:hst:VARNAM:at`.

Constraints are initially stored as described in the *constraint definitions*, where the last value in the list determines which type of constraint is defined.

11.3.1 Alternate parameters (New Var)

Parameter redefinition ("New Var" constraints) is done by creating an expression that relates several parameters:

```
Mx1 * Px + My1 * Py +... = ::newvar1
Mx2 * Px + Mz2 * Pz + ... = ::newvar2
```

where Pj is a GSAS-II parameter name and Mjk is a constant (float) multiplier. Alternately, multipliers Mjk can contain a formula (str) that will be evaluated prior to the start of the refinement. In a formula, GSAS-II parameters will be replaced by the value of the parameter before the formula is evaluated, so `'np.cos(0::Ax:2)'` is a valid multiplier. At present, only phase (atom/cell) parameters are available for use in a formula, from the GUI but this can be expanded if needed.

This type of constraint describes an alternate degree of freedom where parameter Px and Py, etc. are varied to keep their ratio fixed according the expression. A new variable parameter is assigned to each degree of freedom when refined. An example where this can be valuable is when two parameters, P1 and P2, have similar values and are highly correlated. It is often better to create a new variable, Ps = P1 + P2, and refine Ps. In the later stages of refinement, a second variable, Pd = P1 - P2, can be defined and it can be seen if refining Pd is supported by the data. Another use will be to define parameters that express "irrep modes" in terms of the fundamental structural parameters.

The "New Var" constraints are stored as a type "f" *constraint (see definitions)*.

11.3.2 Constrained parameters (Const)

A constraint on a set of variables can be supplied in the form of a linear algebraic equation:

```
Nx1 * Px + Ny1 * Py +... = C1
```

where Cn is a constant (float), where Pj is a GSAS-II parameter name, and where Njk is a constant multiplier (float) or a formula (str) that will be evaluated prior to the start of the refinement. In a formula, GSAS-II parameters will be replaced by the value of the parameter before the formula is evaluated, so `'np.cos(0::Ax:2)'` is a valid multiplier. At present, only phase (atom/cell) parameters are available for use in a formula, but this can be expanded if needed.

These equations set an interdependence between parameters. Common uses of parameter constraints are to set rules that decrease the number of parameters, such as restricting the sum of fractional occupancies for atoms that share a site to sum to unity, thus reducing the effective number of variables by one. Likewise, the Uiso value for a H atom "riding" on a C, N or O atom can be related by a fixed offset (the so called B+1 "rule").

The "Const" constraints are stored as a type "c" *constraint (see definitions)*.

11.3.3 Equivalenced parameters (Equiv)

A simplifed way to set up a constraint equation is to define an equivalence, which can be of form:

```
C1 * P1 = C2 * Py
```

or:

```
C1 * P1 = C2 * P2 = C3 * P3 = ...
```

where Cn is a constant (float), where Pj is a GSAS-II parameter name. This means that parameters Py (or P2 and P3) are determined from (or "slaved" to) parameter P1. Alternately, equivalences can be created with *`StoreEquivalence()`* where the multipliers can be a formula (str) that will be evaluated prior to the start of the refinement. In a formula, GSAS-II parameters will be replaced by the value of the parameter before the formula is evaluated, so a multiplier can be specifed as `'2*np.cos(0::Ax:2)'`. At present, only phase (atom/cell) parameters are available for use in such a formula, but this can be expanded if needed.

The first parameter (P1 above) is considered the independent variable and the remaining parameters are dependent variables. The dependent variables are then set from the independent variable.

Note that a constraint expression is conceptually identical to defining constraint equations. The previous set of equalities could also be written as a set of constraint equations in this way:

```
C1 * P1 - C2 * P2 = 0
C1 * P1 - C3 * P3 = 0
...
```

In practice, however, equivalenced parameters are processed in a different and more direct manner than constraint equations.

A parameter can be used in multiple equivalences where it is an independent variable, but if a parameter were used both as a dependent and independent variable then the order that shifts are applied becomes potentially significant. As an example, in this case these two equivalences are "chained":

```
C1 * P1 = C2 * P2
C2 * P2 = C3 * P3
```

where P2 is both a dependent and independent variable. Likewise, if a parameter is used both in equivalences and in "New Var" or "Const" constraints, it also becomes unclear how this should be processed. It is possible to specify equivalences that conflict with constraints. Should parameter be used as both a dependent and an independent variable or if a parameter is used both in an the equivalence and in a "New Var" or "Const" constraints, the equivalence is converted to a constraint (Const) to avoid conflicts. The equivalences that require this are addressed in *GenerateConstraints()* where *CheckEquivalences()* is used to locate problematic variables in equivalences and then change these equivalences to "Const" equations. Also, unneeded equivalences are removed.

For an example of how equivalences may be used, consider a material that has **N** O atoms in the asymmetric unit, all in fairly similar bonding environments and where the diffraction data are sparse. One may wish to reduce the complexity of the model fit to these data by defining Uiso for all O atoms to be the same. This is done by selecting Uiso for any one O atom as an independent variable in a equivalence and setting the remaining **N-1** other O atom Uiso variables as dependent parameters with multipliers of 1. This will require that all O atom Uiso values be identical. The results of this refinement will be simpler to understand than if a set of constraint equations is used, because the refined parameter (named as `ph::Uiso:n`) will be the independent variable, corresponding to the first O atom and all other variables would be expressed in terms of that variable with a single Equivalence expression. The alternate would require **N-1** constraint equations, leaving one degree of freedom with a variable would that is likely only indirectly related to the Uiso values.

Equivalenced parameters ("EQUIV" constraints), when defined by users, or when created to relate phases, are stored as a type "e" *constraint (see definitions)*. Symmetry-generated equivalences are generated prior to display or refinement in *GetPhaseData()* from module *GSASII.GSASIIstrIO*. These are not stored in the data tree.

11.3.4 Hold parameters (Fixed)

When parameters are refined where a single refinement flag determines that several variables are refined at the same time (examples are: cell parameters, atom positions, anisotropic displacement parameters, magnetic moments,…) it can be useful to specify that a specific parameter should not be varied. These will most commonly be generated due to symmetry, but under specific conditions, there may be other good reasons to constrain a parameter.

The "Hold" constraints are stored as a type "h" *constraint (see definitions)*.

11.4 Constraint Processing

When constraints will be used or edited, they are processed using a series of calls. This is done in GSAS-II from several locations:

- For error checking from the tree in *GSASIIconstrGUI*, routine *CheckConstraints()* loads constraints from the data tree.

- When the table of refined parameters is shown, constraints are also processed in function *GSASII.GSASIIdataGUI.GSASII.OnShowLSParms()* using *CheckConstraints()*
- To write parameters in the Export sections of the program, *GSASII.GSASIIfiles.ExportBaseclass.loadParmDict()* loads results as well as constraints from the tree. This works a bit differently from the above, so it makes direct calls to the constraints routines.
- For error checking from a GPX file *GSASII.GSASIIstrIO.ReadCheckConstraints()* loads constraints (called in *GSASIIdataGUI* and *GSASIIscriptable*), which is similar to *CheckConstraints()*. *ReadCheckConstraints()* is called by *GSASII.GSASIIdataGUI.GSASII.OnRefine()* and *GSASII.GSASIIdataGUI.GSASII.OnSeqRefine()* before constraints are generated for use in refinements so they can be shown in the GUI. This is also called to check for errors in *GSASII.GSASIIscriptable.G2Project*.
- To create the constraints for use in a refinement, in *GSASIIstrMain*, functions *Refine()* and *SeqRefine()* load and process the constraints again. This is repeated here because *Refine()* and *SeqRefine()* are intended to operate as stand-alone routines that may be called directly.
- After sequential fits have been completed, the previously processed constraint info is read from the sequential results section of the data tree. Function *GSASII.GSASIIseqGUI.UpdateSeqResults()* displays the sequential results table also processes constraints.

TODO: Note that G2stIO.makeTwinFrConstr is called only in one place. It probably needs to be included in all of the above.

When constraints are processed, the following steps are used:

1. Constraints are stored in separate lists in the data tree to simplify their creation and their GUI display. In the initial processing, all of the stored constraints are appended into a single list.
2. Then *InitVars()* is used to initialize the global variables in this module (*GSASIImapvars*). This may be done before the previous step.
3. Then *ProcessConstraints()* is used to initially process the constraints user-supplied constraints (from the data tree), as described in *Constraint Reorganization*. When constraints are read from a GPX file, rather than the data tree, use *GSASII.GSASIIstrIO.ReadConstraints()* (which calls *ProcessConstraints()*).
4. Symmetry-generated equivalences are then created in *GSASII.GSASIIstrIO.GetPhaseData()*, which also calls *GSASII.GSASIIstrIO.cellVary()* and for Pawley refinements *GSASII.GSASIIstrIO.GetPawleyConstr()*. These are entered directly into this module's globals using *StoreEquivalence()*.
5. Constraints/equivalences are then checked for possible conflicts with *GenerateConstraints()* which in turn calls *CheckEquivalences()*. These routines group the constraints and possibly reorganize them, as discussed below for *GenerateConstraints()* (*discussed here*) and for *CheckEquivalences()* (*discussed here*).
6. Note that for debugging, *VarRemapShow()* can be called at any point after *GenerateConstraints()* has been called. This will display the generated constraints.

11.4.1 Constraint Reorganization [ProcessConstraints()]

ProcessConstraints() is used to initially process the constraints from the list of dict entries. The "Const" and "New Var" are placed into two lists (`constrDict` and `fixedList`) that are later used for parameter grouping (in *GenerateConstraints()*). "Hold" and "Equivalence" constraints are separated into separate storage.

For "Const" entries,
a dict with multiple entries is placed in `constrDict` where each dict key is the parameter name and the value is the multiplier for the parameter, while `fixedList` gets a string value corresponding to the constant value for the expression.

For "New Var" entries,
a dict with multiple entries defined identically to to that used in "Const" entries. The differences between "New Var" and "Const" entries is that for "Const" entries, a constant value (as a string) is placed in

`fixedList` while for "New Var" entries corresponding entry in `fixedList` is None. Also, one or two additional entries are created in the dict for "New Var" constraints: an entry with key "_vary" is given the value of True or False depending on the refinement flag setting; an entry with key "_name" will be created if the "New Var" parameter has a supplied name.

For "Hold" entries,
User-supplied "Hold" constraints are stored in global variable `holdParmList`. Initialized in *`InitVars()`*; set in *`StoreHold()`*. Type of hold is stored in `holdParmType`.

Equivalences are stored
using *`StoreEquivalence()`* into this module's globals (`dependentParmList`, `arrayList`, `invarrayList`, `indParmList`, and `symGenList`). For each equivalence:

- a list with one entry, the name of the independent parameter is placed in `indParmList`;
- a list with one or more parameter name is placed in `dependentParmList`;
- the value None is added to `arrayList`;
- a list of multipliers for each dependent variable is placed in `invarrayList`
- an entry of either True or False is placed in `symGenList`, where True indicates that the entry has been generated from symmetry.

The output from *`ProcessConstraints()`* will have the form as below, where the first entry is a "Const" and the second is a "New Var".

```
constrDict = [
     {'0:12:Scale': 2.0, '0:14:Scale': 4.0, '0:13:Scale': 3.0, '0:0:Scale':␣
↪0.5},
     {'2::C(10,6,1)': 1.0, '1::C(10,6,1)': 1.0, '_vary':True}]
fixedList = ['5.0', None]
```

11.4.2 Constraint Checking and Grouping [`GenerateConstraints()`]

Function *`GenerateConstraints()`* is used to process the parameter equivalences and constraint lists created in *`ProcessConstraints()`* (`constrDict` and `fixedList`). *`GenerateConstraints()`* is used to generate error/warning messages, to set up lists that are used to show this information for the GUI (using *`getConstrError()`*) and to generate the information stored in *global arrays* that are used later to apply the constraints.

When a sequential refinement is in progress, the constraints are scanned for parameters that have a wildcard (*) for the histogram number, such as 1::*Scale which would refer to the phase fraction for Phase ` in every histogram. The "*" will be replaced with the number of the current histogram.

Equivalences are checked with *`CheckEquivalences()`* (described in detail *below*). This may result in the creation of additional "Hold" and "Constr" constraints being added to the `constrDict` and `fixedList` lists.

The "Const" and "New Var" constraint expressions are then scanned for problems:

Constraints cannot be processed without changes if any of the terms within have the following:

- **Undefined parameters** or **Multiplier of zero**

  If any parameters in a constraint are undefined or have a parameter multiplier of zero the constraint group is not used.

  If some, but not all, parameters in a constraint are undefined or have a parameter multiplier of zero and remaining valid parameters will be set as "Hold". One exception: atom position constraints (p::dA[xyz]:#) will be assumed as zero.

- **Hold (Fixed) parameters** and **Unvaried parameters**: New Var constraints

If any parameters in a new var constraint are either not refined, or are marked as "Hold" the constraint can not be varied. Any parameters in that group will be set as "Hold"

- **Hold (Fixed) parameters** and **Unvaried parameters**: Constraint Equations

  If any parameters in a constraint equation are either not refined, or are marked as "Hold" those parameters can be removed from the constraint, with an adjustment of the equation sum.

Constraint expressions ("Const" and "New Var") are sorted by routine *GroupConstraints()* into groups so that each group contains the minimum number of entries that ensures each parameter is referenced in only one group. This is done by scanning the list of dicts in `constrDict` one by one and making a list of parameters used in that constraint expression. Any expression that contains a parameter in that list is added to the current group and those parameters are added to this list of parameters. The list of ungrouped expressions is then scanned again until no more expressions are added to the current group. This process is repeated until every expression has been placed in a group. Function *GroupConstraints()* returns two lists of lists. The first has, for each group, a list of the indices in `constrDict` that comprise the group (there can be only one). The second list contains, for each group, the unique parameter names in that group.

Each constraint group is then processed. First, wildcard parameters are renamed (in a sequential refinement). Any held parameters that are used in constraints are noted as errors. The number of refined parameters and the number of parameters that are not defined in the current refinement are also noted. It is fine if all parameters in a group are not defined or all are not varied, but if some are defined and others not or some are varied and others not, this constitutes an error.

The contents of each group is then examined. Groups with a single parameter (holds) are ignored. Then for each group, the number of parameters in the group (Np) and the number of expressions in the group (Nc) are counted and for each expression. If Nc > Np, then the constraint is overdetermined, which also constitutes an error.

The parameter multipliers for each expression are then assembled:

```
M1a * P1 + M2a * P2 +... Mka * Pk
M1b * P1 + M2b * P2 +... Mkb * Pk
...
M1j * P1 + M2j * P2 +... Mkj * Pk
```

From this it becomes possible to create a Nc x Np matrix, which is called the constraint matrix:

$$\begin{pmatrix} M_{1a} & M_{2a} & ... & M_{ka} \\ M_{1b} & M_{2b} & ... & M_{kb} \\ ... & & & \\ M_{1j} & M_{2j} & ... & M_{kj} \end{pmatrix}$$

When Nc<Np, then additional rows need to be added to the matrix and to the vector that contains the value for each row (`fixedList`) where values are `None` for New Vars and a constant for fixed values. This then can describe a system of Np simultaneous equations:

$$\begin{pmatrix} M_{1a} & M_{2a} & ... & M_{ka} \\ M_{1b} & M_{2b} & ... & M_{kb} \\ ... & & & \\ M_{1j} & M_{2j} & ... & M_{kj} \end{pmatrix} \begin{pmatrix} P_1 \\ P_2 \\ ... \\ P_k \end{pmatrix} = \begin{pmatrix} C_1 & C_2 & ... & C_k \end{pmatrix}$$

This is done by creating a square matrix from the group using *_FillArray()*. The top Nc rows in the matrix are filled as described above. Then *_RowEchelon()* is used to see if those entries in the matrix can be coverted to row-echelon form. This will raise an Exception there is linear dependence between the initial Nc rows (which means that no matter what values are used for any remaining rows, that the matrix will be singular). If that is not the case and Nc<Np then

any remaining rows that were not specified are filled in. For each of these rows, first only the diagonal element in that row of the matrix is set to 1 and the upper portion of the matrix is again tested with *_RowEchelon()* to check for linear independence. This is likely to be non-singular, but should *_RowEchelon()* fail, *_FillArray()* will then try setting each other element in that row to either 1 or -1. One of those options should be linearly independent from every other row of the matrix.

The Gram-Schmidt process, implemented in *GramSchmidtOrtho()*, is used to find orthonormal unit vectors which are used to replace the remaining Np-Nc rows of the matrix. This will fail with a `ConstraintException` if this is not possible (singular matrix), but that would be unexpected since the matrix could be converted to row-echelon form. The Gram-Schmidt result is placed in `constrArr` as a numpy array.

Rows in the matrix corresponding to "New Var" constraints and those that were generated by the Gram-Schmidt process are provided with parameter names. These names are generated using `paramPrefix`, which is set to `"::constr"`, plus a number to make the new parameter name unique, unless a name was specified for the "New Var" entry by using a `"_name"` element in the constraint dict.

Finally the parameters used as input to the constraint are placed in this module's globals `dependentParmList` and the constraint matrix is placed in into `arrayList`. This can be used to compute the initial values for "New Var" parameters. The inverse of the constraint matrix is placed in `invarrayList` and a list of the "New Var" parameters and a list of the fixed values (as str's) is placed in `indParmList`. Finally the appropriate entry in `symGenList` is set to False to indicate that this is not a symmetry generated constraint.

11.4.3 Equivalence Checking and Reorganization [`CheckEquivalences()`]

Equivalences need to be checked for usages that could be internally conflicted or have possible conflicts with other constraints. Function *CheckEquivalences()* is called within *GenerateConstraints()* to diagnose and where possible resolve such uses, as discussed below.

Mixed parameter use:

> **Note** that multiple passes, cycling through the equivalences may be needed to find all mixed-use parameters, as will be discussed further, below.

- A parameter should not show up as a dependent variable in two equivalence expressions, such as:

```
::x1 -> ::x3
::x2 -> ::x3
```

  This will be processed by turning the equivalences into two constraint equations:

```
::x1 - ::x3 = 0
::x2 - ::x3 = 0
```

  which can be satisfied when `::x1 = ::x2 = ::x3`. If `::x1` and `::x2` had been intended to be independent parameters, then the above equivalences would be conflict and cannot be statisfied.

- If a parameter is used both as an independent and as a dependent variable (*chaining*), as is in these two equivalence expressions:

```
::x1 -> ::x2 & ::x4
::x2 -> ::x3
```

  This can also be addressed by turning these equivalences into three constraint equations:

```
::x1 - ::x2 = 0
::x1 - ::x4 = 0
::x2 - ::x3 = 0
```

which can be satisfied when `::x1 = ::x2 = ::x3 = ::x4`

- Use of parameters in both equivalences and "Const" or "New Var" constraint expressions makes logical sense:

```
::x1 -> ::x2 & ::x4
::x2 + ::x3 = 0
```

This can also be addressed by turning the equivalence into two constraint equations:

```
::x1 - ::x2 = 0
::x1 - ::x4 = 0
```

With the addition of the "Const" equation (`::x2 + ::x3 = 0`), the solution will require `::x1 = ::x2 = -1.0*::x3 = ::x4`

- Cycling is needed to find all equivalences that must be converted. Consider this set of constraints:

```
::x2 + ::x3 = 0
::x1 -> ::x2
::x1 -> ::x4
```

In the first pass the equivalence with `::x2` would be converted to a "Const" constraint and in the second pass the other equivalence with `::x1` would be converted.

Mixing Hold (Fixed) parameters in equivalences

- If one parameter (or more) is designated as a "Hold" in an equivalence, then all parameters in that equivalence cannot be varied. Considering this equivalence:

```
::x1 -> ::x2 & ::x4
```

If any of the three parameters (`::x1`, `::x2`, or *::x4*) are marked as Hold, then the other two parameters may not be varied and will also be set with a "Hold".

Unvaried parameters in equivalences

- If no parameters in an equivalence are varied, then the equivalence is ignored.
- If only some parameters are marked as varied then *none of the parameters can be varied*; any varied parameters will be set with a "Hold".

Undefined parameters in equivalences

Parameters may be placed in equivalences that are not actually defined in a project. This can occur in two ways. If an equivalence is created in the GUI for a parameter that is later supplanted with a different model (for example, changing from isotropic size broadening to uniaxial broadening replaces the isotropic broadening term with two different uniaxial terms) or symmetry may require restrictions on anisotropic ADPs that are not in use).

- If the independent parameter is undefined, then any dependent parameters that are defined are set as "Hold" and the equivalence is ignored.
- If all dependent parameters are undefined, then the equivalence is ignored.
- If a dependent parameter is undefined, then that parameter is dropped from the equivalence.

Multiplier of zero in equivalences

Any dependent parameter that has a multiplier of zero will be dropped from the equivalence. If no terms remain, then the equivalence is ignored. (Independent parameters do not have a multiplier).

11.5 *Global Variables*

This module uses a number of global variables. One set is used to store the constraints and equivalences after processing by *StoreEquivalence()* and *GenerateConstraints()*. These globals are expected to be used only by this module's (*GSASIImapvars*) internal routines.

Lists with information from Constraint Equation and New Var constraints. Each entry in these variables is related to a group of constraints.

variable	explanation
dependentParmList	a list containing group of lists of parameters used in the group. The columns of the matrices in *arrayList* match the order of parameters here. Note that parameters listed in dependentParmList will not be included in the Hessian as their derivatives will not affect the model
indParmList	a list containing groups of variables or constants matching the columns of the matrices in *invarrayList*.
arrayList	a list containing group of relationship matrices to relate parameters in dependentParmList to those in indParmList.
invarrayList	a list containing group of relationship matrices to relate parameters in indParmList to those in dependentParmList. Unlikely to be used externally.
symGenList	a list of boolean values that will be True to indicate that an equivalence was generated internally GSAS-II meaning it is generated based on symmetry, twining or Pawley overlap.

Lists with information from Hold and Equivalence constraints. Each entry in these variables is related to a group of constraints.

variable	explanation
holdParmList	a list of parameters that have been marked as "Hold". Unlikely to be accessed outside this module. Initialized in *InitVars()*; set in *StoreHold()*.
holdParmType	The reason why a parameter has been marked as "Hold". Unlikely to be accessed outside this module. Initialized in *InitVars()*; set in *StoreHold()*.
constrParms	dict with lists of variables in equivalences, constraint equations and new var expressions. Used within *GetIndependentVars()*, and *GetDependentVars()*. Best if not referenced outside this module. Contains elements: • 'dep-equiv': dependent parameters set by equivalences • 'dep-constr': dependent parameters set by constraint equations or new var expressions • 'indep-equiv': dependent parameters used in equivalences • 'indep-constr': dependent parameters created from constraint equations or new var expressions
saveVaryList	a list of the varied parameters used when constraints were last processed.

A second set of global variables are set in *GenerateConstraints()* with lists of parameter names from equivalences and constraints. Used in *CheckEquivalences()* and *getConstrError()*.

variable	explanation
depVarList	a list of the parameters used in equivalences as dependent parameters for all equivalences initially specified (including those to be reclassified as "Constr" constraints.)
indepVarList	a list of the parameters used in equivalences as independent parameters for all equivalences initially specified (including those to be reclassified as "Constr" constraints.)
constrVarList	a list of the parameters that are used in "Constr" or "New Var" constraints. Does not include those in equivalences to be reclassified as "Constr" constraints.)

A third set of global variables to store equivalence warning information. Set in *CheckEquivalences()* and *GenerateConstraints()*. Used in *getConstrError()* to display warning messages.

variable	explanation
convVarList	parameters in equivalences that were converted to "Const" constraints
multdepVarList	parameters used as dependent parameters in equivalences multiple times
unvariedParmsList	parameters used in equivalences and constraints that are not varied
undefinedVars	parameters used in equivalences that are not defined in the parameter dict (parmDict)
groupErrors	parameters in constraints that cause grouping errors

11.6 *GSASIImapvars Routines/variables*

Classes and routines defined in `GSASIImapvars` follow.

Note that parameter names in GSAS-II are strings of form `<ph#>:<hst#>:<nam>` or `<ph#>::<nam>:<at#>` where `<ph#>` is a phase number, `<hst#>` is a histogram number and `<at#>` is an atom number. `<nam>` is a name that determines the parameter type (see `GSASIIobj.CompileVarDesc()`). When stored in the data tree, parameters are saved as `GSASIIobj.G2VarObj` objects so that they can be resolved if the phase/histogram order changes.

`GSASII.GSASIImapvars.`**`CheckEquivalences`**(*constrDict*, *varyList*, *fixedList*, *parmDict=None*, *seqHistNum=None*)

Process equivalence constraints, looking for conflicts such as where a parameter is used in both an equivalence and a constraint expression or where chaining is done (A->B and B->C).

Removes equivalences or parameters from equivalences or converts equivalences to constraints as described for *Equivalence Checking and Reorganization*.

Parameters

- **`constrDict`** (*dict*) – a list of dicts defining relationships/constraints
- **`varyList`** (*list*) – list of varied parameters (defined during refinements only)
- **`fixedList`** (*list*) – a list of values specifying a fixed value for each dict in constrDict. Values are either strings that can be converted to floats or `None` if the constraint defines a new parameter rather than a constant.
- **`parmDict`** (*dict*) – a dict containing defined parameters and their values. Used to find equivalences where a parameter is has been removed from a refinement.
- **`seqHistNum`** (*int*) – the hId number of the current histogram in a sequential fit. None (default) otherwise.

Returns

warning messages about changes that need to be made to equivalences

GSASII.GSASIImapvars.**ComputeDepESD**(*covMatrix*, *varyList*, *noSym=False*)

Compute uncertainties for dependent parameters from independent ones returns a dictionary containing the esd values for dependent parameters

Parameters

- **covMatrix** (*np.array*) – the full covariance matrix
- **varyList** (*list*) – the names of the variables matching the columns and rows in covMatrix
- **noSym** (*bool*) – When True symmetry generated parameters are not included. Do this so that redundant s.u.'s eare not shown. When False (default) s.u. values for all dependent parameters are placed in the returned dict.

exception GSASII.GSASIImapvars.**ConstraintException**

Defines an Exception that is used when an exception is raised processing constraints. Raised in *GenerateConstraints()* during sequential fits. Possible (but highly unlikely) to be raised in *CheckEquivalences()* (called by *GenerateConstraints()*) if an infinite loop is detected. Also raised in *GramSchmidtOrtho()* and *_SwapColumns()* but caught within *GenerateConstraints()*.

__weakref__

list of weak references to the object

GSASII.GSASIImapvars.**CountUserConstraints**()

Count the number of user-supplied constraints in use for the current refinement. Symmetry generated constraints are not counted as well as constraints that involve variables that are not varied.

This is used for CIF reporting.

Returns

an integer count

GSASII.GSASIImapvars.**Dict2Deriv**(*varyList*, *derivDict*, *dMdv*)

Compute derivatives for Independent Parameters from the derivatives for the original parameters

Parameters

- **varyList** (*list*) – a list of parameters names that will be varied
- **derivDict** (*dict*) – a dict containing derivatives for parameter values keyed by the parameter names.
- **dMdv** (*list*) – a Jacobian, as a list of np.array containing derivatives for dependent parameter computed from derivDict

GSASII.GSASIImapvars.**Dict2Map**(*parmDict*)

Applies the constraints defined using *StoreEquivalence()*, *GroupConstraints()* and *GenerateConstraints()* by changing values in a dict containing the parameters. This should be done after refinement and before the parameters are used for any computations

Parameters

parmDict (*dict*) – a dict containing parameter values keyed by the parameter names. After this is called, all the dependent variables will be updated based on constraints and equivalences.

GSASII.GSASIImapvars.**EvaluateMultipliers**(*constList*, **dicts*)

Convert multipliers for constraints and equivalences that are specified as strings into values. The strings can specify values in the parameter dicts as well as normal Python functions, such as "2*np.cos(0::Ax:2/2.)"

Parameters

- **constList** (*list*) – a list of dicts containing constraint expressions

- ***dicts** – one or more dicts containing GSAS-II parameters and their values can be specified

Returns

an empty string if there were no errors, or an error message listing the strings that could not be converted.

GSASII.GSASIImapvars.**GenerateConstraints**(*varyList*, *constrDict*, *fixedList*, *parmDict=None*, *seqHistNum=None*, *raiseException=False*)

Takes a list of relationship entries that have been stored by *ProcessConstraints()* into lists `constrDict` and `fixedList`

This routine then calls *CheckEquivalences()* for internal consistency. This includes converting equivalenced variables into constraints when a variable is used in both.

Once checked, parameters are grouped so that any parameters that are used in more than one constraint are grouped together. This allows checking for incompatible logic (for example, when four constraints are specified for three variables).

If parmDict is not None, the parameter groups are checked for constraints where some parameters are varied, but not others. If so, the value for that unvaried parameter is subtracted from the constant in the constraint.

Once all checks are complete, the constraints are then converted to the form used to apply them, saving them as global variables within this module.

Parameters

- **varyList** (*list*) – a list of parameters names (strings of form `<ph>:<hst>:<nam>`) that will be varied. Note that this is changed here unless set to None. None is used to indicate that all constraints should be generated.
- **constrDict** (*dict*) – a list of dicts defining relationships/constraints (as described in *GroupConstraints()*)
- **fixedList** (*list*) – a list of values specifying a fixed value for each dict in constrDict. Values are either strings that can be converted to floats, float values or None if the constraint defines a new parameter.
- **parmDict** (*dict*) – a dict containing all parameters defined in current refinement.
- **seqHistNum** (*int*) – the hId number of the current histogram in a sequential fit. None (default) otherwise.
- **raiseException** (*bool*) – When True, generation of an error causes an exception to be raised (used in sequential fits)

Returns

errmsg,warning,groups,parmlist

errmsg
Is an error message or empty if no errors were found

warning
Is a warning message about constraints that have been ignored or changed

groups
Lists parameter groups

parmlist
Lists parameters in each parameter groups

GSASII.GSASIImapvars.**GetDependentVars**(*opt=None*)

Return a list of dependent variables: e.g. parameters that are constrained in terms of other parameters

Parameters
opt (*str*) – type of dependent variables. 'equiv': from equivalences, 'constr': from constraints None (default): all

Returns
a list of parameter names

GSASII.GSASIImapvars.**GetDroppedSym**(*seqmode*, *seqhistnum*)

Return automatically generated (equivalence) relationships that were converted to constraint equations

Returns
a list of strings containing the details of the equivalences

GSASII.GSASIImapvars.**GetIndependentVars**()

Return a list of independent variables: e.g. parameters that are slaved to other parameters by constraints

Returns
a list of parameter names

GSASII.GSASIImapvars.**GetSymEquiv**(*seqmode*, *seqhistnum*)

Return the automatically generated (equivalence) relationships.

Returns
a list of strings containing the details of the contraint relationships

GSASII.GSASIImapvars.**GramSchmidtOrtho**(*a*, *nkeep=0*)

Use the Gram-Schmidt process (http://en.wikipedia.org/wiki/Gram-Schmidt) to find orthonormal unit vectors relative to first row.

If nkeep is non-zero, the first nkeep rows in the array are not changed

input:
arrayin: a 2-D non-singular square array

returns:
a orthonormal set of unit vectors as a square array

GSASII.GSASIImapvars.**GroupConstraints**(*constrDict*)

Divide the constraints into groups that share no parameters.

Parameters
constrDict (*dict*) – a list of dicts defining relationships/constraints

```
constrDict = [{<constr1>}, {<constr2>}, ...]
```

where {<constr1>} is {'var1': mult1, 'var2': mult2,… }

Returns

two lists of lists:

- a list of grouped contraints where each constraint grouped containts a list of indices for constraint constrDict entries
- a list containing lists of parameter names contained in each group

GSASII.GSASIImapvars.**InitVars**()

Initializes all constraint information

GSASII.GSASIImapvars.**Map2Dict**(*parmDict*, *varyList*)

Updates the parameter dictionary and the varyList using the equivalence and constraint input. This should be called at least once, after the constraints have been defined using *StoreEquivalence()*, *GroupConstraints()* and *GenerateConstraints()* and before any parameter refinement is done.

This completes the parameter dictionary by defining values for parameters created by constraints based on the constraints that define them using the values for the current parameters. It also removes all dependent variables from the varyList

Parameters

- **parmDict** (*dict*) – a dict containing parameter values keyed by the parameter names. For new variables created by constraints, entries will be added to the dictionary, if not already present, or the values will be recomputed.
- **varyList** (*list*) – a list of parameters names. Will be modified.

GSASII.GSASIImapvars.**ProcessConstraints**(*constList*, *seqmode='use-all'*, *seqhst=None*)

Interpret the constraints in the constList input into a dictionary, etc. All GSASIIobj.G2VarObj objects are mapped to the appropriate phase/hist/atoms based on the object internals (random Ids). If this can't be done (if a phase has been deleted, etc.), the variable is ignored. If the constraint cannot be used due to too many dropped variables, it is counted as ignored. In the case of sequential refinements, the current histogram number is substituted for a histogram number of "*".

NB: this processing does not include symmetry imposed constraints

Parameters

- **constList** (*list*) – a list of lists where each item in the outer list specifies a constraint of some form, as described in the GSASIIobj *Constraint definitions*.
- **seqmode** (*str*) – one of 'use-all', 'wildcards-only' or 'auto-wildcard'. When seqmode=='wildcards-only' then any constraint with a numerical histogram number is skipped. With seqmode=='auto-wildcard', any non-null constraint number is set to the selected histogram.
- **seqhst** (*int*) – number for current histogram (used for 'wildcards-only' or 'auto-wildcard' only). Should be None for non-sequential fits.

Returns

a tuple of (constrDict,fixedList,ignored) where:

- constrDict (list of dicts) contains the constraint relationships
- fixedList (list) contains the fixed values for each type of constraint.
- ignored (int) counts the number of invalid constraint items (should always be zero!)

GSASII.GSASIImapvars.**StoreEquivalence**(*independentVar*, *dependentList*, *symGen=True*)

Takes a list of dependent parameter(s) and stores their relationship to a single independent parameter (independentVar).

Called with user-supplied constraints by *ProcessConstraints()*, with Pawley constraints from GSASIIstrIO.GetPawleyConstr(), with Unit Cell constraints from GSASIIstrIO.cellVary() with symmetry-generated atom constraints from GSASIIstrIO.GetPhaseData()

There is no harm in using StoreEquivalence with the same independent variable:

```
StoreEquivalence('x',('y',))
StoreEquivalence('x',('z',))
```

but the same outcome can be obtained with a single call:

```
StoreEquivalence('x',('y','z'))
```

The latter will run more efficiently.

Note that mixing independent and dependent variables, such as:

```
StoreEquivalence('x',('y',))
StoreEquivalence('y',('z',))
```

is a poor choice. The module will attempt to fix this by transforming the equivalence to a "Const" constraint.

Parameters

- **independentVar** (*str*) – name of master parameter that will be used to determine the value to set the dependent variables
- **dependentList** (*list*) – a list of parameters that will set from independentVar. Each item in the list can be a string with the parameter name or a tuple containing a name and multiplier: `['::parm1',('::parm2',.5),]`

GSASII.GSASIImapvars.**StoreHold**(*var*, *holdType=None*)

Takes a variable name and prepares it to be removed from the refined variables.

Called with user-supplied constraints by *ProcessConstraints()*. At present symGen is not used, but could be set up to track Holds generated by symmetry.

GSASII.GSASIImapvars.**SubfromParmDict**(*s*, *prmDict*)

Process a string as a multiplier and convert it to a float value. This is done by subsituting any GSAS-II parameter names that appear in the string that have associated values in the parameter dict with the value for that parameter.

Parameters

- **s** (*str*) – a string to be converted to a value
- **prmDict** (*dict*) – a dictionary with keys as GSAS-II parameter names and values the corresponding parameter value.

Returns

the evaluated expression as a float.

GSASII.GSASIImapvars.**VarKeys**(*constr*)

Finds the keys in a constraint that represent parameters e.g. eliminates any that start with '_'

Parameters

constr (*dict*) – a single constraint entry of form:

```
{'var1': mult1, 'var2': mult2,... '_notVar': val,...}
```

(see *GroupConstraints()*)

Returns

a list of keys where any keys beginning with '_' are removed.

GSASII.GSASIImapvars.**VarRemapShow**(*varyList=None*, *inputOnly=False*, *linelen=60*)

List out the saved relationships. This should be done after the constraints have been defined using *StoreEquivalence()*, *GroupConstraints()* and *GenerateConstraints()*.

Returns
a string containing the details of the contraint relationships

GSASII.GSASIImapvars.**VarRemapSumm**()

Summarize the constraints in a single line

Returns
a string summarizing the contraint relationships. May be blank

GSASII.GSASIImapvars.**_FillArray**(*sel*, *d*, *collist*)

Construct a n by n matrix [n = len(collist)] with the initial m rows [m = len(sel)] using the relationships defined in the expressions dict, d. Since m may be smaller than n, the remaining rows are filled with rows that are tested to not create a singular matrix.

Parameters

- **sel** (*list*) – a list of indices in dict d
- **d** (*list*) – a list of dict's where each dict describes an expression from a constraint equation or a new var
- **collist** (*list*) – a list parameter names.

Returns
an n by n numpy.array matrix

GSASII.GSASIImapvars.**_FormatConstraint**(*RelDict*, *RelVal*)

Formats a Constraint or Function for use in a convenient way

GSASII.GSASIImapvars.**_RowEchelon**(*m*, *arr*, *collist*)

Convert the first m rows in Matrix arr to row-echelon form exchanging columns in the matrix and collist as needed.

throws an exception if the matrix is singular because the first m rows are not linearly independent

GSASII.GSASIImapvars.**_SwapColumns**(*i*, *m*, *v*)

Swap columns in matrix m as well as the labels in v so that element (i,i) is replaced by the first non-zero element in row i after that element

Throws an exception if there are no non-zero elements in that row

GSASII.GSASIImapvars.**_showEquiv**(*varlist*, *mapvars*, *invmultarr*, *longmsg=False*)

Format an equivalence relationship, note that varlist, mapvars, invmultarr are elements of dependentParmList, indParmList, invarrayList

GSASII.GSASIImapvars.**arrayList = []**

a list of of relationship matrices that map model parameters in each constraint group (in *dependentParmList*) to generated (New Var) parameters.

GSASII.GSASIImapvars.**consNum = 0**

The number to be assigned to the next constraint to be created

GSASII.GSASIImapvars.**constrParms = {'dep-constr': [], 'dep-equiv': [], 'indep-constr': [], 'indep-equiv': []}**

A dict with parameters in equivalences, compiled from (*dependentParmList*) and (*indParmList*). Used within *GetIndependentVars()* and *GetDependentVars()*.

GSASII.GSASIImapvars.**constrVarList = []**

List of parameters used in "Constr" and "New Var" constraints

GSASII.GSASIImapvars.**convVarList = []**

parameters in equivalences that were converted to "Const" constraints

GSASII.GSASIImapvars.**depVarList = []**

A list of all dependent parameters in equivalences

GSASII.GSASIImapvars.**dependentParmList = []**

a list of lists where each item contains a list of parameters in each constraint group. note that parameters listed in dependentParmList should not be refined directly.

GSASII.GSASIImapvars.**droppedSym = []**

A list of symmetry generated equivalences that have been converted to constraint equations in *CheckEquivalences()*

GSASII.GSASIImapvars.**getConstrError**(*constrLst*, *seqmode*, *seqhst*)

This is used to display error messages for constraints and equivalence relations

Parm list constrLst
a single constraint or equivalence as saved in the data tree (see *constraint definitions*).

Parameters

- **seqmode** (*str*) – one of 'use-all', 'wildcards-only' or 'auto-wildcard'
- **seqhst** (*int*) – number for current histogram (used for 'wildcards-only' or 'auto-wildcard' only). Should be None for non-sequential fits.

Returns
error, msg where error (bool) is True if the constraint/equivalence creates an error, msg (str) can be a warning or an error

GSASII.GSASIImapvars.**getInvConstraintEq**(*var*, *varyList*)

For a dependent variable, find the constraint that defines the dependent variable in terms of varied independent variables. This works for constraint equations (via new var or generated parameters) or equivalences. For equivalences the result will lists of length 1

Parameters

- **var** (*str*) – named of refined variable (e.g. 0:0:Scale)
- **varyList** (*list*) – list of refined variables

Returns
vList,mList where vList is a list of variables and mList is a list of multipliers for that variable (floats)

GSASII.GSASIImapvars.**groupErrors = []**

parameters in constraints where parameter grouping and matrix inversion fails

GSASII.GSASIImapvars.**holdParmList = []**

List of parameters that should not be refined ("Hold"s). Set in *StoreHold()*. Initialized in *InitVars()*.

GSASII.GSASIImapvars.**holdParmType = {}**

The reason why a parameter has been marked as "Hold". Initialized in *InitVars()*; set in *StoreHold()*.

GSASII.GSASIImapvars.**indParmList = []**

a list of lists where each item contains a list for each constraint group with fixed values for constraint equations and names of generated/New Var parameters. In the case of equivalences, the name of a single independent parameter is stored.

GSASII.GSASIImapvars.**indepVarList = []**

A list of all independent parameters in equivalences

GSASII.GSASIImapvars.**invarrayList = []**

a list of of inverse-relationship matrices that map constrained values and generated (New Var) parameters (in *indParmList*) to model parameters (in *dependentParmList*).

GSASII.GSASIImapvars.**multdepVarList = []**

parameters used as dependents multiple times in equivalences

GSASII.GSASIImapvars.**normParms**(*parmDict*)

Attempt to put parameters into the right ballpark by scaling to enforce constraint equations

GSASII.GSASIImapvars.**paramPrefix = '::constr'**

A prefix for generated parameter names

GSASII.GSASIImapvars.**saveVaryList = []**

A list of the varied parameters that was last supplied when constraints were processed. This is set in *GenerateConstraints()* and updated in *Map2Dict()*. Used in *VarRemapShow()*

GSASII.GSASIImapvars.**symGenList = []**

A list of flags that if True indicates a constraint was generated by symmetry

GSASII.GSASIImapvars.**undefinedVars = []**

parameters used in equivalences that are not defined in the parameter dict

GSASII.GSASIImapvars.**unvariedParmsList = []**

parameters used in equivalences that are not varied

CHAPTER

TWELVE

GSASIIIMAGE: IMAGE CALC MODULE

12.1 Summary/Contents

Image calibration, masking & image integration routines.

Note that the GSAS-II image coordinate system is defined as follows: standing facing the x-ray source (e.g. behind the beamstop), the synchrotron ring will be to the left (for a left handed synchrotron – almost all are left handed). That left-right direction defines X. Y is up and thus Z is toward the source/sample. The resulting 2D image is then viewed from the sample position (e.g. between the x-ray source and the detector). The detector is addressed in units of pixels (or distances using the pixel size) with the origin as the lower left corner. The beam center is measured from this point; usually somewhere near the center of the image, thus both Xc & Yc will be greater than zero unless the beam center is not on the image. Note that when images are displayed in image viewers, most software puts the origin in the upper left corner.

Section Contents

12.2 GSASIIimage Routines

Classes and routines defined in `GSASIIimage` follow.

`GSASII.GSASIIimage.`**`AutoPixelMask`**(*Image*, *Masks*, *Controls*, *numChans*, *dlg=None*)

Find "bad" regions on an image and creata a pixel mask to remove them. This works by masking pixels that are well outside the range of the median at that radial distance. This is ~4x faster than the original version from RBVD. Developed by Howard Yanxon, Wenqian Xu and James Weng.

Called from GSASIIimgGUI.UpdateMasks.OnFindPixelMask (single image) and GSASIIimgGUI.UpdateMasks.OnAutoFindPixelMask (multiple images) [see `GSASIIimgGUI.UpdateMasks()`]

Parameters

- **`Image`** (*`np.array`*) – 2D data structure describing a diffaction image
- **`Masks`** (*`dict`*) – contents of Masks data tree
- **`Controls`** (*`dict`*) – diffraction & calibration parameters for image from IMG data tree entry
- **`numChans`** (*`int`*) – number of channels in eventual 2theta pattern after integration

- **dlg** (*wx.Dialog*) – a widget that can be used to show the status of the pixel mask scan and can optionally be used to cancel the scan. If dlg=None then this is ignored (for non-GUI use).

Returns
a mask array with the same shape as Image or None if the the scan is cancelled from the dlg Dialog.

GSASII.GSASIIimage.**AzimuthIntegrate**(*image*, *data*, *masks*, *ringId*, *blkSize=1024*)

Integrate by azimuth around the ring masked region in 0.5 deg steps

GSASII.GSASIIimage.**DoPolaCalib**(*ImageZ*, *imageData*, *arcTth*)

Determine image polarization by successive integrations with & without preset arc mask. After initial search, does a set of five with offset azimuth to get mean(std) result.

GSASII.GSASIIimage.**EdgeFinder**(*image*, *data*)

this makes list of all x,y where I>edgeMin suitable for an ellipse search? Not currently used but might be useful in future?

GSASII.GSASIIimage.**FastAutoPixelMask**(*Image*, *Masks*, *Controls*, *numChans*, *dlg=None*)

Find "bad" regions on an image and create a pixel mask to remove them. This works by masking pixels that are m*sigma outside the range of the median at that radial distance using the using the fmask C module (based on the AIRXD C++ code https://github.com/AdvancedPhotonSource/AIRXD-ML-PUB, developed by Howard Yanxon, Wenqian Xu and James Weng.)

This is much faster than AutoPixelMask, which does pretty much the same computation, but uses pure Python/numpy code.

Called from GSASIIimgGUI.UpdateMasks.OnFindPixelMask (single image) and GSASIIimgGUI.UpdateMasks.OnAutoFindPixelMask (multiple images) [see `GSASIIimgGUI.UpdateMasks()`]

Parameters

- **Image** (*np.array*) – 2D data structure describing a diffaction image
- **Masks** (*dict*) – contents of Masks data tree
- **Controls** (*dict*) – diffraction & calibration parameters for image from IMG data tree entry
- **numChans** (*int*) – number of channels in eventual 2theta pattern after integration

Returns
a bool mask array with the same shape as Image

GSASII.GSASIIimage.**Fill2ThetaAzimuthMap**(*masks*, *TAr*, *tam*, *image*, *ringMask=False*)

Makes masked intensity correction arrays that depend on image intensity, 2theta and azimuth. Masking is generated from the combination of the following: an array previously generated by *MakeMaskMap()* combined with Thresholds, Rings and Arcs mask input.

These correction arrays are generated for a rectangular section of an image (must be 1024x1024 or smaller) where the size is determined the input arrays.

Note that older, less optimized, code has been left commented out below in case there are future problems or questions.

Parameters

- **masks** (*dict*) – GSAS-II mask settings
- **TAr** (*np.array*) – 2theta/azimuth/correction arrays, reshaped
- **tam** (*np.array*) – mask array from *MakeMaskMap()*
- **image** (*np.array*) – image array

Returns
a list of 4 masked arrays with values for: azimuth, 2-theta, intensity/polarization, dist**2/d0**2

GSASII.GSASIIimage.**FitDetector**(*rings*, *varyList*, *parmDict*, *Print=True*, *covar=False*)

Fit detector calibration parameters

Parameters

- **rings** (*np.array*) – vector of ring positions
- **varyList** (*list*) – calibration parameters to be refined
- **parmDict** (*dict*) – all calibration parameters
- **Print** (*bool*) – set to True (default) to print the results
- **covar** (*bool*) – set to True to return the covariance matrix (default is False)

Returns
[chisq,vals,sigList] unless covar is True, then [chisq,vals,sigList,coVarMatrix] is returned

GSASII.GSASIIimage.**FitImageSpots**(*Image*, *ImMax*, *ind*, *pixSize*, *nxy*, *spotSize=1.0*)

Used with "s" key in image plots to search for spot masks

GSASII.GSASIIimage.**FitMultiDist**(*rings*, *varyList*, *parmDict*, *keyArray*, *Print=True*, *covar=False*, *progressDlg=None*)

Fit detector calibration parameters with multi-distance data

Parameters

- **rings** (*np.array*) – vector of ring positions (x,y,d-space)
- **varyList** (*list*) – calibration parameters to be refined
- **parmDict** (*dict*) – calibration parameters
- **keyArray** (*list*) – key to look up distances for each ring position. Length must be the same as rings.
- **Print** (*bool*) – set to True (default) to print the results
- **covar** (*bool*) – set to True to return the covariance matrix (default is False)
- **progressDlg** – wx.ProgressDialog object or None (default)

Returns
[chisq,vals,sigDict] unless covar is True, then [chisq,vals,sigDict,coVarMatrix] is returned

GSASII.GSASIIimage.**FitStrSta**(*Image*, *StrSta*, *Controls*)

Needs a doc string

GSASII.GSASIIimage.**FitStrain**(*rings*, *p0*, *dset*, *wave*, *phi*, *StaType*)

Fits external strain tensor from distortion of Bragg rings in images

GSASII.GSASIIimage.**GetAzm**(*x*, *y*, *data*)

Give azimuth value for detector x,y position; calibration info in data

GSASII.GSASIIimage.**GetDetectorXY**(*dsp*, *azm*, *data*)

Get detector x,y position from d-spacing (dsp), azimuth (azm,deg) & image controls dictionary (data) - new version

GSASII.GSASIIimage.**GetDsp**(*x*, *y*, *data*)

Give d-spacing value for detector x,y position; calibration info in data

GSASII.GSASIIimage.**GetEllipse**(*dsp*, *data*)

uses Dandelin spheres to find ellipse or hyperbola parameters from detector geometry as given in image controls dictionary (data) and a d-spacing (dsp)

GSASII.GSASIIimage.**GetEllipse2**(*tth*, *dxy*, *dist*, *cent*, *tilt*, *phi*)

uses Dandelin spheres to find ellipse or hyperbola parameters from detector geometry on output radii[0] (b-minor axis) set < 0. for hyperbola

GSASII.GSASIIimage.**GetTth**(*x*, *y*, *data*)

Give 2-theta value for detector x,y position; calibration info in data

GSASII.GSASIIimage.**GetTthAzm**(*x*, *y*, *data*)

Give 2-theta, azimuth values for detector x,y position; calibration info in data

GSASII.GSASIIimage.**GetTthAzmDsp**(*x*, *y*, *data*)

Computes a 2theta, etc. from a detector position and calibration constants - checked OK for ellipses & hyperbola. Use for detector 2-theta != 0.

Returns

np.array(tth,azm,dsp) where tth is 2theta, azm is the azimutal angle, and dsp is the d-space - only used for non-zero detector 2-thetas

GSASII.GSASIIimage.**GetTthAzmDsp2**(*x*, *y*, *data*)

Computes a 2theta, etc. from a detector position and calibration constants - checked OK for ellipses & hyperbola. Use only for detector 2-theta = 0

Returns

np.array(tth,azm,dsp) where tth is 2theta, azm is the azimutal angle, and dsp is the d-space - not used in integration

GSASII.GSASIIimage.**GetTthAzmG**(*x*, *y*, *data*)

Give 2-theta, azimuth & geometric corr. values for detector x,y position; calibration info in data - only used in integration for detector 2-theta != 0. checked OK for ellipses & hyperbola This is the slow step in image integration

GSASII.GSASIIimage.**GetTthP**(*x*, *y*, *parmDict*, *dist=None*, *detX=None*, *detY=None*)

Compute angle between detector normal & sample scattering ray vector For Multi-distance use, dist, detX & detY change for different detector positions.

Parameters

- **x** (*float/array*) – detector x-position in mm
- **y** (*float/array*) – detector y-position in mm
- **parmDict** (*dict*) – dictionary of detector orientation parameters in fitting routine. Used entries are: det-X, det-Y, tilt, dist & phi In multi-dist, used entries are: tilt & phi, and the following are specified:
- **dist** (*float/array*) – detector distance
- **detX** (*float/array*) – X Position of beam center
- **detY** (*float/array*) – Y Position of beam center

Returns

float/array angle: = 2-theta if tilt is zero

GSASII.GSASIIimage.**ImageCalibrate**(*G2frame*, *data*)

Called to perform an initial image calibration after points have been selected for the inner ring.

Called only from `OnImRelease` (mouse release) in `GSASIIplot.PlotImage()`, thus expected to be used from GUI only (not scripted)

GSASII.GSASIIimage.**ImageCompress**(*image*, *scale*)

Reduces size of image by selecting every n'th point param: image array: original image param: scale int: intervsl between selected points returns: array: reduced size image

GSASII.GSASIIimage.**ImageIntegrate**(*image*, *data*, *masks*, *blkSize=128*, *returnN=False*, *useTA=None*, *useMask=None*)

Integrate an image; called from `OnIntegrate()` and `OnIntegrateAll()` inside `GSASIIimgGUI.UpdateImageControls()` as well as `GSASIIscriptable.G2Image.Integrate()`.

Parameters

- **image** (*np.array*) – contains the 2-D image
- **data** (*np.array*) – specifies controls/calibration parameters for an image
- **masks** (*np.array*) – specifies masks parameters for an image
- **blkSize** (*int*) – a blocksize that is selected for speed
- **returnN** (*bool*) – If True, causes an extra matrix (NST) to be returned. The default is False.
- **useTA** (*np.array*) – contains a cached set of blocked 2theta/azimuth/correction matrices (see *MakeUseTA()*) for the current image. The default, None, causes this to be computed as needed.
- **useMask** (*np.array*) – contains a cached set of blocked masks (see *MakeUseMask()*) for the current image. The default, None, causes this to be computed as needed.

Returns

list ints, azms, Xvals, cancel (or ints, azms, Xvals, NST, cancel if returnN is True), where azms is a list of `M` azimuth values that were requested for integration, ints is a list `M` arrays of diffraction intensities (where each array of diffraction data is length `N`), Xvals is an array of "x" values, 2theta, Q, log(q) (determined by data['binType']), also of length `N`. Variable cancel will always be False, since a status window is no longer supported.

GSASII.GSASIIimage.**ImageLocalMax**(*image*, *w*, *Xpix*, *Ypix*)

Needs a doc string

GSASII.GSASIIimage.**ImageRecalibrate**(*G2frame*, *ImageZ*, *data*, *masks*, *getRingsOnly=False*)

Called to repeat the calibration on an image, usually called after calibration is done initially to improve the fit, but also can be used after reading approximate calibration parameters, if they are close enough that the first ring can be found.

Parameters

- **G2frame** – The top-level GSAS-II frame or None, to skip plotting
- **ImageZ** (*np.Array*) – the image to calibrate
- **data** (*dict*) – the Controls dict for the image
- **masks** (*dict*) – a dict with masks

Returns

a list containing vals,varyList,sigList,parmDict,covar or rings (with an array of x, y, and d-space values) if getRingsOnly is True or an empty list, in case of an error

GSASII.GSASIIimage.**Make2ThetaAzimuthMap**(*data*, *iLim*, *jLim*)

Makes a set of matrices that provide the 2-theta, azimuth and geometric correction values for each pixel in an image taking into account the detector orientation. Can be used for the entire image or a rectangular section of an image (determined by iLim and jLim).

This is used in two ways. For image integration, the computation is done over blocks of fixed size (typically 128 or 256 pixels) but for pixel mask generation, the two-theta matrix for all pixels is computed. Note that for integration, this routine will be called to generate sections as needed or may be called by *MakeUseTA()*, which creates all sections at once, so they can be reused multiple times.

Parameters

- **data** (*dict*) – GSAS-II image data object (describes the image)
- **iLim** (*list*) – boundary along x-pixels
- **jLim** (*list*) – boundary along y-pixels

Returns

TA, array[4,nI,nJ]: 2-theta, azimuth, 2 geometric corrections

GSASII.GSASIIimage.**MakeMaskMap**(*data*, *masks*, *iLim*, *jLim*)

Makes a mask array from masking parameters that are not determined by image calibration parameters or the image intensities. Thus this uses mask Frames, Polygons, Points, and Lines settings (but not Thresholds, Rings or Arcs). Used on a rectangular section of an image (must be 1024x1024 or smaller) where the size is determined by iLim and jLim.

Parameters

- **data** (*dict*) – GSAS-II image data object (describes the image)
- **iLim** (*list*) – boundary along x-pixels
- **jLim** (*list*) – boundary along y-pixels

Returns

array[nI,nJ] TA: X, Y

GSASII.GSASIIimage.**MakeUseMask**(*data*, *masks*, *blkSize=128*)

Precomputes a set of blocked mask arrays for the mask elements that do not depend on the instrument controls (see *MakeMaskMap()*). This computation is done optionally, but provides speed as the results from this can be cached to avoid recomputation for a series of images with the same mask parameters.

Parameters

- **data** (*np.array*) – specifies mask parameters for an image
- **blkSize** (*int*) – a blocksize that is selected for speed

Returns

a list of TA blocks

GSASII.GSASIIimage.**MakeUseTA**(*data*, *blkSize=128*)

Precomputes the set of blocked arrays for 2theta-azimuth mapping from the controls settings of the current image for image integration. This computation is done optionally, but provides speed as the results from this can be cached to avoid recomputation for a series of images with the same calibration parameters.

Parameters

- **data** (*np.array*) – specifies parameters for an image
- **blkSize** (*int*) – a blocksize that is selected for speed

Returns

a list of TA blocks

GSASII.GSASIIimage.**SamAbs**(*data*, *tax*, *tay*, *muT*)

Compute sample absorption correction for images

GSASII.GSASIIimage.**TestFastPixelMask**()

Test if the fast (C) version of Auto Pixel Masking is available.

Returns
True if the airxd.mask package can be imported; False otherwise.

GSASII.GSASIIimage.**calcFij**(*omg*, *phi*, *azm*, *th*)

Uses parameters as defined by Bob He & Kingsley Smith, Adv. in X-Ray Anal. 41, 501 (1997)

Parameters

- **omg** – his omega = sample omega rotation; 0 when incident beam || sample surface, 90 when perp. to sample surface
- **phi** – his phi = sample phi rotation; usually = 0, axis rotates with omg.
- **azm** – his chi = azimuth around incident beam
- **th** – his theta = theta

GSASII.GSASIIimage.**checkEllipse**(*Zsum*, *distSum*, *xSum*, *ySum*, *dist*, *x*, *y*)

Needs a doc string

GSASII.GSASIIimage.**ellipseCalcD**(*B*, *xyd*, *varyList*, *parmDict*, *keyArray=None*, *progressDlg=None*)

Compute the deviations from the ellipse point locations

GSASII.GSASIIimage.**makeMat**(*Angle*, *Axis*)

Make rotation matrix from Angle and Axis

Parameters

- **Angle** (*float*) – in degrees
- **Axis** (*int*) – 0 for rotation about x, 1 for about y, etc.

GSASII.GSASIIimage.**makeRing**(*dsp*, *ellipse*, *pix*, *reject*, *scalex*, *scaley*, *image*, *mul=1*)

Needs a doc string

GSASII.GSASIIimage.**peneCorr**(*tth*, *dep*, *dist*)

Compute empirical position correction due to detector absorption

Parameters

- **tth** (*float/array*) – angle between detector normal & scattered ray vector
- **dep** (*float*) – coefficient
- **dist** (*float*) – sample to detector surface in mm

Returns
float/array distance: correction for penetration

GSASII.GSASIIimage.**polymask**(*data*, *Poly*, *Spots=[]*)

Applies spot(point), polygon & frame masks via calls to matplotlib routines; should be called only once each during image processing. A separate call is used for a frame. Individual masked blocks are then pulled from the output array.

Parameters

- **data** (*dict*) – GSAS-II image data object (describes the image)
- **Poly** (*list*) – list of polygons; if empty, returns None
- **Spots** (*list*) – list of spots/points; if empty, returns None

Returns

Zimg, array[Nx,Ny] size of full image mask for all polygons/spots or frame considered

CHAPTER

THIRTEEN

GSASIIMATH: COMPUTATION MODULE

13.1 Summary/Contents

Least-squares minimization and misc. computation routines.

Section Contents

13.2 GSASIImath Classes and routines

Routines defined in `GSASIImath` follow.

`GSASII.GSASIImath.`**`AV2Q`**(*A*, *V*)

convert angle (radians) & vector to quaternion q=r+ai+bj+ck

`GSASII.GSASIImath.`**`AVdeg2Q`**(*A*, *V*)

convert angle (degrees) & vector to quaternion q=r+ai+bj+ck

`GSASII.GSASIImath.`**`AddHydrogens`**(*AtLookUp*, *General*, *Atoms*, *AddHydId*)

Add hydrogen atoms according to "standard" geometry & updates positions upon request if atoms move. Thermal parmameters are set at 1.1-1.5 * atom U distances adapted from ITC C 9.5.1.1

`GSASII.GSASIImath.`**`ApplyModeDisp`**(*data*)

Applies ISODISTORT mode displacements to atom lists. This changes the contents of the Draw Atoms positions and the Atoms positions.

Parameters

`data` (*dict*) – the contents of the Phase data tree item for a particular phase

`GSASII.GSASIImath.`**`ApplyModulation`**(*data*, *tau*)

Applies modulation to drawing atom positions & Uijs for given tau

`GSASII.GSASIImath.`**`ApplySeqData`**(*data*, *seqData*, *PF2=False*)

Applies result from seq. refinement to drawing atom positions & Uijs

Parameters

- **`data`** (*dict*) – GSAS-II phase data structure

- **seqData** (*dict*) – GSAS-II sequential refinement results structure
- **PF2** (*bool*) – if True then seqData is from a sequential run of PDFfit2

Returns
list drawAtoms: revised Draw Atoms list

GSASII.GSASIImath.**AtomsCollect**(*data*, *Ind*, *Sel*)

Finds the symmetry set of atoms for those selected. Selects the one closest to the selected part of the unit cell for the selected atoms

Parameters

- **data** – the phase data structure
- **Ind** (*list*) – list of selected atom indices
- **Sel** (*int*) – an index with the selected plane or location in the unit cell to find atoms closest to

Returns
the list of unique atoms where selected atoms may have been replaced. Anisotropic Uij's are transformed

GSASII.GSASIImath.**BessIn**(*nmax*, *x*)

compute modified Bessel function I(n,x) from scipy routines & recurrance relation returns sequence of I(n,x) for n in range [-nmax…0…nmax]

Parameters

- **nmax** (*integer*) – maximul order for In(x)
- **x** (*float*) – argument for In(x)

Returns numpy array
[I(-nmax,x)…I(0,x)…I(nmax,x)]

GSASII.GSASIImath.**BessJn**(*nmax*, *x*)

compute Bessel function J(n,x) from scipy routine & recurrance relation returns sequence of J(n,x) for n in range [-nmax…0…nmax]

Parameters

- **nmax** (*integer*) – maximul order for Jn(x)
- **x** (*float*) – argument for Jn(x)

Returns numpy array
[J(-nmax,x)…J(0,x)…J(nmax,x)]

GSASII.GSASIImath.**CalcAngle**(*angle_dict*, *angle_atoms*, *parmDict*)

Used in class:*GSASIIobj.ExpressionCalcObj* to compute bond angles when defined in an expression.

GSASII.GSASIImath.**CalcAngleDeriv**(*angle_dict*, *angle_atoms*, *parmDict*)

Used to compute s.u. values on angles tracked in the sequential results table

GSASII.GSASIImath.**CalcDist**(*distance_dict*, *distance_atoms*, *parmDict*)

Used in class:*GSASIIobj.ExpressionCalcObj* to compute bond distances when defined in an expression.

GSASII.GSASIImath.**CalcDistDeriv**(*distance_dict*, *distance_atoms*, *parmDict*)

Used to compute s.u. values on distances tracked in the sequential results table

GSASII.GSASIImath.**CalcIsoCoords**(*Phase*, *parmDict*, *covdata={}*)

Compute the coordinate positions from ISODISTORT displacement mode values Uncertainties are computed if covdata is supplied.

Parameters

- **Phase** (*dict*) – contents of tree entry for selected phase
- **parmDict** (*dict*) – a dict with values for the modes; note that in the parmDict from refinements the mode values are not normalized, but this assumes they are.
- **Phase** – full covariance information from tree

Returns

modeDict,posDict where modeDict contains pairs of mode values and mode s.u. values; posDict contains pairs of displacement values and their s.u. values.

GSASII.GSASIImath.**CalcIsoDisp**(*Phase*, *parmDict={}*, *covdata={}*)

Compute the ISODISTORT displacement variable values from the atomic coordinates, applying the p::dA?:n displacements if parmDict is supplied. Uncertainties are computed if covdata is supplied.

GSASII.GSASIImath.**Cart2Polar**(*X*, *Y*, *Z*)

convert Cartesian X,Y,Z to polar coordinates in deg R - radius, 0<=Az<=360: azimuth angle from X-axis, 0<=Pl<=180: polar angle from Z-axis All are same length arrays

GSASII.GSASIImath.**ChargeFlip**(*data*, *reflDict*, *pgbar*)

default doc string

Parameters

name (*type*) – description

Returns

type name: description

GSASII.GSASIImath.**Den2Vol**(*Elements*, *density*)

converts density to molecular volume

Parameters

- **Elements** (*dict*) – elements in molecular formula; each must contain Num: number of atoms in formula Mass: at. wt.
- **density** (*float*) – material density in gm/cm^3

Returns

float volume: molecular volume in A^3

GSASII.GSASIImath.**DrawAtomsReplaceByID**(*data*, *loc*, *atom*, *ID*)

Replace all atoms in drawing array with an ID matching the specified value

GSASII.GSASIImath.**El2EstVol**(*Elements*)

Estimate volume from molecular formula; assumes atom volume = 10A^3

Parameters

Elements (*dict*) – elements in molecular formula; each must contain Num: number of atoms in formula

Returns

float volume: estimate of molecular volume in A^3

GSASII.GSASIImath.**El2Mass**(*Elements*)

compute molecular weight from Elements

Parameters
Elements (*dict*) – elements in molecular formula; each must contain Num: number of atoms in formula Mass: at. wt.

Returns
float mass: molecular weight.

GSASII.GSASIImath.**FillAtomLookUp**(*atomData*, *indx*)

create a dictionary of atom indexes with atom IDs as keys

Parameters

- **atomData** (*list*) – Atom table to be used
- **indx** (*int*) – pointer to position of atom id in atom record (typically cia+8)

Returns
dict atomLookUp: dictionary of atom indexes with atom IDs as keys

GSASII.GSASIImath.**FindAllNeighbors**(*phase*, *FrstName*, *AtNames*, *notName=''*, *Orig=None*, *Short=False*, *searchType='Bond'*)

Find neighboring atoms Uses Bond search criteria unless searchType is set to non-default

GSASII.GSASIImath.**FindAtomIndexByIDs**(*atomData*, *loc*, *IDs*, *Draw=True*)

finds the set of atom array indices for a list of atom IDs. Will search either the Atom table or the drawAtom table.

Parameters

- **atomData** (*list*) – Atom or drawAtom table containting coordinates, etc.
- **loc** (*int*) – location of atom id in atomData record
- **IDs** (*list*) – atom IDs to be found
- **Draw** (*bool*) – True if drawAtom table to be searched; False if Atom table is searched

Returns
list indx: atom (or drawAtom) indices

GSASII.GSASIImath.**FindTrue**(*t*)

Returns ranges of indices of array t where t is True or False

Parameters

- **t** (*list*) – a list or 1-D array of bool values
- **Condition** (*bool*) – select if True values oa list of indices

Returns
two list of indices, one where all values in the range are True and the second where all the values are False.

GSASII.GSASIImath.**Fourier4DMap**(*data*, *reflDict*)

default doc string

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**FourierMap**(*data*, *reflDict*)

default doc string

Parameters
name (*type*) – description

Returns
type name: description

exception GSASII.GSASIImath.**G2NormException**

__weakref__
list of weak references to the object

GSASII.GSASIImath.**GetAngleSig**(*Oatoms*, *Atoms*, *Amat*, *SGData*, *covData={}*)

Not used

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**GetAtomCoordsByID**(*pId*, *parmDict*, *AtLookup*, *indx*)

default doc string

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**GetAtomFracByID**(*pId*, *parmDict*, *AtLookup*, *indx*)

default doc string

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**GetAtomItemsById**(*atomData*, *atomLookUp*, *IdList*, *itemLoc*, *numItems=1*)

gets atom parameters for atoms using atom IDs

Parameters

- **atomData** (*list*) – Atom table to be used
- **atomLookUp** (*dict*) – dictionary of atom indexes with atom IDs as keys
- **IdList** (*list*) – atom IDs to be found
- **itemLoc** (*int*) – pointer to desired 1st item in an atom table entry
- **numItems** (*int*) – number of items to be retrieved

Returns
type name: description

GSASII.GSASIImath.**GetAtomMomsByID**(*pId*, *parmDict*, *AtLookup*, *indx*)

default doc string

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**GetAtomsById**(*atomData*, *atomLookUp*, *IdList*)

gets a list of atoms from Atom table that match a set of atom IDs

Parameters

- **atomData** (*list*) – Atom table to be used
- **atomLookUp** (*dict*) – dictionary of atom indexes with atom IDs as keys
- **IdList** (*list*) – atom IDs to be found

Returns
list atoms: list of atoms found

GSASII.GSASIImath.**GetDATSig**(*Oatoms*, *Atoms*, *Amat*, *SGData*, *covData={}*)

default doc string

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**GetDistSig**(*Oatoms*, *Atoms*, *Amat*, *SGData*, *covData={}*)

not used

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**GetMag**(*mag*, *Cell*)

Compute magnetic moment magnitude.

Parameters

- **mag** (*list*) – atom magnetic moment parms (must be magnetic!)
- **Cell** (*list*) – lattice parameters

Returns
moment magnitude as float

GSASII.GSASIImath.**GetMagDerv**(*mag*, *Cell*)

Compute magnetic moment derivatives numerically

Parameters

- **mag** (*list*) – atom magnetic moment parms (must be magnetic!)
- **Cell** (*list*) – lattice parameters

Returns
moment derivatives as floats

GSASII.GSASIImath.**GetSHCoeff**(*pId*, *parmDict*, *SHkeys*)

default doc string

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**GetTorsionSig**(*Oatoms*, *Atoms*, *Amat*, *SGData*, *covData={}*)

default doc string

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**GetXYZDist**(*xyz*, *XYZ*, *Amat*)

gets distance from position xyz to all XYZ, xyz & XYZ are np.array
and are in crystal coordinates; Amat is crystal to Cart matrix

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**HessianLSQ**(*func*, *x0*, *Hess*, *args=()*, *ftol=1.49012e-08*, *xtol=1e-06*, *maxcyc=0*, *lamda=-3*, *Print=False*, *refPlotUpdate=None*)

Minimize the sum of squares of a function (f) evaluated on a series of values (y): $\sum_{y=0}^{N_{obs}} f(y, args)$ where $x = argmin(\sum_{y=0}^{N_{obs}} (func(y)^2, axis = 0))$

Parameters

- **func** (*function*) – callable method or function should take at least one (possibly length N vector) argument and returns M floating point numbers.
- **x0** (*np.ndarray*) – The starting estimate for the minimization of length N
- **Hess** (*function*) – callable method or function A required function or method to compute the weighted vector and Hessian for func. It must be a symmetric NxN array
- **args** (*tuple*) – Any extra arguments to func are placed in this tuple.
- **ftol** (*float*) – Relative error desired in the sum of squares.
- **xtol** (*float*) – Relative tolerance of zeros in the SVD solution in nl.pinv.
- **maxcyc** (*int*) – The maximum number of cycles of refinement to execute, if -1 refine until other limits are met (ftol, xtol)
- **lamda** (*int*) – initial Marquardt lambda=10**lamda
- **Print** (*bool*) – True for printing results (residuals & times) by cycle

Returns

(x,cov_x,infodict) where

- x : ndarray The solution (or the result of the last iteration for an unsuccessful call).
- cov_x : ndarray Uses the fjac and ipvt optional outputs to construct an estimate of the jacobian around the solution. This matrix must be multiplied by the residual standard deviation to get the covariance of the parameter estimates – see curve_fit.

  – or None if a singular matrix encountered (indicates very flat curvature in direction), or some other error is encountered.

– or an empty array if a refinement was not performed because no valid variables were refined.

- infodict : dict, a dictionary of optional outputs with the keys:
  - 'fvec' : the function evaluated at the output
  - 'num cyc':
  - 'nfev': number of objective function evaluation calls
  - 'lamMax':
  - 'psing': list of variable variables that have been removed from the refinement
  - 'SVD0': -1 for singlar matrix, -2 for objective function exception, Nzeroes = # of SVD 0's
  - 'Hcorr': list entries (i,j,c) where i & j are of highly correlated variables & c is correlation coeff.

GSASII.GSASIImath.**HessianSVD**(*func*, *x0*, *Hess*, *args=()*, *ftol=1.49012e-08*, *xtol=1e-06*, *maxcyc=0*, *lamda=-3*, *Print=False*, *refPlotUpdate=None*)

Minimize the sum of squares of a function (f) evaluated on a series of values (y): $\sum_{y=0}^{N_{obs}} f(y, args)$ where $x = argmin(\sum_{y=0}^{N_{obs}} (func(y)^2, axis = 0))$

Parameters

- **func** (*function*) – callable method or function should take at least one (possibly length N vector) argument and returns M floating point numbers.
- **x0** (*np.ndarray*) – The starting estimate for the minimization of length N
- **Hess** (*function*) – callable method or function A required function or method to compute the weighted vector and Hessian for func. It must be a symmetric NxN array
- **args** (*tuple*) – Any extra arguments to func are placed in this tuple.
- **ftol** (*float*) – Relative error desired in the sum of squares.
- **xtol** (*float*) – Relative tolerance of zeros in the SVD solution in nl.pinv.
- **maxcyc** (*int*) – The maximum number of cycles of refinement to execute, if -1 refine until other limits are met (ftol, xtol)
- **Print** (*bool*) – True for printing results (residuals & times) by cycle

Returns

(x,cov_x,infodict) where

- x : ndarray The solution (or the result of the last iteration for an unsuccessful call).
- cov_x : ndarray Uses the fjac and ipvt optional outputs to construct an estimate of the jacobian around the solution. None if a singular matrix encountered (indicates very flat curvature in some direction). This matrix must be multiplied by the residual standard deviation to get the covariance of the parameter estimates – see curve_fit.
- infodict : dict a dictionary of optional outputs with the keys:
  - 'fvec' : the function evaluated at the output
  - 'num cyc':
  - 'nfev':
  - 'lamMax':0.
  - 'psing':

– 'SVD0':

GSASII.GSASIImath.**MagMod**(*glTau*, *xyz*, *modQ*, *MSSdata*, *SGData*, *SSGData*)

this needs to make magnetic moment modulations & magnitudes as fxn of gTau points; NB: this allows only 1 mag. wave fxn.

GSASII.GSASIImath.**MakeDrawAtom**(*data*, *atom*, *oldatom=None*)

needs a description

GSASII.GSASIImath.**Modulation**(*H*, *HP*, *nWaves*, *Fmod*, *Xmod*, *Umod*, *glTau*, *glWt*)

H: array nRefBlk x ops X hklt HP: array nRefBlk x ops X hklt proj to hkl nWaves: list number of waves for frac, pos, uij & mag Fmod: array 2 x atoms x waves (sin,cos terms) Xmod: array atoms X 3 X ngl Umod: array atoms x 3x3 x ngl glTau,glWt: arrays Gauss-Lorentzian pos & wts

GSASII.GSASIImath.**ModulationDerv**(*H*, *HP*, *Hij*, *nWaves*, *waveShapes*, *Fmod*, *Xmod*, *UmodAB*, *SCtauF*, *SCtauX*, *SCtauU*, *glTau*, *glWt*)

Compute Fourier modulation derivatives H: array ops X hklt proj to hkl HP: array ops X hklt proj to hkl Hij: array 2pi^2[a*^2h^2 b*^2k^2 c*^2l^2 a*b*hk a*c*hl b*c*kl] of projected hklm to hkl space

GSASII.GSASIImath.**ModulationTw**(*H*, *HP*, *nWaves*, *Fmod*, *Xmod*, *Umod*, *glTau*, *glWt*)

H: array nRefBlk x tw x ops X hklt HP: array nRefBlk x tw x ops X hklt proj to hkl Fmod: array 2 x atoms x waves (sin,cos terms) Xmod: array atoms X ngl X 3 Umod: array atoms x ngl x 3x3 glTau,glWt: arrays Gauss-Lorentzian pos & wts

GSASII.GSASIImath.**NCScattDen**(*Elements*, *vol*, *wave=0.0*)

Estimate neutron scattering density from molecular formula & volume; ignores valence, but includes anomalous effects

Parameters

- **Elements** (*dict*) – elements in molecular formula; each element must contain Num: number of atoms in formula Z: atomic number
- **vol** (*float*) – molecular volume in A^3
- **wave** (*float*) – optional wavelength in A

Returns

float rho: scattering density in 10^10cm^-2; if wave > 0 the includes f' contribution

Returns

float mu: if wave>0 absorption coeff in cm^-1 ; otherwise 0

Returns

float fpp: if wave>0 f" in 10^10cm^-2; otherwise 0

GSASII.GSASIImath.**OmitMap**(*data*, *reflDict*, *pgbar=None*)

default doc string

Parameters

name (*type*) – description

Returns

type name: description

GSASII.GSASIImath.**PeaksEquiv**(*data*, *Ind*)

Find the equivalent map peaks for those selected. Works on the contents of data['Map Peaks'].

Parameters

- **data** – the phase data structure

- **Ind** (*list*) – list of selected peak indices

Returns
augmented list of peaks including those related by symmetry to the ones in Ind

GSASII.GSASIImath.**PeaksUnique**(*data*, *Ind*, *Sel*, *dlg*)

Finds the symmetry unique set of peaks from those selected. Selects the one closest to the center of the unit cell. Works on the contents of data['Map Peaks']. Called from OnPeaksUnique in GSASIIphsGUI.py,

Parameters

- **data** – the phase data structure
- **Ind** (*list*) – list of selected peak indices
- **Sel** (*int*) – selected column to find peaks closest to
- **dlg** (*wx object*) – progress bar dialog box

Returns
the list of symmetry unique peaks from among those given in Ind

GSASII.GSASIImath.**Polar2Cart**(*R*, *Az*, *Pl*)

Convert polar vector in deg to Cartesian coordinates R - radius, 0<=Az<=360: azimuth angle from X-axis, 0<=Pl<=180: polar angle from Z-axis All are same length arrays

GSASII.GSASIImath.**Q2AV**(*Q*)

convert quaternion to angle (radians 0-2pi) & normalized vector q=r+ai+bj+ck

GSASII.GSASIImath.**Q2AVdeg**(*Q*)

convert quaternion to angle (degrees 0-360) & normalized vector q=r+ai+bj+ck

GSASII.GSASIImath.**Q2Mat**(*Q*)

make rotation matrix from quaternion q=r+ai+bj+ck

GSASII.GSASIImath.**QsymAxis**(*Q*, *symAxis*)

forms product of orientation Q vector & symAxis

Parameters

- **Q** (*list*) – quaternion
- **aymAxis** (*list*) – symmetry axis for rigid body: could be None

Returns
list Q: new quaternion

GSASII.GSASIImath.**RotPolbyM**(*R*, *Az*, *Pl*, *M*)

Rotate polar coordinates by rotation matrix

GSASII.GSASIImath.**RotPolbyQ**(*R*, *Az*, *Pl*, *Q*)

Rotate polar coordinates by quaternion

GSASII.GSASIImath.**RotateRBXYZ**(*Bmat*, *Cart*, *oriQ*, *symAxis=None*)

rotate & transform cartesian coordinates to crystallographic ones no translation applied. To be used for numerical derivatives

Parameters

- **Bmat** (*array*) – Orthogonalization matrix, see GSASIIlattice.cell2AB()
- **Cart** (*array*) – 2D array of coordinates
- **Q** (*array*) – quaternion as an np.array

- **symAxis** (*tuple*) – if not None (default), specifies the symmetry axis of the rigid body, which will be aligned to the quaternion vector.

Returns
2D array of fractional coordinates, without translation to origin

GSASII.GSASIImath.**SSChargeFlip**(*data*, *reflDict*, *pgbar*)

default doc string

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**SearchMap**(*generalData*, *drawingData*, *Neg=False*)

Does a search of a density map for peaks meeting the criterion of peak height is greater than mapData['cutOff']/100 of mapData['rhoMax'] where mapData is data['General']['mapData']; the map is also in mapData.

Parameters

- **generalData** – the phase data structure; includes the map
- **drawingData** – the drawing data structure
- **Neg** – if True then search for negative peaks (i.e. H-atoms & neutron data)

Returns

(peaks,mags,dzeros) where

- peaks : ndarray x,y,z positions of the peaks found in the map
- mags : ndarray the magnitudes of the peaks
- dzeros : ndarray the distance of the peaks from the unit cell origin
- dcent : ndarray the distance of the peaks from the unit cell center

GSASII.GSASIImath.**SetDefaultDData**(*dType*, *histoName*, *NShkl=0*, *NDij=0*)

Sets default values for various histogram parameters param: str dType: 3 letter histogram type, e.g. 'PNT' param: str histoName: histogram name as it aoears in tree param: NShkl int: number of generalized mustrain coefficients - depends on symmetry param: NDij int: number of hydrostatic strain coefficients - depends on symmetry

returns dict: default data for histogram - found in data tab for phase/histogram

GSASII.GSASIImath.**SetMolCent**(*model*, *RBData*)

default doc string

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**UpdateHKLFvals**(*histoName*, *phaseData*, *reflData*)

Update the flag field and the d-space field in HKLF reflection table.

Data tree contents are passed into this routine as data objects (strings, dicts,…) rather than data tree references so that this can called both from GUI routines as well as used in scripting.

This gets called from `GSASIIphsGUI.CheckAddHKLF()` (which gets called in OnHklfAdd, inside `GSASIIphsGUI.UpdatePhaseData()` and `GSASIIddataGUI.MakeHistPhaseWin()`). Also, in

GSASIIscriptable.G2Project.link_histogram_phase() and in routines OnImportPhase or OnImportSfact inside func:*GSASIIdataGUI.GSASIImain*.

GSASII.GSASIImath.**UpdateMCSAxyz**(*Bmat*, *MCSA*)

default doc string

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**UpdateRBUIJ**(*Bmat*, *Cart*, *RBObj*)

Transform TLS to Uij for RB objects

Parameters

- **Bmat** (*array*) – Orthogonalization matrix, see GSASIIlattice.cell2AB()
- **Cart** (*array*) – 2D array of coordinates wrt RB coordinates
- **RBObj** (*dict*) – rigid body selection/orientation information

Returns
list Uout : new atom thermal parameters

GSASII.GSASIImath.**UpdateRBXYZ**(*Bmat*, *RBObj*, *RBData*, *RBType*)

returns crystal coordinates for atoms described by RBObj. Note that RBObj['symAxis'], if present, determines the symmetry axis of the rigid body, which will be aligned to the quaternion direction.

Parameters

- **Bmat** (*np.array*) – see GSASIIlattice.cell2AB()
- **rbObj** (*dict*) – rigid body selection/orientation information
- **RBData** (*dict*) – rigid body tree data structure
- **RBType** (*str*) – rigid body type, 'Vector' or 'Residue'

Returns
coordinates for rigid body as XYZ,Cart where XYZ is the location in crystal coordinates and Cart is in cartesian wrt rigid body coordinates

GSASII.GSASIImath.**ValEsd**(*value*, *esd=0*, *nTZ=False*)

Format a floating point number with a given level of precision or with in crystallographic format with a "esd", as value(esd). If esd is negative the number is formatted with the level of significant figures appropriate if abs(esd) were the esd, but the esd is not included. if the esd is zero, approximately 6 significant figures are printed. nTZ=True causes "extra" zeros to be removed after the decimal place. for example:

- "1.235(3)" for value=1.2346 & esd=0.003
- "1.235(3)e4" for value=12346. & esd=30
- "1.235(3)e6" for value=0.12346e7 & esd=3000
- "1.235" for value=1.2346 & esd=-0.003
- "1.240" for value=1.2395 & esd=-0.003
- "1.24" for value=1.2395 & esd=-0.003 with nTZ=True
- "1.23460" for value=1.2346 & esd=0.0

Parameters

- **value** (*float*) – number to be formatted
- **esd** (*float*) – uncertainty or if esd < 0, specifies level of precision to be shown e.g. esd=-0.01 gives 2 places beyond decimal
- **nTZ** (*bool*) – True to remove trailing zeros (default is False)

Returns
value(esd) or value as a string

GSASII.GSASIImath.**Vol2Den**(*Elements*, *volume*)

converts volume to density

Parameters

- **Elements** (*dict*) – elements in molecular formula; each must contain Num: number of atoms in formula Mass: at. wt.
- **volume** (*float*) – molecular volume in A^3

Returns
float density: material density in gm/cm^3

GSASII.GSASIImath.**XScattDen**(*Elements*, *vol*, *wave=0.0*)

Estimate X-ray scattering density from molecular formula & volume; ignores valence, but includes anomalous effects

Parameters

- **Elements** (*dict*) – elements in molecular formula; each element must contain Num: number of atoms in formula Z: atomic number
- **vol** (*float*) – molecular volume in A^3
- **wave** (*float*) – optional wavelength in A

Returns
float rho: scattering density in 10^10cm^-2; if wave > 0 the includes f' contribution

Returns
float mu: if wave>0 absorption coeff in cm^-1 ; otherwise 0

Returns
float fpp: if wave>0 f" in 10^10cm^-2; otherwise 0

GSASII.GSASIImath.**adjHKLmax**(*SGData*, *Hmax*)

default doc string

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**anneal**(*func*, *x0*, *args=()*, *schedule='fast'*, *T0=None*, *Tf=1e-12*, *maxeval=None*, *maxaccept=None*, *maxiter=400*, *feps=1e-06*, *quench=1.0*, *c=1.0*, *lower=-100*, *upper=100*, *dwell=50*, *slope=0.9*, *ranStart=False*, *ranRange=0.1*, *autoRan=False*, *dlg=None*)

Minimize a function using simulated annealing.

Schedule is a schedule class implementing the annealing schedule. Available ones are 'fast', 'cauchy', 'boltzmann'

Parameters

- **func** (*callable*) – f(x, *args) Function to be optimized.
- **x0** (*ndarray*) – Initial guess.
- **args** (*tuple*) – Extra parameters to *func*.
- **schedule** (*base_schedule*) – Annealing schedule to use (a class).
- **T0** (*float*) – Initial Temperature (estimated as 1.2 times the largest cost-function deviation over random points in the range).
- **Tf** (*float*) – Final goal temperature.
- **maxeval** (*int*) – Maximum function evaluations.
- **maxaccept** (*int*) – Maximum changes to accept.
- **maxiter** (*int*) – Maximum cooling iterations.
- **feps** (*float*) – Stopping relative error tolerance for the function value in last four coolings.
- **quench,c** (*float*) – Parameters to alter fast_sa schedule.
- **lower,upper** (*float/ndarray*) – Lower and upper bounds on *x*.
- **dwell** (*int*) – The number of times to search the space at each temperature.
- **slope** (*float*) – Parameter for log schedule
- **ranStart=False** (*bool*) – True for set 10% of ranges about x

Returns

(xmin, Jmin, T, feval, iters, accept, retval) where

- xmin (ndarray): Point giving smallest value found.
- Jmin (float): Minimum value of function found.
- T (float): Final temperature.
- feval (int): Number of function evaluations.
- iters (int): Number of cooling iterations.
- accept (int): Number of tests accepted.
- retval (int): Flag indicating stopping condition:
  - 0: Points no longer changing
  - 1: Cooled to final temperature
  - 2: Maximum function evaluations
  - 3: Maximum cooling iterations reached
  - 4: Maximum accepted query locations reached
  - 5: Final point not the minimum amongst encountered points

Notes: Simulated annealing is a random algorithm which uses no derivative information from the function being optimized. In practice it has been more useful in discrete optimization than continuous optimization, as there are usually better algorithms for continuous optimization problems.

Some experimentation by trying the difference temperature schedules and altering their parameters is likely required to obtain good performance.

The randomness in the algorithm comes from random sampling in numpy. To obtain the same results you can call numpy.random.seed with the same seed immediately before calling scipy.optimize.anneal.

We give a brief description of how the three temperature schedules generate new points and vary their temperature. Temperatures are only updated with iterations in the outer loop. The inner loop is over range(dwell), and new points are generated for every iteration in the inner loop. (Though whether the proposed new points are accepted is probabilistic.)

For readability, let d denote the dimension of the inputs to func. Also, let x_old denote the previous state, and k denote the iteration number of the outer loop. All other variables not defined below are input variables to scipy.optimize.anneal itself.

In the 'fast' schedule the updates are

```
u ~ Uniform(0, 1, size=d)
y = sgn(u - 0.5) * T * ((1+ 1/T)**abs(2u-1) -1.0)
xc = y * (upper - lower)
x_new = x_old + xc

T_new = T0 * exp(-c * k**quench)
```

GSASII.GSASIImath.**calcRamaEnergy**(*phi*, *psi*, *Coeff=[]*)

Computes pseudo potential energy from a pair of torsion angles and a numerical description of the potential energy surface. Used to create penalty function in LS refinement: $Eval(\phi,\psi) = C[0] * exp(-V/1000)$

where $V = -C[3] * (\phi - C[1])^2 - C[4] * (\psi - C[2])^2 - 2 * (\phi - C[1]) * (\psi - C[2])$

Parameters

- **phi** (*float*) – first torsion angle (ϕ)
- **psi** (*float*) – second torsion angle (ψ)
- **Coeff** (*list*) – pseudo potential coefficients

Returns

list (sum,Eval): pseudo-potential difference from minimum & value; sum is used for penalty function.

GSASII.GSASIImath.**calcTorsionEnergy**(*TOR*, *Coeff=[]*)

default doc string

Parameters

name (*type*) – description

Returns

type name: description

GSASII.GSASIImath.**dropTerms**(*bad*, *hessian*, *indices*, **vectors*)

Remove the 'bad' terms from the Hessian and vector

Parameters

- **bad** (*tuple*) – a list of variable (row/column) numbers that should be removed from the hessian and vector. Example: (0,3) removes the 1st and 4th column/row
- **hessian** (*np.array*) – a square matrix of length n x n
- **indices** (*np.array*) – the indices of the least-squares vector of length n referenced to the initial variable list; as this routine is called multiple times, more terms may be removed from this list
- **additional-args** – various least-squares model values, length n

Returns

hessian, indices, vector0, vector1,… where the lengths are now n' x n' and n', with n' = n - len(bad)

GSASII.GSASIImath.**findOffset**(*SGData*, *A*, *Fhkl*)

default doc string

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**findSSOffset**(*SGData*, *SSGData*, *A*, *Fhklm*)

default doc string

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**fmtPhaseContents**(*compdict*)

Format results from *phaseContents()*

GSASII.GSASIImath.**getAngSig**(*VA*, *VB*, *Amat*, *SGData*, *covData={}*)

Compute an interatomic angle and its uncertainty from two vectors each between an orgin atom and either of a pair of nearby atoms.

Parameters

- **VA** (*np.array*) – an interatomic vector as a structure
- **VB** (*np.array*) – an interatomic vector also as a structure
- **Amat** (*np.array*) – unit cell parameters as an A vector
- **SGData** (*dict*) – symmetry information
- **covData** (*dict*) – covariance information including the covariance matrix and the list of varied parameters. If not supplied, the s.u. values are returned as zeros.

Returns
angle, sigma(angle)

GSASII.GSASIImath.**getAngleDerv**(*Oxyz*, *Axyz*, *Bxyz*, *Amat*, *Tunit*, *symNo*, *SGData*)

computes the numerical derivative of the angle between two vectors (generated from pairs of atoms) used here in *getAngSig()*.

GSASII.GSASIImath.**getAtomPtrs**(*data*, *draw=False*)

get atom data pointers cx,ct,cs,cia in Atoms or Draw Atoms lists NB:may not match column numbers in displayed table

param: dict: data phase data structure draw: boolean True if Draw Atoms list pointers are required return: cx,ct,cs,cia pointers to atom xyz, type, site sym, uiso/aniso flag

GSASII.GSASIImath.**getAtomXYZ**(*atoms*, *cx*)

Create an array of fractional coordinates from the atoms list

Parameters

- **atoms** (*list*) – atoms object as found in tree
- **cx** (*int*) – offset to where coordinates are found

Returns
np.array with shape (n,3)

GSASII.GSASIImath.**getCWgam**(*ins*, *pos*)

get CW peak profile gamma

Parameters

- **ins** (*dict*) – instrument parameters with at least 'X', 'Y' & 'Z' as values only
- **pos** (*float*) – 2-theta of peak

Returns
float getCWgam: peak gamma

GSASII.GSASIImath.**getCWgamDeriv**(*pos*)

get derivatives of CW peak profile gamma wrt X, Y & Z

Parameters
pos (*float*) – 2-theta of peak

Returns
list getCWgamDeriv: d(gam)/dX & d(gam)/dY

GSASII.GSASIImath.**getCWsig**(*ins*, *pos*)

get CW peak profile sigma^2

Parameters

- **ins** (*dict*) – instrument parameters with at least 'U', 'V', & 'W' as values only
- **pos** (*float*) – 2-theta of peak

Returns
float getCWsig: peak sigma^2

GSASII.GSASIImath.**getCWsigDeriv**(*pos*)

get derivatives of CW peak profile sigma^2 wrt U,V, & W

Parameters
pos (*float*) – 2-theta of peak

Returns
list getCWsigDeriv: d(sig^2)/dU, d(sig)/dV & d(sig)/dW

GSASII.GSASIImath.**getDensity**(*generalData*, *hist=None*, *data=None*)

Calculate crystal structure density. Uses cell values only, unless hist & data are supplied. In that case it uses the Dij terms as well.

Parameters

- **generalData** (*dict*) – The General dictionary in Phase
- **hist** (*str*) – optional name of a histogram. When not None, the volume adjusted bt the Dij (hydrostatic strain terms) is used to compute the density.

Parm dict data
reference to entire phase data array. Required if hist is specified. Ignored otherwise.

Returns
crystal density in gm/cm^3 (float) and Matthews Coeff. (Vm or Volume/mass, float)

GSASII.GSASIImath.**getDistDerv**(*Oxyz*, *Txyz*, *Amat*, *Tunit*, *Top*, *SGData*)

computes the numerical derivative of the distance between two atoms. Used in *CalcDistDeriv()* (seq. table) and in GSASIIstrMain.RetDistAngle() to compute the s.u. on the distance

Parameters

- **Oxyz** (*list*) – list of x, y, & z values for the Origin atom
- **Txyz** (*list*) – list of x, y, & z values for the Target atom
- **Amat** (*np.array*) – The reciprocal cell tensor
- **Tunit** – translation applied to the target atom
- **Top** (*int*) – symmetry operation applied to the target atom
- **SGData** (*dict*) – space group object

Returns
the derivative w/r to the six coordinates, Oxyz & Txyz

GSASII.GSASIImath.**getEDgam**(*ins*, *pos*)

get ED peak profile gam

Parameters

- **ins** (*dict*) – instrument parameters with at least X, Y & Z as values only
- **pos** (*float*) – energy of peak as keV

Returns
float getEDsig: peak gam im keV

GSASII.GSASIImath.**getEDgamDeriv**(*ins*, *pos*)

get derivatives of ED peak profile gam wrt X, Y & Z

Parameters
pos (*float*) – energy of peak in keV

Returns
list getEDsigDeriv: d(gam)/dX, d(gam)/dY & d(gam)/dZ,

GSASII.GSASIImath.**getEDsig**(*ins*, *pos*)

get ED peak profile sig

Parameters

- **ins** (*dict*) – instrument parameters with at least 'A', 'B' & 'C' as values only
- **pos** (*float*) – energy of peak as keV

Returns
float getEDsig: peak sigma^2 im keV**2

GSASII.GSASIImath.**getEDsigDeriv**(*ins*, *pos*)

get derivatives of ED peak profile sig wrt A, B & C

Parameters
pos (*float*) – energy of peak in keV

Returns
list getEDsigDeriv: d(sig)/dA, d(sig)/dB & d(sig)/dC,

GSASII.GSASIImath.**getMass**(*generalData*)

Computes mass of unit cell contents

Parameters
generalData (*dict*) – The General dictionary in Phase

Returns
float mass: Crystal unit cell mass in AMU.

GSASII.GSASIImath.**getMeanWave**(*Parms*)

returns mean wavelength from Instrument parameters dictionary

Parameters
Parms (*dict*) – Instrument parameters; must contain: Lam: single wavelength or Lam1,Lam2: Ka1,Ka2 radiation wavelength I(L2)/I(L1): Ka2/Ka1 ratio

Returns
float wave: mean wavelength

GSASII.GSASIImath.**getPinkAlpha**(*ins*, *tth*)

get pink neutron peak alpha profile

Parameters

- **ins** (*dict*) – instrument parameters with at least 'alpha' as values only
- **tth** (*float*) – 2-theta of peak

Returns
float getPinkNalpha: peak alpha

GSASII.GSASIImath.**getPinkAlphaDeriv**(*tth*)

get alpha derivatives of pink neutron peak profile

Parameters
tth (*float*) – 2-theta of peak

Returns
float getPinkNalphaDeriv: d(alp)/d(alpha-0), d(alp)/d(alpha-1)

GSASII.GSASIImath.**getPinkBeta**(*ins*, *tth*)

get pink neutron peak profile beta

Parameters

- **ins** (*dict*) – instrument parameters with at least 'beta-0' & 'beta-1' as values only
- **tth** (*float*) – 2-theta of peak

Returns
float getPinkbeta: peak beta

GSASII.GSASIImath.**getPinkBetaDeriv**(*tth*)

get beta derivatives of pink neutron peak profile

Parameters
tth (*float*) – 2-theta of peak

Returns
list getPinkNbetaDeriv: d(beta)/d(beta-0) & d(beta)/d(beta-1)

GSASII.GSASIImath.**getRBAngSig**(*VA*, *VB*, *Amat*, *SGData*, *covData*, *multiParmDict*, *changedParmDict*)

Compute an interatomic angle and its uncertainty from two vectors each between an orgin atom and either of a pair of nearby atoms. Uncertainties are computed taling into account rigid bodies and constraints.

Parameters

- **VA** (*np.array*) – an interatomic vector as a structure
- **VB** (*np.array*) – an interatomic vector also as a structure
- **Amat** (*np.array*) – unit cell parameters as an A vector
- **SGData** (*dict*) – symmetry information

- **covData** (*dict*) – covariance information including the covariance matrix and the list of varied parameters. If not supplied, the s.u. values are returned as zeros.

Returns
angle, sigma(angle)

GSASII.GSASIImath.**getRBDistDerv**(*OdxyzNames*, *TdxyzNames*, *Amat*, *Tunit*, *Top*, *SGData*, *multiParmDict*, *changedParmDict*, *varyList*, *sigList*)

computes the numerical derivative of the distance between two atoms. Used where one or both is in a rigid body. Used in `GSASIIstrMain.RetDistAngle()` to compute the s.u. on the distance

Parameters

- **OxyzNames** (*list*) – parameter names for x, y, & z for the Origin atom
- **TxyzNames** (*list*) – parameter names for x, y, & z for the Target atom
- **Amat** (*np.array*) – The reciprocal cell tensor
- **Tunit** – translation applied to the target atom
- **Top** (*int*) – symmetry operation applied to the target atom
- **SGData** (*dict*) – space group object
- **multiParmDict** (*dict*) – multiParmDict[var] is the parameter dict where var has been offset by sigma (see sigList), as well as any parameters dependent on var
- **changedParmDict** (*dict*) – changedParmDict[var] is a list the parameters changed in multiParmDict[var]
- **varyList** (*list*) – list of varied parameters in the covariance matrix
- **sigList** (*list*) – list of s.u. values for each of entry in varyList

Returns
the derivative w/r to the six coordinates, Oxyz & Txyz

GSASII.GSASIImath.**getRBTransMat**(*X*, *Y*)

Get transformation for Cartesian axes given 2 vectors X will be parallel to new X-axis; X cross Y will be new Z-axis & (X cross Y) cross Y will be new Y-axis Useful for rigid body axes definintion

Parameters

- **X** (*array*) – normalized vector
- **Y** (*array*) – normalized vector

Returns
array M: transformation matrix

use as XYZ' = np.inner(M,XYZ) where XYZ are Cartesian

GSASII.GSASIImath.**getRamaDeriv**(*XYZ*, *Amat*, *Coeff*)

Computes numerical derivatives of torsion angle pair pseudo potential with respect of crystallographic atom coordinates of the 5 atom sequence

Parameters

- **XYZ** (*nparray*) – crystallographic coordinates of 5 atoms
- **Amat** (*nparray*) – crystal to cartesian transformation matrix
- **Coeff** (*list*) – pseudo potential coefficients

Returns
list (deriv) derivatives of pseudopotential with respect to 5 atom crystallographic xyz coordinates.

GSASII.GSASIImath.**getRestAngle**(*XYZ*, *Amat*)

Compute interatomic angle(s) for use in restraints

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**getRestChiral**(*XYZ*, *Amat*)

compute a chiral restraint

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**getRestDeriv**(*Func*, *XYZ*, *Amat*, *ops*, *SGData*)

Compute numerical derivatives of restraints for use in minimization

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**getRestDist**(*XYZ*, *Amat*)

Compute interatomic distance(s) for use in restraints

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**getRestPlane**(*XYZ*, *Amat*)

Compute deviations from a best plane through atoms for use in restraints

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**getRestPolefig**(*ODFln*, *SamSym*, *Grid*)

default doc string

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**getRestPolefigDerv**(*HKL*, *Grid*, *SHCoeff*)

default doc string

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**getRestRama**(*XYZ*, *Amat*)

Computes a pair of torsion angles in a 5 atom string

Parameters

- **XYZ** (*nparray*) – crystallographic coordinates of 5 atoms
- **Amat** (*nparray*) – crystal to cartesian transformation matrix

Returns
list (phi,psi) two torsion angles in degrees

GSASII.GSASIImath.**getRestTorsion**(*XYZ*, *Amat*)

compute a torsion restraint

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**getRho**(*xyz*, *mapData*)

get scattering density at a point by 8-point interpolation param xyz: coordinate to be probed param: mapData: dict of map data

Returns
density at xyz

GSASII.GSASIImath.**getRhos**(*XYZ*, *rho*)

get scattering density at an array of point by 8-point interpolation this is faster than gerRho which is only used for single points. However, getRhos is replaced by scipy.ndimage.interpolation.map_coordinates which does a better job & is just as fast. Thus, getRhos is unused in GSAS-II at this time. param xyz: array coordinates to be probed Nx3 param: rho: array copy of map (NB: don't use original!)

Returns
density at xyz

GSASII.GSASIImath.**getSyXYZ**(*XYZ*, *ops*, *SGData*)

default doc

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**getTOFalpha**(*ins*, *dsp*)

get TOF peak profile alpha

Parameters

- **ins** (*dict*) – instrument parameters with at least 'alpha' as values only
- **dsp** (*float*) – d-spacing of peak

Returns
float getTOFalpha: peak alpha

GSASII.GSASIImath.**getTOFalphaDeriv**(*dsp*)

get alpha derivatives of TOF peak profile

Parameters
dsp (*float*) – d-spacing of peak

Returns
float getTOFalphaDeriv: d(alp)/d(alpha)

GSASII.GSASIImath.**getTOFbeta**(*ins*, *dsp*)

get TOF peak profile beta

Parameters

- **ins** (*dict*) – instrument parameters with at least 'beat-0', 'beta-1' & 'beta-q' as values only
- **dsp** (*float*) – d-spacing of peak

Returns
float getTOFbeta: peak beat

GSASII.GSASIImath.**getTOFbetaDeriv**(*dsp*)

get derivatives of TOF peak profile beta wrt beta-0, beta-1, & beat-q

Parameters
dsp (*float*) – d-spacing of peak

Returns
list getTOFbetaDeriv: d(beta)/d(beat-0), d(beta)/d(beta-1) & d(beta)/d(beta-q)

GSASII.GSASIImath.**getTOFgamma**(*ins*, *dsp*)

get TOF peak profile gamma

Parameters

- **ins** (*dict*) – instrument parameters with at least 'X', 'Y' & 'Z' as values only
- **dsp** (*float*) – d-spacing of peak

Returns
float getTOFgamma: peak gamma

GSASII.GSASIImath.**getTOFgammaDeriv**(*dsp*)

get derivatives of TOF peak profile gamma wrt X, Y & Z

Parameters
dsp (*float*) – d-spacing of peak

Returns
list getTOFgammaDeriv: d(gam)/dX & d(gam)/dY

GSASII.GSASIImath.**getTOFsig**(*ins*, *dsp*)

get TOF peak profile sigma^2

Parameters

- **ins** (*dict*) – instrument parameters with at least 'sig-0', 'sig-1' & 'sig-q' as values only
- **dsp** (*float*) – d-spacing of peak

Returns
float getTOFsig: peak sigma^2

GSASII.GSASIImath.**getTOFsigDeriv**(*dsp*)

get derivatives of TOF peak profile sigma^2 wrt sig-0, sig-1, & sig-q

Parameters
dsp (*float*) – d-spacing of peak

Returns
list getTOFsigDeriv: d(sig0/d(sig-0), d(sig)/d(sig-1) & d(sig)/d(sig-q)

GSASII.GSASIImath.**getTorsionDeriv**(*XYZ*, *Amat*, *Coeff*)

default doc string

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**getVCov**(*varyNames*, *varyList*, *covMatrix*)

obtain variance-covariance terms for a set of variables. NB: the varyList and covMatrix were saved by the last least squares refinement so they must match.

Parameters

- **varyNames** (*list*) – variable names to find v-cov matric for
- **varyList** (*list*) – full list of all variables in v-cov matrix
- **covMatrix** (*nparray*) – full variance-covariance matrix from the last least squares refinement

Returns
nparray vcov: variance-covariance matrix for the variables given in varyNames

GSASII.GSASIImath.**getWave**(*Parms*)

returns wavelength from Instrument parameters dictionary

Parameters
Parms (*dict*) – Instrument parameters; must contain: Lam: single wavelength or Lam1: Ka1 radiation wavelength

Returns
float wave: wavelength

GSASII.GSASIImath.**invQ**(*Q*)

get inverse of quaternion q=r+ai+bj+ck; q* = r-ai-bj-ck

GSASII.GSASIImath.**make2Quat**(*A*, *B*)

Make quaternion from rotation of A vector to B vector

Parameters
A,B (*np.array*) – Cartesian 3-vectors

Returns
quaternion & rotation angle in radians q=r+ai+bj+ck

GSASII.GSASIImath.**makeQuat**(*A*, *B*, *C*)

Make quaternion from rotation of A vector to B vector about C axis

Parameters
A,B,C (*np.array*) – Cartesian 3-vectors

Returns
quaternion & rotation angle in radians q=r+ai+bj+ck

GSASII.GSASIImath.**makeWaves**(*waveTypes*, *FSSdata*, *XSSdata*, *USSdata*, *MSSdata*, *Mast*)

waveTypes: array nAtoms: 'Fourier','ZigZag' or 'Block' FSSdata: array 2 x atoms x waves (sin,cos terms) XSSdata: array 2x3 x atoms X waves (sin,cos terms) USSdata: array 2x6 x atoms X waves (sin,cos terms) MSSdata: array 2x3 x atoms X waves (sin,cos terms)

Mast: array orthogonalization matrix for Uij

GSASII.GSASIImath.**makeWavesDerv**(*ngl*, *waveTypes*, *FSSdata*, *XSSdata*, *USSdata*, *Mast*)

Only for Fourier waves for fraction, position & adp (probably not used for magnetism) FSSdata: array 2 x atoms x waves (sin,cos terms) XSSdata: array 2x3 x atoms X waves (sin,cos terms) USSdata: array 2x6 x atoms X waves (sin,cos terms) Mast: array orthogonalization matrix for Uij

GSASII.GSASIImath.**mcsaSearch**(*data*, *RBdata*, *reflType*, *reflData*, *covData*, *pgbar*, *start=True*)

default doc string

Parameters
name (*type*) – description

Returns
type name: description

GSASII.GSASIImath.**normQ**(*QA*)

get length of quaternion & normalize it q=r+ai+bj+ck

GSASII.GSASIImath.**patchIsoDisp**(*ISO*)

patch: look for older ISODISTORT imports (<Nov 2021)

GSASII.GSASIImath.**phaseContents**(*phase*)

Compute the unit cell and asymmetric unit contents for a phase.

This has been tested only on type='nuclear' phases and might need adaptation for phases of other types, if the phase type does not have an occupancy defined.

Parameters
phase (*dict*) – the dict for a phase, as found in the data tree

Returns
acomp,ccomp where acomp is the asymmetric unit contents and ccomp is the contents of the unit cell

GSASII.GSASIImath.**pinv**(*a*, *rcond=1e-15*)

Compute the (Moore-Penrose) pseudo-inverse of a matrix. Modified from numpy.linalg.pinv; assumes a is Hessian & returns no. zeros found Calculate the generalized inverse of a matrix using its singular-value decomposition (SVD) and including all *large* singular values.

Parameters

- **a** (*array*) – (M, M) array_like - here assumed to be LS Hessian Matrix to be pseudo-inverted.
- **rcond** (*float*) – Cutoff for small singular values. Singular values smaller (in modulus) than *rcond* * largest_singular_value (again, in modulus) are set to zero.

Returns
B : (M, M) ndarray The pseudo-inverse of *a*

Raises: LinAlgError
If the SVD computation does not converge.

Notes:

The pseudo-inverse of a matrix A, denoted A^+, is defined as: "the matrix that 'solves' [the least-squares problem] $Ax = b$," i.e., if $\bar{x}$ is said solution, then A^+ is that matrix such that $\bar{x} = A^+b$.

It can be shown that if $Q_1 \Sigma Q_2^T = A$ is the singular value decomposition of A, then $A^+ = Q_2 \Sigma^+ Q_1^T$, where $Q_{1,2}$ are orthogonal matrices, Σ is a diagonal matrix consisting of A's so-called singular values, (followed, typically, by zeros), and then Σ^+ is simply the diagonal matrix consisting of the reciprocals of A's singular values (again, followed by zeros). [1]

References: .. [1] G. Strang, *Linear Algebra and Its Applications*, 2nd Ed., Orlando, FL, Academic Press, Inc., 1980, pp. 139-142.

GSASII.GSASIImath.**printRho**(*SGLaue*, *rho*, *rhoMax*)

default doc string

Parameters

name (*type*) – description

Returns

type name: description

GSASII.GSASIImath.**prodQQ**(*QA*, *QB*)

Grassman quaternion product QA,QB quaternions; q=r+ai+bj+ck

GSASII.GSASIImath.**prodQVQ**(*Q*, *V*)

compute the quaternion vector rotation qvq-1 = v' q=r+ai+bj+ck

GSASII.GSASIImath.**randomAVdeg**(*r0*, *r1*, *r2*, *r3*)

create random angle (deg),vector from 4 random number in range (-1,1)

GSASII.GSASIImath.**randomQ**(*r0*, *r1*, *r2*, *r3*)

create random quaternion from 4 random numbers in range (-1,1)

GSASII.GSASIImath.**searchBondRestr**(*origAtoms*, *targAtoms*, *bond*, *Factor*, *GType*, *SGData*, *Amat*, *defESD=0.01*, *dlg=None*)

Search for bond distance restraints.

GSASII.GSASIImath.**setHcorr**(*info*, *Amat*, *xtol*, *problem=False*)

Find & report high correlation terms in covariance matrix

GSASII.GSASIImath.**setPeakparms**(*Parms*, *Parms2*, *pos*, *mag*, *ifQ=False*, *useFit=False*)

set starting peak parameters for single peak fits from plot selection or auto selection

Parameters

- **Parms** (*dict*) – instrument parameters dictionary
- **Parms2** (*dict*) – table lookup for TOF profile coefficients
- **pos** (*float*) – peak position in 2-theta, TOF or Q (ifQ=True)
- **mag** (*float*) – peak top magnitude from pick
- **ifQ** (*bool*) – True if pos in Q
- **useFit** (*bool*) – True if use fitted CW Parms values (not defaults)

Returns

list XY: peak list entry: for CW: [pos,0,mag,1,sig,0,gam,0] for TOF: [pos,0,mag,1,alp,0,bet,0,sig,0,gam,0] for Pink: [pos,0,mag,1,alp,0,bet,0,sig,0,gam,0] NB: mag refinement set by default, all others off

GSASII.GSASIImath.**setSVDwarn**(*info*, *Amat*, *Nzeros*, *indices*)

Find & report terms causing SVD zeros

GSASII.GSASIImath.**setupRBDistDerv**(*parmDict*, *varyList*, *sigList*, *rigidbodyDict*, *Phases*)

Compute a copy of the parameter dict (parmDict) with each varied parameter incremented by 1 s.u. value and with those values extended to other parameters due to rigid bodies or constraints. This gets called once to prepare for s.u. computations in func:*GSASIIstrMain.RetDistAngle* and the values are reused in every distance and angle computation.

Returns

multiParmDict,changedParmDict where

- multiParmDict[k] (k in varyList, plus None) is the copy of parmDict where parameter k has been changed;
- changedParmDict[k] (k in varyList) is a list of all the parameters that have been changed in multiParmDict[k] after applying RB & constraints

GSASII.GSASIImath.**sortArray**(*data*, *pos*, *reverse=False*)

data is a list of items sort by pos in list; reverse if True

GSASII.GSASIImath.**wavekE**(*wavekE*)

Convert wavelength to energy & vice versa

Parameters

waveKe (*float*) – wavelength in A or energy in kE

Returns float waveKe

the other one

CHAPTER

FOURTEEN

GSASIIINDEX: CELL INDEXING MODULE

14.1 Summary/Contents

Unit cell indexing routines (based on work of A. Coehlo) and cell refinement from peak positions

Section Contents

14.2 GSASIIindex routines

Classes and routines defined in `GSASIIindex` follow.

`GSASII.GSASIIindex.`**`A2values`**(*ibrav*, *A*)

needs a doc string

`GSASII.GSASIIindex.`**`DoIndexPeaks`**(*peaks*, *controls*, *bravais*, *dlg*, *ifX20=True*, *timeout=None*, *M20_min=2.0*, *X20_max=None*, *return_Nc=False*, *cctbx_args=None*)

needs a doc string

`GSASII.GSASIIindex.`**`FitHKL`**(*ibrav*, *peaks*, *A*, *Pwr*)

needs a doc string

`GSASII.GSASIIindex.`**`FitHKLE`**(*tth*, *ibrav*, *peaks*, *A*)

needs a doc string

`GSASII.GSASIIindex.`**`FitHKLT`**(*difC*, *ibrav*, *peaks*, *A*, *Z*, *Zref*)

needs a doc string

`GSASII.GSASIIindex.`**`FitHKLTSS`**(*difC*, *ibrav*, *peaks*, *A*, *V*, *Vref*, *Z*, *Zref*)

needs a doc string

`GSASII.GSASIIindex.`**`FitHKLZ`**(*wave*, *ibrav*, *peaks*, *A*, *Z*, *Zref*)

needs a doc string

`GSASII.GSASIIindex.`**`FitHKLZSS`**(*wave*, *ibrav*, *peaks*, *A*, *V*, *Vref*, *Z*, *Zref*)

needs a doc string

GSASII.GSASIIindex.**IndexPeaks**(*peaks*, *HKL*)

needs a doc string

GSASII.GSASIIindex.**IndexSSPeaks**(*peaks*, *HKL*)

needs a doc string

GSASII.GSASIIindex.**TestData**()

needs a doc string

GSASII.GSASIIindex.**Values2A**(*ibrav*, *values*)

needs a doc string

GSASII.GSASIIindex.**calc_M20**(*peaks*, *HKL*, *ifX20=True*)

needs a doc string

GSASII.GSASIIindex.**calc_M20SS**(*peaks*, *HKL*)

needs a doc string

GSASII.GSASIIindex.**findBestCell**(*dlg*, *ncMax*, *A*, *Ntries*, *ibrav*, *peaks*, *V1*, *ifX20=True*, *cctbx_args=None*)

needs a doc string

GSASII.GSASIIindex.**getDmax**(*peaks*)

needs a doc string

GSASII.GSASIIindex.**getDmin**(*peaks*)

needs a doc string

GSASII.GSASIIindex.**halfCell**(*ibrav*, *A*, *peaks*)

needs a doc string

GSASII.GSASIIindex.**monoCellReduce**(*ibrav*, *A*)

needs a doc string

GSASII.GSASIIindex.**oddPeak**(*indx*, *peaks*)

needs a doc string

GSASII.GSASIIindex.**ran2axis**(*k*, *N*)

needs a doc string

GSASII.GSASIIindex.**ranAbyR**(*Bravais*, *A*, *k*, *N*, *ranFunc*)

needs a doc string

GSASII.GSASIIindex.**ranAbyV**(*Bravais*, *dmin*, *dmax*, *V*)

needs a doc string

GSASII.GSASIIindex.**ranaxis**(*dmin*, *dmax*)

needs a doc string

GSASII.GSASIIindex.**rancell**(*Bravais*, *dmin*, *dmax*)

needs a doc string

GSASII.GSASIIindex.**refinePeaks**(*peaks*, *ibrav*, *A*, *ifX20=True*, *cctbx_args=None*)

needs a doc string

GSASII.GSASIIindex.**refinePeaksE**(*peaks*, *tth*, *ibrav*, *A*)

needs a doc string

GSASII.GSASIIindex.**refinePeaksT**(*peaks*, *difC*, *ibrav*, *A*, *Zero*, *ZeroRef*)

needs a doc string

GSASII.GSASIIindex.**refinePeaksTSS**(*peaks*, *difC*, *Inst*, *SGData*, *SSGData*, *maxH*, *ibrav*, *A*, *vec*, *vecRef*, *Zero*, *ZeroRef*)

needs a doc string

GSASII.GSASIIindex.**refinePeaksZ**(*peaks*, *wave*, *ibrav*, *A*, *Zero*, *ZeroRef*)

needs a doc string

GSASII.GSASIIindex.**refinePeaksZSS**(*peaks*, *wave*, *Inst*, *SGData*, *SSGData*, *maxH*, *ibrav*, *A*, *vec*, *vecRef*, *Zero*, *ZeroRef*)

needs a doc string

GSASII.GSASIIindex.**rotOrthoA**(*A*)

needs a doc string

GSASII.GSASIIindex.**scaleAbyV**(*A*, *V*)

needs a doc string

GSASII.GSASIIindex.**sortM20**(*cells*)

needs a doc string

GSASII.GSASIIindex.**swapMonoA**(*A*)

needs a doc string

CHAPTER

FIFTEEN

VISUALIZATION ROUTINES

15.1 *Summary/Contents*

Routines for visualization, using matplotlib and OpenGL graphics. Note that the plot toolbar is customized with `GSASII-Itoolbar`

Section Contents

15.2 *List of Graphics routines*

The following plotting routines are defined:

plotting routine	action
`PlotPatterns()`	Powder pattern plotting
`PublishRietveldPlot()`	Create publication-quality Rietveld plots from `PlotPatterns()` plot
`PlotImage()`	Plots of 2D detector images
`PlotPeakWidths()`	Plot instrument broadening terms as function of 2-theta/TOF
`PlotCovariance()`	Show covariance terms in 2D
`PlotStructure()`	Crystal structure plotting with balls, sticks, lines, ellipsoids, polyhedra and magnetic moments
`PlotBeadModel()`	Plots representation of protein shape from small angle scattering
`Plot1DSngl()`	1D stick plots of structure factors
`PlotSngl()`	Structure factor plotting
`Plot3DSngl()`	3D Structure factor plotting
`PlotDeltSig()`	Normal probability plot (powder or single crystal)
`PlotISFG()`	PDF analysis: displays I(Q), S(Q), F(Q) and G(r)
`PlotCalib()`	CW or TOF peak calibration
`PlotXY()`	Simple plot of xy data

continues on next page

Table 1 – continued from previous page

plotting routine	action
`PlotXYZ()`	Simple contour plot of xyz data
`PlotXYZvect()`	Quiver Plot for 3D cartesian vectors
`Plot3dXYZ()`	Surface Plot for 3D vectors
`PlotAAProb()`	Protein “quality” plot
`PlotStrain()`	Plot of strain data, used for diagnostic purposes
`PlotSASDSizeDist()`	Small angle scattering size distribution plot
`PlotPowderLines()`	Plot powder pattern as a stick plot (vertical lines)
`PlotSizeStrainPO()`	Plot 3D mustrain/size/preferred orientation figure
`PlotTexture()`	Pole figure, inverse pole figure plotting
`ModulationPlot()`	Plots modulation information
`PlotTorsion()`	Plots MC torsion angles
`PlotRama()`	Ramachandran of energetically allowed regions for dihedral angles in protein
`PlotSelectedSequence()`	Plot one or more sets of values selected from the sequential refinement table
`PlotIntegration()`	Rectified plot of 2D image after image integration with 2-theta and azimuth as coordinates
`PlotTRImage()`	test plot routine
`PlotRigidBody()`	show rigid body structures as balls & sticks
`PlotLayers()`	show layer structures as balls & sticks
`PlotFPAconvolutors()`	plots the convolutors from Fundamental Parameters
`PlotClusterXYZ()`	plots the result of cluster analysis

15.3 *Window management routines*

The above plotting routines place their graphics in the GSAS-II Plot Window, which contains a `GSASIIplot.G2PlotNoteBook` tabbed panel allowing multiple plots to be viewed. Methods `GSASIIplot.G2PlotNoteBook.addMpl()` (2-D matplotlib), `GSASIIplot.G2PlotNoteBook.add3D()` (3-D matplotlib), and `GSASIIplot.G2PlotNoteBook.addOgl()` (OpenGL) are used to create tabbed plot objects to hold plots of the following classes: `GSASIIplot.G2PlotMpl` (2-D matplotlib), `GSASIIplot.G2Plot3D` (3-D matplotlib), and `GSASIIplot.G2PlotOgl` (OpenGL). Note that two `GSASIIplot.G2PlotNoteBook` methods are potentially used to determine how plot updates after a refinement are handled:

class method	description
`GSASIIplot.G2PlotNoteBook.RegisterRedrawRoutine()`	This specifies a function to redraw the plot after the data tree has been reloaded. Be sure this updates data objects with new values from the tree, when needed.
`GSASIIplot.G2PlotNoteBook.SetNoDelete()`	Use this to indicate that a plot does not need to be updated after a refinement and should not be closed.

These two methods define the following attributes (variables) in the plot tab classes:

variable	default	use
replotFunction	None	Defines a routine to be called to update the plot after a refinement (unless None). Use `GSASIIplot.G2PlotNoteBook.RegisterRedrawRoutine()` to define this (and replotArgs & replotKwArgs). Plotting functions that take significant time to complete should probably not use this.)
replotArgs	[]	Defines the positional arguments to be supplied to the replotFunction function or method.
replotKwArgs	{}	Defines the keyword arguments to be supplied to the replotFunction function or method.
plotRequiresRedraw	True	If set to True, after a refinement, the plot will be closed (in `GSASIIdataGUI.GSASII.ResetPlots()`) if it was not updated after the refinement. Set this to False using `GSASIIplot.G2PlotNoteBook.SetNoDelete()` for plots that should not be deleted or do not change based on refinement results.
plotInvalid	False	Used to track if a plot has been updated. Set to False in `GSASIIplot.G2PlotNoteBook.FindPlotTab()` when a plot is drawn. After a refinement is completed, method `GSASIIdataGUI.GSASII.ResetPlots()` sets plotInvalid to False for all plots before any routines are called.

15.4 *GSASIIplot Classes and Routines*

Classes and routines defined in `GSASIIplot` follow.

`GSASII.GSASIIplot.`**`ComputeArc`**(*angI*, *angO*, *wave*, *azm0=0*, *azm1=362*)

Computes arc/ring arrays in with inner and outer radii from angI,angO and beginning and ending azimuths azm0,azm1 (optional). Returns the inner and outer ring/arc arrays.

class `GSASII.GSASIIplot.`**`G2Plot3D`**(*parent*, *id=-1*, *dpi=None*, ***kwargs*)

Creates a 3D Matplotlib plot in the GSAS-II graphics window

`__init__`(*parent*, *id=-1*, *dpi=None*, ***kwargs*)

class `GSASII.GSASIIplot.`**`G2PlotMpl`**(*parent*, *id=-1*, *dpi=None*, *publish=None*, ***kwargs*)

Creates a Matplotlib 2-D plot in the GSAS-II graphics window

`__init__`(*parent*, *id=-1*, *dpi=None*, *publish=None*, ***kwargs*)

class `GSASII.GSASIIplot.`**`G2PlotNoteBook`**(*parent*, *id=-1*, *G2frame=None*)

create a tabbed panel to hold a GSAS-II graphics window

`Delete`(*name*)

delete a tabbed page

`FindPlotTab`(*label*, *Type*, *newImage=True*, *publish=None*)

Open a plot tab for initial plotting, or raise the tab if it already exists Set a flag (Page.plotInvalid) that it has been redrawn Record the name of the this plot in self.lastRaisedPlotTab

Parameters

- **`label`** (*str*) – title of plot
- **`Type`** (*str*) – determines the type of plot that will be opened.

  'mpl' for 2D graphs in matplotlib 'ogl' for openGL '3d' for 3D plotting in matplotlib
- **`newImage`** (*bool*) – forces creation of a new graph for matplotlib plots only (defaults as True)

- **publish** (*function*) – reference to routine used to create a publication version of the current mpl plot (default is None, which prevents use of this).

Returns

new,plotNum,Page,Plot,limits where

- new: will be True if the tab was just created
- plotNum: is the tab number
- Page: is the subclassed wx.Panel (*G2PlotMpl*, etc.) where the plot appears
- Plot: the mpl.Axes object for the graphic (mpl) or the figure for openGL.
- limits: for mpl plots, when a plot already exists, this will be a tuple with plot scaling. None otherwise.

GetTabIndex(*label*)

Look up a tab label and return the index in the notebook (this appears to be independent to the order it is dragged to – at least in Windows) as well as the associated wx.Panel

An exception is raised if the label is not found

InvokeTreeItem(*pid*)

This is called to select an item from the tree using the self.allowZoomReset flag to prevent a reset to the zoom of the plot (where implemented)

OnNotebookKey(*event*)

Called when a keystroke event gets picked up by the notebook window rather the child. This is not expected, but somehow it does sometimes on the Mac and perhaps Linux.

Assume that the page associated with the currently displayed tab has a child, .canvas; give that child the focus and pass it the event.

OnPageChanged(*event*)

respond to someone pressing a tab on the plot window. Called when a plot tab is clicked. on some platforms (Mac for sure) this is also called when a plot is created or selected with .SetSelection() or .SetFocus().

(removed) The self.skipPageChange is used variable is set to suppress repeated replotting.

RaisePageNoRefresh(*Page*)

Raises a plot tab without triggering a refresh via OnPageChanged

RegisterRedrawRoutine(*name*, *routine=None*, *args=()*, *kwargs={}*)

Save information to determine how to redraw a plot

Parameters

- **name** (*str*) – label on tab of plot
- **routine** (*Object*) – a function to be called
- **args** – a list of positional parameters for the function
- **kwargs** – a dict with keyword parameters for the function

Rename(*oldName*, *newName*)

rename a tab

SetHelpButton(*help*)

Adds a Help button to the status bar on plots.

TODO: This has a problem with G2pwpl.PlotPatterns where creation of the HelpButton causes the notebook tabs to be duplicated. A manual resize fixes that, but the SendSizeEvent has not worked.

SetNoDelete(*name*)

Indicate that a plot does not need to be redrawn

SetSelectionNoRefresh(*plotNum*)

Raises a plot tab without triggering a refresh via OnPageChanged

__init__(*parent*, *id=-1*, *G2frame=None*)

_addPage(*name*, *page*)

Add the newly created page to the notebook and associated lists.

Parameters

- **name** – the label placed on the tab, which should be unique
- **page** – the wx.Frame for the matplotlib, openGL, etc. window

add3D(*name=''*)

Add a tabbed page with a 3D plot

addMpl(*name=''*, *publish=None*)

Add a tabbed page with a matplotlib plot

addOgl(*name=''*)

Add a tabbed page with an openGL plot

clear()

clear all pages from plot window

class GSASII.GSASIIplot.**G2PlotOgl**(*parent*, *id=-1*, *dpi=None*, ***kwargs*)

Creates an OpenGL plot in the GSAS-II graphics window

__init__(*parent*, *id=-1*, *dpi=None*, ***kwargs*)

class GSASII.GSASIIplot.**GSASIItoolbar**(*plotCanvas*, *publish=None*, *Arrows=True*)

Override the matplotlib toolbar so we can add more icons

OnArrow(*event*)

reposition limits to scan or zoom by button press

OnHelp(*event*)

Respond to press of help button on plot toolbar

OnKey(*event*)

Provide user with list of keystrokes defined for plot as well as an alternate way to access the same functionality

__init__(*plotCanvas*, *publish=None*, *Arrows=True*)

Adds additional icons to toolbar

_update_view()

Overrides the post-buttonbar update action to invoke a redraw; needed for plot magnification

get_zoompan()

Return "Zoom" if Zoom is active, "Pan" if Pan is active, or None if neither

set_message(*s*)

this removes spurious text messages from the tool bar

GSASII.GSASIIplot.**ModulationPlot**(*G2frame*, *data*, *atom*, *ax*, *off=0*)

Needs a description

GSASII.GSASIIplot.**OnStartMask**(*G2frame*)

Initiate the start of a Frame or Polygon map, etc. Called from a menu command (GSASIIimgGUI) or from OnImPlotKeyPress. Variable G2frame.MaskKey contains a single letter ('f' or 'p', etc.) that determines what type of mask is created.

Parameters

G2frame (*wx.Frame*) – The main GSAS-II tree "window"

GSASII.GSASIIplot.**OnStartNewDzero**(*G2frame*)

Initiate the start of adding a new d-zero to a strain data set

Parameters

- **G2frame** (*wx.Frame*) – The main GSAS-II tree "window"
- **eventkey** (*str*) – a single letter ('a') that triggers the addition of a d-zero.

GSASII.GSASIIplot.**Plot1DSngl**(*G2frame*, *newPlot=False*, *hklRef=None*, *Super=0*, *Title=False*)

1D Structure factor plotting package - displays reflections as sticks proportional to F, F**2, etc. as requested

GSASII.GSASIIplot.**Plot3DSngl**(*G2frame*, *newPlot=False*, *Data=None*, *hklRef=None*, *Title=False*)

3D Structure factor plotting package - displays reflections as spots proportional to F, F**2, etc. as requested as 3D array via pyOpenGl

GSASII.GSASIIplot.**Plot3dXYZ**(*G2frame*, *nX*, *nY*, *Zdat*, *labelX='X'*, *labelY='Y'*, *labelZ='Z'*, *newPlot=False*, *Title=''*, *Centro=False*)

Creates a surface Plot for 3D vectors

GSASII.GSASIIplot.**PlotAAProb**(*G2frame*, *resNames*, *Probs1*, *Probs2*, *Title=''*, *thresh=None*, *pickHandler=None*)

Needs a description

GSASII.GSASIIplot.**PlotBarGraph**(*G2frame*, *Xarray*, *Xname=''*, *Yname='Number'*, *Title=''*, *PlotName=None*, *ifBinned=False*, *maxBins=None*)

does a vertical bar graph

GSASII.GSASIIplot.**PlotBeadModel**(*G2frame*, *Atoms*, *defaults*, *PDBtext*)

Bead modelplotting package. For bead models from SHAPES

GSASII.GSASIIplot.**PlotCalib**(*G2frame*, *Inst*, *XY*, *Sigs*, *newPlot=False*)

plot of CW or TOF peak calibration

GSASII.GSASIIplot.**PlotClusterXYZ**(*G2frame*, *YM*, *XYZ*, *CLuDict*, *Title=''*, *PlotName='cluster'*)

To plot cluster analysis results

Parameters

- **G2frame** (*wx.Frame*) – The main GSAS-II tree "window"
- **YM** (*array*) – data matrix; plotted as contour
- **XYZ** (*array*) – array of 3D PCA coordinates; plotted as 3D scatter plot
- **CLuDict** (*dict*) – Cluster info; may have dendrogram & Kmeans results

- **Title** (*str*) – plot title
- **PlotName** (*str*) – plot tab name

GSASII.GSASIIplot.**PlotCovariance**(*G2frame*, *Data*, *Cube=False*)

Plots the covariance matrix. Also shows values for parameters and their standard uncertainties (esd's) or the correlation between variables.

GSASII.GSASIIplot.**PlotDeform**(*G2frame*, *general*, *atName*, *atType*, *deform*, *UVmat*, *radial*, *neigh*)

Plot deformation atoms & neighbors

GSASII.GSASIIplot.**PlotDeltSig**(*G2frame*, *kind*, *PatternName=None*)

Produces normal probability plot for a powder or single crystal histogram

GSASII.GSASIIplot.**PlotExposedImage**(*G2frame*, *newPlot=False*, *event=None*)

General access module for 2D image plotting

GSASII.GSASIIplot.**PlotFPAconvolutors**(*G2frame*, *NISTpk*, *conv2T=None*, *convI=None*, *convList=None*)

Plot the convolutions used for the current peak computed with `GSASIIfpaGUI.doFPAcalc()`

GSASII.GSASIIplot.**PlotISFG**(*G2frame*, *data*, *newPlot=False*, *plotType=''*, *peaks=None*)

Plotting package for PDF analysis; displays I(Q), S(Q), F(Q) and G(r) as single or multiple plots with waterfall and contour plots as options

GSASII.GSASIIplot.**PlotImage**(*G2frame*, *newPlot=False*, *event=None*, *newImage=True*)

Plot of 2D detector images as contoured plot. Also plot calibration ellipses, masks, etc. Plots whatever is in G2frame.ImageZ

Parameters

- **G2frame** (*wx.Frame*) – main GSAS-II frame
- **newPlot** (*bool*) – if newPlot is True, the plot is reset (zoomed out, etc.)
- **event** – matplotlib mouse event (or None)
- **newImage** (*bool*) – If True, the Figure is cleared and redrawn

GSASII.GSASIIplot.**PlotIntegration**(*G2frame*, *newPlot=False*, *event=None*)

Plot of 2D image after image integration with 2-theta and azimuth as coordinates

GSASII.GSASIIplot.**PlotLayers**(*G2frame*, *Layers*, *laySeq*, *defaults*, *firstCall=False*)

Layer plotting package. Can show layer structures as balls & sticks

GSASII.GSASIIplot.**PlotNamedFloatHBarGraph**(*G2frame*, *Xvals*, *Ynames*, *Xlabel='Value'*, *Ylabel=''*, *Title=''*, *PlotName=None*)

does a horizintal bar graph

GSASII.GSASIIplot.**PlotPeakWidths**(*G2frame*, *PatternName=None*)

Plotting of instrument broadening terms as function of Q Seen when "Instrument Parameters" chosen from powder pattern data tree. Parameter PatternName allows the PWDR to be referenced as a string rather than a wx tree item, defined in G2frame.PatternId.

GSASII.GSASIIplot.**PlotPowderLines**(*G2frame*, *indexFrom=''*)

plotting of powder lines (i.e. no powder pattern) as sticks

GSASII.GSASIIplot.**PlotRama**(*G2frame*, *phaseName*, *Rama*, *RamaName*, *Names=[]*, *PhiPsi=[]*, *Coeff=[]*)

needs a doc string

GSASII.GSASIIplot.**PlotRawImage**(*G2frame*, *image*, *label*, *newPlot=False*)

Plot an image without axes etc.

GSASII.GSASIIplot.**PlotRigidBody**(*G2frame*, *rbType*, *AtInfo*, *rbData*, *defaults*)

RB plotting package. Can show rigid body structures as balls & sticks

GSASII.GSASIIplot.**PlotSASDPairDist**(*G2frame*)

Needs a description

GSASII.GSASIIplot.**PlotSASDSizeDist**(*G2frame*)

Needs a description

GSASII.GSASIIplot.**PlotSelectedSequence**(*G2frame*, *ColumnList*, *TableGet*, *SelectX*, *fitnum=None*, *fitvals=None*)

Plot a result from a sequential refinement

Parameters

- **G2frame** (*wx.Frame*) – The main GSAS-II tree "window"
- **ColumnList** (*list*) – list of int values corresponding to columns selected as y values
- **TableGet** (*function*) – a function that takes a column number as argument and returns the column label, the values and there ESDs (or None)
- **SelectX** (*function*) – a function that returns a selected column number (or None) as the X-axis selection

GSASII.GSASIIplot.**PlotSizeStrainPO**(*G2frame*, *data*, *hist=''*)

Plot 3D mustrain/size/preferred orientation figure. In this instance data is for a phase

GSASII.GSASIIplot.**PlotSngl**(*G2frame*, *newPlot=False*, *Data=None*, *hklRef=None*, *Title=''*)

Structure factor plotting package - displays zone of reflections as rings proportional to F, F**2, etc. as requested via matpltlib; plots are not geometrically correct

GSASII.GSASIIplot.**PlotStrain**(*G2frame*, *data*, *newPlot=False*)

plot of strain data, used for diagnostic purposes

GSASII.GSASIIplot.**PlotStructure**(*G2frame*, *data*, *firstCall=False*, *pageCallback=None*)

Crystal structure plotting package. Can show structures as balls, sticks, lines, thermal motion ellipsoids and polyhedra. Magnetic moments shown as black/red arrows according to spin state

Parameters

- **G2frame** (*wx.Frame*) – main GSAS-II window
- **data** (*dict*) – dict with plotting information (see *Phase Tree object*)
- **firstCall** (*bool*) – If True, this is the initial call and causes the plot to be shown twice (needed for Mac and possibly linux)
- **pageCallback** (*function*) – a callback function to update items on the parent page. Currently implemented for RB Models tab only

GSASII.GSASIIplot.**PlotTRImage**(*G2frame*, *tax*, *tay*, *taz*, *newPlot=False*)

a test plot routine - not normally used

GSASII.GSASIIplot.**PlotTexture**(*G2frame*, *data*, *Start=False*)

Pole figure, inverse pole figure plotting. dict generalData contains all phase info needed which is in data

GSASII.GSASIIplot.**PlotTorsion**(*G2frame*, *phaseName*, *Torsion*, *TorName*, *Names=[]*, *Angles=[]*, *Coeff=[]*)

needs a doc string

GSASII.GSASIIplot.**PlotXY**(*G2frame*, *XY*, *XY2=[]*, *labelX='X'*, *labelY='Y'*, *newPlot=False*, *Title=''*, *lines=False*, *points2=False*, *names=[]*, *names2=[]*, *vertLines=[]*)

simple plot of xy data

Parameters

- **G2frame** (*wx.Frame*) – The main GSAS-II tree "window"
- **XY** (*list*) – a list of X,Y array pairs; len(X) = len(Y)
- **XY2** (*list*) – a secondary list of X,Y pairs
- **labelX** (*str*) – label for X-axis
- **labelY** (*str*) – label for Y-axis
- **newPlot** (*bool*) – =True if new plot is to be made
- **Title** (*str*) – title for plot
- **lines** (*bool*) – = True if lines desired for XY plot; XY2 always plotted as lines
- **points2** (*bool*) – = False if XY2 is plotted as points despite lines
- **names** (*list*) – legend names for each XY plot as list a of str values
- **names2** (*list*) – legend names for each XY2 plot as list a of str values
- **vertLines** (*list*) – lists of vertical line x-positions; can be one for each XY

Returns

nothing

GSASII.GSASIIplot.**PlotXYZ**(*G2frame*, *XY*, *Z*, *labelX='X'*, *labelY='Y'*, *newPlot=False*, *Title=''*, *zrange=None*, *color=None*, *buttonHandler=None*)

simple contour plot of xyz data

Parameters

- **G2frame** (*wx.Frame*) – The main GSAS-II tree "window"
- **XY** (*list*) – a list of X,Y arrays
- **Z** (*list*) – a list of Z values for each X,Y pair
- **labelX** (*str*) – label for X-axis
- **labelY** (*str*) – label for Y-axis
- **newPlot** (*bool*) – =True if new plot is to be made
- **Title** (*str*) – title for plot
- **zrange** (*list*) – [zmin,zmax]; default=None to use limits in Z
- **color** (*str*) – one of mpl.cm.dated.keys(); default=None to use G2frame.ContourColor

Returns

nothing

GSASII.GSASIIplot.**PlotXYZvect**(*G2frame*, *X*, *Y*, *Z*, *R*, *labelX='X'*, *labelY='Y'*, *labelZ='Z'*, *Title=''*, *PlotName=None*)

To plot a quiver of quaternion vectors colored by the rotation

Parameters

- **G2frame** (*wx.Frame*) – The main GSAS-II tree "window"
- **X,Y,Z** (*list*) – list of X,Y,Z arrays
- **R** (*list*) – a list of rotations (0-90) for each X,Y,Z in degrees
- **labelX,labelY,labelZ** (*str*) – labels for X,Y,Z-axes
- **Title** (*str*) – plot title
- **PlotName** (*str*) – plot tab name

GSASII.GSASIIplot.**ToggleMultiSpotMask**(*G2frame*)

Turns on and off MultiSpot selection mode; displays a subtitle on plot the is cleared by the next PlotImage call

GSASII.GSASIIplot.**UpdatePolygon**(*pick*, *event*, *polygon*)

Update a polygon (or frame) in response to the location of the mouse. Delete the selected point if moved on top of another. With right button add a point after the current button.

GSASII.GSASIIplot.**Write2csv**(*fil*, *dataItems*, *header=False*)

Write a line to a CSV file

Parameters

- **fil** (*object*) – file object
- **dataItems** (*list*) – items to write as row in file
- **header** (*bool*) – True if all items should be written with quotes (default is False)

GSASII.GSASIIplot.**_Old_Paired_data = {'blue': [(0.0, 0.8901960849761963, 0.8901960849761963), (0.09090909090909091, 0.7058823704719543, 0.7058823704719543), (0.18181818181818182, 0.5411764979362488, 0.5411764979362488), (0.2727272727272727, 0.1725490242242813, 0.1725490242242813), (0.36363636363636365, 0.6000000238418579, 0.6000000238418579), (0.45454545454545453, 0.10980392247438431, 0.10980392247438431), (0.5454545454545454, 0.43529412150382996, 0.43529412150382996), (0.6363636363636364, 0.0, 0.0), (0.7272727272727273, 0.8392156958580017, 0.8392156958580017), (0.8181818181818182, 0.6039215922355652, 0.6039215922355652), (0.9090909090909091, 0.6000000238418579, 0.6000000238418579), (1.0, 0.1568627506494522, 0.1568627506494522)], 'green': [(0.0, 0.8078431487083435, 0.8078431487083435), (0.09090909090909091, 0.47058823704719543, 0.47058823704719543), (0.18181818181818182, 0.8745098114013672, 0.8745098114013672), (0.2727272727272727, 0.6274510025978088, 0.6274510025978088), (0.36363636363636365, 0.6039215922355652, 0.6039215922355652), (0.45454545454545453, 0.10196078568696976, 0.10196078568696976), (0.5454545454545454, 0.7490196228027344, 0.7490196228027344), (0.6363636363636364, 0.49803921580314636, 0.49803921580314636), (0.7272727272727273, 0.6980392336845398, 0.6980392336845398), (0.8181818181818182, 0.239215686917305, 0.239215686917305), (0.9090909090909091, 1.0, 1.0), (1.0, 0.3490196168422699, 0.3490196168422699)], 'red': [(0.0, 0.6509804129600525, 0.6509804129600525), (0.09090909090909091, 0.12156862765550613, 0.12156862765550613), (0.18181818181818182, 0.6980392336845398, 0.6980392336845398), (0.2727272727272727, 0.20000000298023224, 0.20000000298023224), (0.36363636363636365, 0.9843137264251709, 0.9843137264251709), (0.45454545454545453, 0.8901960849761963, 0.8901960849761963), (0.5454545454545454, 0.9921568632125854, 0.9921568632125854), (0.6363636363636364, 1.0, 1.0), (0.7272727272727273, 0.7921568751335144, 0.7921568751335144), (0.8181818181818182, 0.4156862795352936, 0.4156862795352936), (0.9090909090909091, 1.0, 1.0), (1.0, 0.6941176652908325, 0.6941176652908325)]}**

In matplotlib 2.0.x+ the Paired color map was dumbed down to 16 colors. _Old_Paired_data is the pre-2.0 Paired color map found in matplotlib._cm.py and is used to create color map GSPaired.

This can be done on request for other color maps. N.B. any new names must be explicitly added to the color list obtained from mpl.cm.datad.keys() (currently 10 places in GSAS-II code).

class GSASII.GSASIIplot.**_tabPlotWin**(*parent*, *id=-1*, *dpi=None*, ***kwargs*)

Creates a basic tabbed plot window for GSAS-II graphics

__init__(*parent*, *id=-1*, *dpi=None*, ***kwargs*)

GSASII.GSASIIplot.**onLegendPick**(*event*)

When a line in the legend is selected, find the matching line in the plot and then highlight it by adding/enlarging markers. Set up a timer to make a reset after delay selected in SetupLegendPick

15.5 *GSASIIpwdplot Routines*

Classes and routines defined in GSASIIpwdplot follow.

GSASII.GSASIIpwdplot.**CopyRietveldPlot**(*G2frame*, *Pattern*, *Plot*, *Page*, *figure*, *phaseList*)

Copy the contents of the Rietveld graph from the plot window to another mpl figure which can be on screen or can be a file for hard copy. Uses values from Pattern to also generate a delta/sigma plot below the main figure, since the weights are not available from the plot.

Parameters

- **Pattern** (*list*) – histogram object from data tree
- **Plot** (*mpl.axes*) – The axes object from the Rietveld plot
- **Page** (*wx.Panel*) – The tabbed panel for the Rietveld plot
- **figure** (*matplotlib.figure.Figure*) – The figure object from the Rietveld plot

GSASII.GSASIIpwdplot.**PlotPatterns**(*G2frame*, *newPlot=False*, *plotType='PWDR'*, *data=None*, *extraKeys=[]*, *refineMode=False*, *indexFrom=''*, *fromTree=False*)

Powder pattern plotting package - displays single or multiple powder patterns as intensity vs 2-theta, q or TOF. Can display multiple patterns as "waterfall plots" or contour plots. Log I plotting available.

Note that information needed for plotting will be found in:

- G2frame.PatternId: contains the tree item for the current histogram
- G2frame.PickId: contains the actual selected tree item (can be child of histogram)
- G2frame.HKL: used for tooltip display of hkl for a selected/generated phase's reflections when mouse is moved to a reflection location; HKL locations shown (usually as an orange line) in "Index Peak List" & "Unit Cells List" plots. N.B. reflection tick markers are generated from each phase's reflection list.
- G2frame.Extinct: used for display of extinct reflections (in blue) for generated reflections when "show extinct" is selected.

Parameters

- **G2frame** (*wx.Frame*) – main GSAS-II window
- **newPlot** – Set to True when a new type of plot is drawn (default False)
- **plotType** – Type of data entry to be plotted (SASD, REFD, PWDR) (default is 'PWDR')
- **data** (*list*) – Contents of histogram
- **extraKeys** (*list*) – list of str values with extra "command" keys to act on plot
- **refineMode** (*bool*) – Set to True when called from inside a refinement default is False

- **indexFrom** (*str*) – Status line message used to label indexing results
- **fromTree** (*bool*) – will be set to True when called from the data tree

Returns
if refineMode is True, returns a reference to `refPlotUpdate()`. Otherwise, nothing is returned

GSASII.GSASIIpwdplot.**PublishRietveldPlot**(*G2frame*, *Pattern*, *Plot*, *Page*, *reuse=None*)

Creates a window to show a customizable "Rietveld" plot. Exports that plot as a publication-quality file. Will only work only when a single pattern is displayed.

Parameters

- **G2Frame** (*wx.Frame*) – the main GSAS-II window
- **Pattern** (*list*) – list of np.array items with obs, calc (etc.) diffraction pattern
- **Plot** (*mpl.axes*) – axes of the graph in plot window
- **Page** (*wx.Panel*) – tabbed panel containing the plot

GSASII.GSASIIpwdplot.**ReplotPattern**(*G2frame*, *newPlot*, *plotType*, *PatternName=None*, *PickName=None*)

This does the same as PlotPatterns except that it expects the information to be plotted (pattern name, item picked in tree + eventually the reflection list) to be passed as names rather than references to wx tree items, defined as class entries

GSASII.GSASIIpwdplot.**changePlotSettings**(*G2frame*, *Plot*)

Code in development to allow changes to plot settings prior to export of plot with "floppy disk" button

GSASII.GSASIIpwdplot.**configPartialDisplay**(*G2frame*, *phaseColors*, *RefreshPlot*)

Select which phase are diplayed with phase partials and how they are displayed

GSASII.GSASIIpwdplot.**initPartialOpts**(*phaseColors*)

Make sure that the options for display of partials are all defined

GSASII.GSASIIpwdplot.**plotVline**(*Page*, *Plot*, *Lines*, *Parms*, *pos*, *color*, *pickrad*, *style='dotted'*)

shortcut to plot vertical lines for limits & Laue satellites. Was used for extrapeaks

GSASII.GSASIIpwdplot.**uneqImgShow**(*figure*, *ax*, *Xlist*, *Ylist*, *cmap*, *vmin*, *vmax*, *Ylbls=[]*)

Plots a contour plot where point spacing varies within a dataset and where the X values may differ between histograms. Note that the length of Xlist and Ylist must be the same and will be the number of histograms to be plotted

Parameters

- **figure** (*matplotlib.figure*) – The figure where the plot will be placed.
- **ax** (*matplotlib.axes*) – The axes where the plot will be made.
- **Xlist** (*list*) – A list of X values for each histogram.
- **Ylist** (*list*) – A list of intensities for each histogram.
- **cmap** (*matplotlib.colormap*) – The colormap used for shading intensities.
- **vmin** (*float*) – Minimum intensity.
- **vmax** (*float*) – float Maximum intensity.
- **Ylbls** (*list*) – Optional. Label to place on each histogram. The default is [] where the axes are labeled normally with the first histogram numbered starting at 0.

CHAPTER

SIXTEEN

GSASIIPWD: POWDER CALCULATIONS

16.1 Summary/Contents

Routines for powder pattern computations, includes peak fitting, creation of PDF fitting scripts, interfaces to DIFFaX & Dysnomia.

Section Contents

16.2 GSASIIpwd Classes and routines

Classes and routines defined in `GSASIIpwd` follow.

`GSASII.GSASIIpwd.`**`Absorb`**(*Geometry*, *MuR*, *Tth*, *Phi=0*, *Psi=0*)

Calculate sample absorption

Parameters

- **`Geometry`** (*str*) – one of 'Cylinder','Bragg-Brentano','Tilting Flat Plate in transmission','Fixed flat plate'
- **`MuR`** (*float*) – absorption coeff * sample thickness/2 or radius
- **`Tth`** – 2-theta scattering angle - can be numpy array
- **`Phi`** (*float*) – flat plate tilt angle - future
- **`Psi`** (*float*) – flat plate tilt axis - future

`GSASII.GSASIIpwd.`**`AbsorbDerv`**(*Geometry*, *MuR*, *Tth*, *Phi=0*, *Psi=0*)

needs a doc string

`GSASII.GSASIIpwd.`**`CalcPDF`**(*data*, *inst*, *limits*, *xydata*)

Computes I(Q), S(Q) & G(r) from Sample, Bkg, etc. diffraction patterns loaded into dict xydata; results are placed in xydata. Calculation parameters are found in dicts data and inst and list limits. The return value is at present an empty list.

GSASII.GSASIIpwd.**Dict2Values**(*parmdict*, *varylist*)

Use before call to leastsq to setup list of values for the parameters in parmdict, as selected by key in varylist

GSASII.GSASIIpwd.**DoPeakFit**(*FitPgm*, *Peaks*, *Background*, *Limits*, *Inst*, *Inst2*, *data*, *fixback=None*, *prevVaryList=[]*, *oneCycle=False*, *controls=None*, *wtFactor=1.0*, *dlg=None*, *noFit=False*)

Called to perform a peak fit, refining the selected items in the peak table as well as selected items in the background.

Parameters

- **FitPgm** (*str*) – type of fit to perform. At present this is ignored.
- **Peaks** (*list*) – a list of peaks. Each peak entry is a list with paired values: The number of pairs depends on the data type (see *getHeaderInfo()*). For CW data there are four values each followed by a refine flag where the values are: position, intensity, sigma (Gaussian width) and gamma (Lorentzian width). From the Histogram/"Peak List" tree entry, dict item "peaks". For some types of fits, overall parameters are placed in a dict entry.
- **Background** (*list*) – describes the background. List with two items. Item 0 specifies a background model and coefficients. Item 1 is a dict. From the Histogram/Background tree entry.
- **Limits** (*list*) – min and max x-value to use
- **Inst** (*dict*) – Instrument parameters
- **Inst2** (*dict*) – more Instrument parameters
- **data** (*numpy.array*) – a 5xn array. data[0] is the x-values, data[1] is the y-values, data[2] are weight values, data[3], [4] and [5] are calc, background and difference intensities, respectively.
- **fixback** (*array*) – fixed background array; same size as data[0-5]
- **prevVaryList** (*list*) – Used in sequential refinements to override the variable list. Defaults as an empty list.
- **oneCycle** (*bool*) – True if only one cycle of fitting should be performed
- **controls** (*dict*) – a dict specifying two values, Ftol = controls['min dM/M'] and derivType = controls['deriv type']. If None default values are used.
- **wtFactor** (*float*) – weight multiplier; = 1.0 by default
- **dlg** (*wx.Dialog*) – A dialog box that is updated with progress from the fit. Defaults to None, which means no updates are done.
- **noFit** (*bool*) – When noFit is True, a refinement is not performed. Default is False.

GSASII.GSASIIpwd.**GetAsfMean**(*ElList*, *Sthl2*)

Calculate various scattering factor terms for PDF calcs

Parameters

- **ElList** (*dict*) – element dictionary contains scattering factor coefficients, etc.
- **Sthl2** (*np.array*) – numpy array of sin theta/lambda squared values

Returns

mean(f^2), mean(f)^2, mean(compton)

GSASII.GSASIIpwd.**GetNumDensity**(*ElList*, *Vol*)

needs a doc string

GSASII.GSASIIpwd.**GetPDFfitAtomVar**(*Phase*, *RMCPdict*)

Find dict of independent "@n" variables for PDFfit in atom constraints

GSASII.GSASIIpwd.**GetSeqCell**(*SGData*, *parmDict*)

For use in processing PDFfit sequential results

GSASII.GSASIIpwd.**ISO2PDFfit**(*Phase*)

Creates new phase structure to be used for PDFfit from an ISODISTORT mode displacement phase. It builds the distortion mode parameters to be used as PDFfit variables for atom displacements from the original parent positions as transformed to the child cell wiht symmetry defined from ISODISTORT.

Parameters

Phase – dict GSAS-II Phase structure; must contain ISODISTORT dict. NB: not accessed otherwise

Returns

dict: GSAS-II Phase structure; will contain ['RMC']['PDFfit'] dict

GSASII.GSASIIpwd.**LaueFringePeakCalc**(*ttArr*, *intArr*, *lam*, *peakpos*, *intens*, *sigma2*, *gamma*, *shol*, *ncells*, *clat*, *dampM*, *dampP*, *calcwid*, *fitPowerM=2*, *fitPowerP=2*, *plot=False*)

Compute the peakshape for a Laue Fringe peak convoluted with a Gaussian, Lorentzian & an axial divergence asymmetry correction.

Parameters

- **ttArr** (*np.array*) – Array of two-theta values (in degrees)
- **intArr** (*np.array*) – Array of intensity values (peaks are added to this)
- **lam** (*float*) – wavelength in Angstrom
- **peakpos** (*float*) – peak position in two-theta (deg.)
- **intens** (*float*) – intensity factor for peak
- **sigma2** (*float*) – Gaussian variance (in centidegrees**2) **
- **gamma** (*float*) – Lorenzian FWHM (in centidegrees) **
- **shol** (*float*) – FCJ (S + H)/L where S=sample-half height, H=slit half-height, L=radius **
- **ncells** (*float*) – number of unit cells in specular direction **
- **clat** (*float*) – c lattice parameter **
- **dampM** (*float*)
- **dampP** (*float*)
- **calcwid** (*float*) – two-theta (deg.) width for cutoff of peak computation. Defaults to 5
- **fitPowerM** (*float*) – exponent used for damping fall-off on minus side of peak
- **fitPowerP** (*float*) – exponent used for damping fall-off on plus side of peak
- **plot** (*bool*) – for debugging, shows contributions to peak

** If term is <= zero, item is removed from convolution

GSASII.GSASIIpwd.**LaueSatellite**(*peakpos*, *wave*, *c*, *ncell*, *j=[-4, -3, -2, -1, 0, 1, 2, 3, 4]*)

Returns the locations of the Laue satellite positions relative to the peak position

Parameters

- **peakpos** (*float*) – the peak position in degrees 2theta

- **ncell** (*float*) – Laue fringe parameter, number of unit cells in layer
- **j** (*list*) – the satellite order, where j=-1 is the first satellite on the lower 2theta side and j=1 is the first satellite on the high 2theta side. j=0 gives the peak position

GSASII.GSASIIpwd.**LorchWeight**(*Q*)

needs a doc string

GSASII.GSASIIpwd.**MEMupdateReflData**(*prfName*, *data*, *reflData*)

Update reflection data with new Fosq, phase result from Dysnomia

Parameters

- **prfName** (*str*) – phase.mem file name
- **reflData** (*list*) – GSAS-II reflection data

GSASII.GSASIIpwd.**MakePDFfitAtomsFile**(*Phase*, *RMCPdict*)

Make the PDFfit atoms file

GSASII.GSASIIpwd.**MakePDFfitRunFile**(*Phase*, *RMCPdict*)

Make the PDFfit python run file

GSASII.GSASIIpwd.**MakefullrmcRun**(*pName*, *Phase*, *RMCPdict*)

Creates a script to run fullrmc. Returns the name of the file that was created.

GSASII.GSASIIpwd.**MakefullrmcSupercell**(*Phase*, *RMCPdict*)

Create a fullrmc supercell from GSAS-II

Parameters

- **Phase** (*dict*) – phase information from data tree
- **RMCPdict** (*dict*) – fullrmc parameters from GUI
- **grpDict** (*list*) – a list of lists where the inner list contains the atom numbers contained in each group. e.g. [[0,1,2,3,4],[5,6],[4,6]] creates three groups with atoms 0-4 in the first atoms 5 & 6 in the second and atoms 4 & 6 in the third. Note that it is fine that atom 4 appears in two groups.

GSASII.GSASIIpwd.**Oblique**(*ObCoeff*, *Tth*)

currently assumes detector is normal to beam

GSASII.GSASIIpwd.**PhaseWtSum**(*G2frame*, *histo*)

Calculate sum of phase mass*phase fraction for PWDR data (exclude magnetic phases)

Parameters

- **G2frame** – GSASII main frame structure
- **histo** (*str*) – histogram name

Returns

sum(scale*mass) for phases in histo

GSASII.GSASIIpwd.**Polarization**(*Pola*, *Tth*, *Azm=0.0*)

Calculate angle dependent x-ray polarization correction (not scaled correctly!)

Parameters

- **Pola** – polarization coefficient e.g 1.0 fully polarized, 0.5 unpolarized
- **Azm** – azimuthal angle e.g. 0.0 in plane of polarization - can be numpy array

- **Tth** – 2-theta scattering angle - can be numpy array which (if either) of these is "right"?

Returns

(pola, dpdPola) - both 2-d arrays * pola = ((1-Pola)*npcosd(Azm)**2+Pola*npsind(Azm)**2)*npcosd(Tth)**2+ (1-Pola)*npsind(Azm)**2+Pola*npcosd(Azm)**2 * dpdPola: derivative needed for least squares

GSASII.GSASIIpwd.**Ruland**(*RulCoff*, *wave*, *Q*, *Compton*)

needs a doc string

GSASII.GSASIIpwd.**SetBackgroundParms**(*Background*)

Loads background parameters into dicts/lists to create varylist & parmdict

GSASII.GSASIIpwd.**SetDefaultREFDModel**()

Fills in default items for the REFD Models dictionary which are defined as follows for each layer:

- Name: name of substance
- Thick: thickness of layer in Angstroms (not present for top & bottom layers)
- Rough: upper surface roughness for layer (not present for toplayer)
- Penetration: mixing of layer substance into layer above-is this needed?
- DenMul: multiplier for layer scattering density (default = 1.0)

Top layer defaults to vacuum (or air/any gas); can be substituted for some other substance.

Bottom layer default: infinitely thisck Silicon; can be substituted for some other substance.

GSASII.GSASIIpwd.**SetDefaultSASDModel**()

Fills in default items for the SASD Models dictionary

GSASII.GSASIIpwd.**SetDefaultSubstances**()

Fills in default items for the SASD Substances dictionary

GSASII.GSASIIpwd.**StackSim**(*Layers*, *ctrls*, *scale=0.0*, *background={}*, *limits=[]*, *inst={}*, *profile=[]*)

Simulate powder or selected area diffraction pattern from stacking faults using DIFFaX

Parameters

- **Layers** (*dict*) – dict with following items

```
{'Laue':'-1','Cell':[False,1.,1.,1.,90.,90.,90,1.],
'Width':[[10.,10.],[False,False]],'Toler':0.01,'AtInfo':{},
'Layers':[],'Stacking':[],'Transitions':[]}
```

- **ctrls** (*str*) – controls string to be written on DIFFaX controls.dif file
- **scale** (*float*) – scale factor
- **background** (*dict*) – background parameters
- **limits** (*list*) – min/max 2-theta to be calculated
- **inst** (*dict*) – instrument parameters dictionary
- **profile** (*list*) – powder pattern data

Note that parameters all updated in place

GSASII.GSASIIpwd.**SurfaceRough**(*SRA*, *SRB*, *Tth*)

Suortti (J. Appl. Cryst, 5,325-331, 1972) surface roughness correction

Parameters

- **SRA** (*float*) – Suortti surface roughness parameter
- **SRB** (*float*) – Suortti surface roughness parameter
- **Tth** (*float*) – 2-theta(deg) - can be numpy array

GSASII.GSASIIpwd.**SurfaceRoughDerv**(*SRA*, *SRB*, *Tth*)

Suortti surface roughness correction derivatives

Parameters

- **SRA** (*float*) – Suortti surface roughness parameter (dimensionless)
- **SRB** (*float*) – Suortti surface roughness parameter (dimensionless)
- **Tth** (*float*) – 2-theta(deg) - can be numpy array

Return list

[dydSRA,dydSRB] derivatives to be used for intensity derivative

GSASII.GSASIIpwd.**TestData**()

needs a doc string

GSASII.GSASIIpwd.**Transmission**(*Geometry*, *Abs*, *Diam*)

Calculate sample transmission

Parameters

- **Geometry** (*str*) – one of 'Cylinder','Bragg-Brentano','Tilting flat plate in transmission','Fixed flat plate'
- **Abs** (*float*) – absorption coeff in cm-1
- **Diam** (*float*) – sample thickness/diameter in mm

GSASII.GSASIIpwd.**UpdatePDFfit**(*Phase*, *RMCPdict*)

Updates various PDFfit parameters held in GSAS-II

GSASII.GSASIIpwd.**Values2Dict**(*parmdict*, *varylist*, *values*)

Use after call to leastsq to update the parameter dictionary with values corresponding to keys in varylist

GSASII.GSASIIpwd.**abeles**(*kz*, *depth*, *rho*, *irho=0*, *sigma=0*)

Optical matrix form of the reflectivity calculation. O.S. Heavens, Optical Properties of Thin Solid Films

Reflectometry as a function of kz for a set of slabs.

Parameters

- **kz** – float[n] (1/Ang). Scattering vector, $2\pi \sin(\theta)/\lambda$. This is $\frac{1}{2}Q_z$.
- **depth** – float[m] (Ang). thickness of each layer. The thickness of the incident medium and substrate are ignored.
- **rho** – float[n,k] (1e-6/Ang^2) Real scattering length density for each layer for each kz
- **irho** – float[n,k] (1e-6/Ang^2) Imaginary scattering length density for each layer for each kz Note: absorption cross section mu = 2 irho/lambda for neutrons
- **sigma** – float[m-1] (Ang) interfacial roughness. This is the roughness between a layer and the previous layer. The sigma array should have m-1 entries.

Slabs are ordered with the surface SLD at index 0 and substrate at index -1, or reversed if kz < 0.

GSASII.GSASIIpwd.**autoBkgCalc**(*bkgdict*, *ydata*)

Compute the autobackground using the selected pybaselines function

Parameters

- **bkgdict** (*dict*) – background parameters
- **ydata** (*np.array*) – array of Y values

Returns

points for background intensity at each Y position

GSASII.GSASIIpwd.**calcIncident**(*Iparm*, *xdata*)

needs a doc string

class GSASII.GSASIIpwd.**cauchy_gen**(*momtype=1*, *a=None*, *b=None*, *xtol=1e-14*, *badvalue=None*, *name=None*, *longname=None*, *shapes=None*, *seed=None*)

Cauchy distribution

cauchy.pdf(x) = 1/(pi*(1+x**2))

This is the t distribution with one degree of freedom.

pdf(*x*, **args*, ***kwds*)

Probability density function at x of the given RV.

16.2.1 Parameters

x

[array_like] quantiles

arg1, arg2, arg3,…

[array_like] The shape parameter(s) for the distribution (see docstring of the instance object for more information)

loc

[array_like, optional] location parameter (default=0)

scale

[array_like, optional] scale parameter (default=1)

16.2.2 Returns

pdf

[ndarray] Probability density function evaluated at x

GSASII.GSASIIpwd.**ellipseSize**(*H*, *Sij*, *GB*)

Implements r=1/sqrt(sum((1/S)*(q.v)^2) per note from Alexander Brady

GSASII.GSASIIpwd.**ellipseSizeDerv**(*H*, *Sij*, *GB*)

Implements r=1/sqrt(sum((1/S)*(q.v)^2) derivative per note from Alexander Brady

GSASII.GSASIIpwd.**factorize**(*num*)

Provide prime number factors for integer num :returns: dictionary of prime factors (keys) & power for each (data)

class GSASII.GSASIIpwd.**fcjde_gen**(*momtype=1*, *a=None*, *b=None*, *xtol=1e-14*, *badvalue=None*, *name=None*, *longname=None*, *shapes=None*, *seed=None*)

Finger-Cox-Jephcoat D(2phi,2th) function for S/L = H/L Ref: J. Appl. Cryst. (1994) 27, 892-900.

Parameters

- **x** – array -1 to 1
- **t** – 2-theta position of peak
- **s** – sum(S/L,H/L); S: sample height, H: detector opening, L: sample to detector opening distance
- **dx** – 2-theta step size in deg

Returns

for fcj.pdf

- T = x*dx+t
- s = S/L+H/L
- if x < 0:

```
fcj.pdf = [1/sqrt({cos(T)**2/cos(t)**2}-1) - 1/s]/|cos(T)|
```

- if x >= 0: fcj.pdf = 0

pdf(*x*, **args*, ***kwds*)

Probability density function at x of the given RV.

16.2.3 Parameters

x
[array_like] quantiles

arg1, arg2, arg3,…
[array_like] The shape parameter(s) for the distribution (see docstring of the instance object for more information)

loc
[array_like, optional] location parameter (default=0)

scale
[array_like, optional] scale parameter (default=1)

16.2.4 Returns

pdf
[ndarray] Probability density function evaluated at x

GSASII.GSASIIpwd.**findPDFfit**()

Find if PDFfit2 is installed (may be local to GSAS-II). Does the following: :returns: the full path to a python executable or None, if it was not found.

GSASII.GSASIIpwd.**findfullrmc**()

Find where fullrmc is installed. Tries the following:

1. Returns the Config var 'fullrmc_exec', if defined. If an executable is found at that location it is assumed to run and supply fullrmc 5.0+
2. The path is checked for a fullrmc image as named by Bachir

Returns
the full path to a python executable that is assumed to have fullrmc installed or None, if it was not found.

GSASII.GSASIIpwd.**findrmcprofile**()

Find where RMCProfile is installed. Tries the following:

1. Returns the Config var *rmcprofile_exec*, if defined. This is the executable to run.
2. The path is checked for a RMCprofile image. Also checked are the location where the GSAS-II Python files are found, the location where GSAS-II binaries are found, the current working directory and the location where the Python interpreter is found. On MacOS the only place where RMCProfile can be installed is /Applications/ so only that is checked.

Returns

the full path to a python executable that is assumed to have fullrmc installed or None, if it was not found.

GSASII.GSASIIpwd.**fullrmcDownload**()

Downloads the fullrmc executable from Bachir's site to the current GSAS-II binary directory.

Does some error checking.

GSASII.GSASIIpwd.**getBackground**(*pfx*, *parmDict*, *bakType*, *dataType*, *xdata*, *fixback=None*)

Computes the background based on parameters that may be taken from a gpx file or the data tree.

Parameters

- **pfx** (*str*) – histogram prefix (:h:)
- **parmDict** (*dict*) – Refinement parameter values
- **bakType** (*str*) – defines background function to be used. Should be one of these: 'chebyschev', 'cosine', 'chebyschev-1', 'Q^2 power series', 'Q^-2 power series', 'lin interpolate', 'inv interpolate', 'log interpolate'
- **dataType** (*str*) – Code to indicate histogram type (PXC, PNC, PNT,…)
- **xdata** (*MaskedArray*) – independent variable, 2theta (deg*100) or TOF (microsec?)
- **fixback** (*numpy.array*) – Array of fixed background points (length matching xdata) or None

Returns

yb,sumBK where yp is an array of background values (length matching xdata) and sumBK is a list with three values. The sumBK[0] is the sum of all yb values, sumBK[1] is the sum of Debye background terms and sumBK[2] is the sum of background peaks.

GSASII.GSASIIpwd.**getBackgroundDerv**(*hfx*, *parmDict*, *bakType*, *dataType*, *xdata*, *fixback=None*)

Computes the derivatives of the background Parameters passed to this may be pulled from gpx file or data tree. See *getBackground()* for parameter definitions.

Returns

dydb,dyddb,dydpk,dydfb where the first three are 2-D arrays of derivatives with respect to the background terms, the Debye terms and the background peak terms vs. the points in the diffracton pattern. The final 1D array is the derivative with respect to the fixed-background multiplier (= the fixed background values).

GSASII.GSASIIpwd.**getEpsVoigt**(*pos*, *alp*, *bet*, *sig*, *gam*, *xdata*)

Compute the double exponential Pseudo-Voigt convolution function for a neutron TOF & CW pink peak in external Fortran routine

GSASII.GSASIIpwd.**getExpFCJVoigt3**(*pos*, *alp*, *bet*, *sig*, *gam*, *shl*, *xdata*)

Compute the Finger-Cox-Jepcoat modified double exponential Pseudo-Voigt convolution function for a CW powder peak in external Fortran routine

param pos: peak position in degrees param sig: Gaussian variance in centideg^2 param alp: Rise exponential coefficient param bet: Decay exponential coefficient param gam: Lorentzian width in centideg param shl: axial divergence parameter (S+H)/L param xdata: array; profile points for peak to be calculated; bounded by 20FWHM to 50FWHM (or vv)

returns: array: calculated peak function at each xdata returns: integral of peak; nominally = 1.0

GSASII.GSASIIpwd.**getFCJVoigt**(*pos*, *intens*, *sig*, *gam*, *shl*, *xdata*)

Compute the Finger-Cox-Jepcoat modified Voigt function for a CW powder peak by direct convolution. This version is not used.

GSASII.GSASIIpwd.**getFCJVoigt3**(*pos*, *sig*, *gam*, *shl*, *xdata*)

Compute the Finger-Cox-Jepcoat modified Pseudo-Voigt function for a CW powder peak in external Fortran routine

param pos: peak position in degrees param sig: Gaussian variance in centideg^2 param gam: Lorentzian width in centideg param shl: axial divergence parameter (S+H)/L param xdata: array; profile points for peak to be calculated; bounded by 20FWHM to 50FWHM (or vv)

returns: array: calculated peak function at each xdata returns: integral of peak; nominally = 1.0

GSASII.GSASIIpwd.**getFWHM**(*pos*, *Inst*, *N=1*)

Compute total FWHM from Thompson, Cox & Hastings (1987) , J. Appl. Cryst. 20, 79-83 via getgamFW(g,s).

Parameters

- **pos** – float peak position in deg 2-theta or tof in musec
- **Inst** – dict instrument parameters
- **N** – int Inst index (0 for input, 1 for fitted)

Returns float
total FWHM of pseudoVoigt in deg or musec

GSASII.GSASIIpwd.**getHKLMpeak**(*dmin*, *Inst*, *SGData*, *SSGData*, *Vec*, *maxH*, *A*)

needs a doc string

GSASII.GSASIIpwd.**getHKLpeak**(*dmin*, *SGData*, *A*, *Inst=None*, *nodup=False*)

Generates allowed by symmetry reflections with d >= dmin NB: GenHKLf & checkMagextc return True for extinct reflections

Parameters

- **dmin** – minimum d-spacing
- **SGData** – space group data obtained from SpcGroup
- **A** – lattice parameter terms A1-A6
- **Inst** – instrument parameter info

Returns
HKLs: np.array hkl, etc for allowed reflections

GSASII.GSASIIpwd.**getHeaderInfo**(*dataType*)

Provide parameter name, label name and formatting information for the contents of the Peak Table and where used in DoPeakFit

GSASII.GSASIIpwd.**getPeakProfile**(*dataType*, *parmDict*, *xdata*, *fixback*, *varyList*, *bakType*)

Computes the profiles from multiple peaks for individual peak fitting for powder patterns. NB: not used for Rietveld refinement

GSASII.GSASIIpwd.**getPeakProfileDerv**(*dataType*, *parmDict*, *xdata*, *fixback*, *varyList*, *bakType*)

Computes the profile derivatives for a powder pattern for single peak fitting

return: np.array([dMdx1,dMdx2,…]) in same order as varylist = backVary,insVary,peakVary order

NB: not used for Rietveld refinement

GSASII.GSASIIpwd.**getPsVoigt**(*pos*, *sig*, *gam*, *xdata*)

Compute the simple Pseudo-Voigt function for a small angle Bragg peak in external Fortran routine

param pos: peak position in degrees param sig: Gaussian variance in centideg^2 param gam: Lorentzian width in centideg param xdata: array; profile points for peak to be calculated

returns: array: calculated peak function at each xdata returns: integral of peak; nominally = 1.0

GSASII.GSASIIpwd.**getTOFwids**(*dsp*, *varyList*, *iPeak*, *parmDict*, *applyMax=True*)

For TOF peaks: determine the values for the four peak width parameters from either the lookup table (in parmDict['pdabc']) or from the instrument parameters, unless the peak is varied or peakInstPrmMode is set, in which case the value is taken from the parameter dictionary

applyMax is used in peak fitting to make sure that values stay "sane" but is set to False to compute values that are used for plotting.

GSASII.GSASIIpwd.**getWidthsCW**(*pos*, *sig*, *gam*, *shl*)

Compute the peak widths used for computing the range of a peak for constant wavelength data. On low-angle side, 50 FWHM are used, on high-angle side 75 are used, low angle side extended for axial divergence (for peaks above 90 deg, these are reversed.)

Parameters

- **pos** – peak position; 2-theta in degrees
- **sig** – Gaussian peak variance in centideg^2
- **gam** – Lorentzian peak width in centidegrees
- **shl** – axial divergence parameter (S+H)/L

Returns

widths; [Gaussian sigma, Lorentzian gamma] in degrees, and low angle, high angle ends of peak; 50 FWHM & 75 FWHM from position reversed for 2-theta > 90 deg.

GSASII.GSASIIpwd.**getWidthsCWA**(*pos*, *alp*, *bet*, *sig*, *gam*, *shl*)

Compute the peak widths used for computing the range of a peak for constant wavelength data with axial divergence. 50 & 75 FWHM are used on each side each extended by exponential coeff.

Parameters

- **pos** – peak position; 2-theta in degrees
- **alp,bet** – TOF peak exponential rise & decay parameters
- **sig** – Gaussian peak variance in centideg^2
- **gam** – Lorentzian peak width in centidegrees
- **shl** – axial divergence parameter (S+H)/L

Returns
widths; [Gaussian sigma, Lorentzian gamma] in degrees, and low angle, high angle ends of peak; 50 FWHM & 75 FWHM from position reversed for 2-theta > 90 deg.

GSASII.GSASIIpwd.**getWidthsCWB**(*pos*, *alp*, *bet*, *sig*, *gam*)

Compute the peak widths used for computing the range of a peak for constant wavelength data without axial divergence. 50 FWHM are used on both sides each extended by exponential coeff.

Parameters

- **pos** – peak position; 2-theta in degrees
- **alp,bet** – TOF peak exponential rise & decay parameters
- **sig** – Gaussian peak variance in centideg^2
- **gam** – Lorentzian peak width in centidegrees

returns: widths; [Gaussian sigma, Lornetzian gamma] in degrees returns: low angle, high angle ends of peak; 50FWHM from position

GSASII.GSASIIpwd.**getWidthsED**(*pos*, *sig*, *gam*)

Compute the peak widths used for computing the range of a peak for energy dispersive data. On low-energy side, 20 FWHM are used, on high-energy side 20 are used

Parameters

- **pos** – peak position; energy in keV (not used)
- **sig** – Gaussian peak variance in keV^2
- **gam** – Lorentzian peak width in keV

Returns
widths; [Gaussian sigma, Lorentzian gamma] in keV, and low angle, high angle ends of peak; 5 FWHM & 5 FWHM from position

GSASII.GSASIIpwd.**getWidthsTOF**(*pos*, *alp*, *bet*, *sig*, *gam*)

Compute the peak widths used for computing the range of a peak for TOF data. 50 FWHM are used on both sides each extended by exponential coeff.

param pos: peak position; TOF in musec (not used) param alp,bet: TOF peak exponential rise & decay parameters param sig: Gaussian peak variance in musec^2 param gam: Lorentzian peak width in musec

returns: widths; [Gaussian sigma, Lornetzian gamma] in musec returns: low TOF, high TOF ends of peak; 50FWHM from position

GSASII.GSASIIpwd.**getdEpsVoigt**(*pos*, *alp*, *bet*, *sig*, *gam*, *xdata*)

Compute the double exponential Pseudo-Voigt convolution function derivatives for a neutron TOF & CW pink peak in external Fortran routine

GSASII.GSASIIpwd.**getdExpFCJVoigt3**(*pos*, *alp*, *bet*, *sig*, *gam*, *shl*, *xdata*)

Compute analytic derivatives the Finger-Cox-Jepcoat modified double exponential Pseudo-Voigt convolution function for a CW powder peak

param pos: peak position in degrees param alp: Rise exponential coefficient param bet: Decay exponential coefficient param sig: Gaussian variance in centideg^2 param gam: Lorentzian width in centideg param shl: axial divergence parameter (S+H)/L param xdata: array; profile points for peak to be calculated; bounded by 20FWHM to 50FWHM (or vv)

returns: arrays: function and derivatives of pos, alp, bet, sig, gam, & shl

GSASII.GSASIIpwd.**getdFCJVoigt3**(*pos*, *sig*, *gam*, *shl*, *xdata*)

Compute analytic derivatives the Finger-Cox-Jepcoat modified Pseudo-Voigt function for a CW powder peak

param pos: peak position in degrees param sig: Gaussian variance in centideg^2 param gam: Lorentzian width in centideg param shl: axial divergence parameter (S+H)/L param xdata: array; profile points for peak to be calculated; bounded by 20FWHM to 50FWHM (or vv)

returns: arrays: function and derivatives of pos, sig, gam, & shl

GSASII.GSASIIpwd.**getdPsVoigt**(*pos*, *sig*, *gam*, *xdata*)

Compute the simple Pseudo-Voigt function derivatives for a small angle Bragg peak peak in external Fortran routine

param pos: peak position in degrees param sig: Gaussian variance in centideg^2 param gam: Lorentzian width in centideg param xdata: array; profile points for peak to be calculated

returns: arrays: function and derivatives of pos, sig & gam NB: the pos derivative has the opposite sign compared to that in other profile functions

GSASII.GSASIIpwd.**getgamFW**(*g*, *s*)

Compute total FWHM from Thompson, Cox & Hastings (1987), J. Appl. Cryst. 20, 79-83 lambda fxn needs FWHM for both Gaussian & Lorentzian components

Parameters

- **g** – float Lorentzian gamma = FWHM(L)
- **s** – float Gaussian sig

Returns float
total FWHM of pseudoVoigt

GSASII.GSASIIpwd.**makeFFTsizeList**(*nmin=1*, *nmax=1023*, *thresh=15*)

Provide list of optimal data sizes for FFT calculations

Parameters

- **nmin** (*int*) – minimum data size >= 1
- **nmax** (*int*) – maximum data size > nmin
- **thresh** (*int*) – maximum prime factor allowed

Returns
list of data sizes where the maximum prime factor is < thresh

GSASII.GSASIIpwd.**makeMEMfile**(*data*, *reflData*, *MEMtype*, *DYSNOMIA*)

make Dysnomia .mem file of reflection data, etc.

Parameters

- **data** (*dict*) – GSAS-II phase data
- **reflData** (*list*) – GSAS-II reflection data
- **MEMtype** (*int*) – 1 for neutron data with negative scattering lengths 0 otherwise
- **DYSNOMIA** (*str*) – path to dysnomia.exe

GSASII.GSASIIpwd.**makePRFfile**(*data*, *MEMtype*)

makes Dysnomia .prf control file from Dysnomia GUI controls

Parameters

- **data** (*dict*) – GSAS-II phase data
- **MEMtype** (*int*) – 1 for neutron data with negative scattering lengths 0 otherwise

Returns str
name of Dysnomia control file

class GSASII.GSASIIpwd.**norm_gen**(*momtype=1*, *a=None*, *b=None*, *xtol=1e-14*, *badvalue=None*, *name=None*, *longname=None*, *shapes=None*, *seed=None*)

Normal distribution

The location (loc) keyword specifies the mean. The scale (scale) keyword specifies the standard deviation.

normal.pdf(x) = exp(-x**2/2)/sqrt(2*pi)

pdf(*x*, **args*, ***kwds*)

Probability density function at x of the given RV.

16.2.5 Parameters

x
[array_like] quantiles

arg1, arg2, arg3,…
[array_like] The shape parameter(s) for the distribution (see docstring of the instance object for more information)

loc
[array_like, optional] location parameter (default=0)

scale
[array_like, optional] scale parameter (default=1)

16.2.6 Returns

pdf
[ndarray] Probability density function evaluated at x

GSASII.GSASIIpwd.**peakInstPrmMode = True**

Determines the mode used for peak fitting. When peakInstPrmMode=True peak width parameters are computed from the instrument parameters (UVW,… or alpha,… etc) unless the individual parameter is refined. This allows the instrument parameters to be refined. When peakInstPrmMode=False, the instrument parameters are not used and cannot be refined. The default is peakFitMode=True. This is changed only in *setPeakInstPrmMode()*, which is called from GSASIIscriptable or GSASIIphsGUI.OnSetPeakWidMode ('Gen unvaried widths' menu item).

GSASII.GSASIIpwd.**setPeakInstPrmMode**(*normal=True*)

Determines the mode used for peak fitting. If normal=True (default) peak width parameters are computed from the instrument parameters unless the individual parameter is refined. If normal=False, peak widths are used as supplied for each peak.

Note that normal=True unless this routine is called. Also, instrument parameters can only be refined with normal=True.

Parameters
normal (*bool*) – setting to apply to global variable *peakInstPrmMode*

CHAPTER

SEVENTEEN

GSAS-II SMALL ANGLE SCATTERING

17.1 *GSASII small angle calculation module*

Classes and routines defined in `GSASIIsasd` follow.

`GSASII.GSASIIsasd.`**`CylinderARFF`**(*Q*, *R*, *args*)

Compute form factor for cylinders - can use numpy arrays

Parameters

- **`Q`** (*`float`*) – Q value array (A-1)
- **`R`** (*`float`*) – cylinder radius (A)
- **`args`** (*`array`*) – [float AR]: cylinder aspect ratio = L/D = L/2R

Returns float
form factor

`GSASII.GSASIIsasd.`**`CylinderARVol`**(*R*, *args*)

Compute cylinder volume for radius & aspect ratio = L/D - numpy array friendly

Parameters

- **`float`** – R radius (A)
- **`args`** (*`array`*) – [float AR]: =L/D=L/2R aspect ratio

Returns float
volume

`GSASII.GSASIIsasd.`**`CylinderDFF`**(*Q*, *L*, *args*)

Compute form factor for cylinders - can use numpy arrays

Parameters

- **`Q`** (*`float`*) – Q value array (A-1)
- **`L`** (*`float`*) – cylinder half length (A)
- **`args`** (*`array`*) – [float R]: cylinder radius (A)

Returns float
form factor

`GSASII.GSASIIsasd.`**`CylinderDVol`**(*L*, *args*)

Compute cylinder volume for length & diameter - numpy array friendly

Parameters

- **`float`** – L half length (A)

- **args** (*array*) – [float D]: diameter (A)

Returns float
volume (A^3)

GSASII.GSASIIsasd.**CylinderFF**(*Q*, *R*, *args*)

Compute form factor for cylinders - can use numpy arrays

Parameters

- **Q** (*float*) – Q value array (A-1)
- **R** (*float*) – cylinder radius (A)
- **args** (*array*) – [float L]: cylinder length (A)

Returns float
form factor

GSASII.GSASIIsasd.**CylinderVol**(*R*, *args*)

Compute cylinder volume for radius & length - numpy array friendly

Parameters

- **R** (*float*) – diameter (A)
- **args** (*array*) – [float L]: length (A)

Returns float
volume (A^3)

GSASII.GSASIIsasd.**DiluteSF**(*Q*, *args=[]*)

Default: no structure factor correction for dilute system

GSASII.GSASIIsasd.**G_matrix**(*q*, *r*, *contrast*, *FFfxn*, *Volfxn*, *args=()*)

Calculates the response matrix $G(Q,r)$

Parameters

- **q** (*float*) – Q
- **r** (*float*) – r
- **contrast** (*float*) – $|\Delta\rho|^2$, the scattering contrast
- **FFfxn** (*function*) – form factor function FF(q,r,args)
- **Volfxn** (*function*) – volume function Vol(r,args)

Returns float
G(Q,r)

GSASII.GSASIIsasd.**GaussCume**(*x*, *pos*, *args*)

Standard Normal cumulative distribution - numpy friendly on x axis

Parameters

- **x** (*float*) – independent axis (can be numpy array)
- **pos** (*float*) – location of distribution
- **scale** (*float*) – width of distribution (sigma)
- **shape** (*float*) – not used

Returns float
Normal cumulative distribution

GSASII.GSASIIsasd.**GaussDist**(*x*, *pos*, *args*)

Standard Normal distribution - numpy friendly on x axis

Parameters

- **x** (*float*) – independent axis (can be numpy array)
- **pos** (*float*) – location of distribution
- **scale** (*float*) – width of distribution (sigma)
- **shape** (*float*) – not used

Returns float
Normal distribution

GSASII.GSASIIsasd.**HardSpheresSF**(*Q*, *args*)

Computes structure factor for not dilute monodisperse hard spheres Refs.: PERCUS,YEVICK PHYS. REV. 110 1 (1958),THIELE J. CHEM PHYS. 39 474 (1968), WERTHEIM PHYS. REV. LETT. 47 1462 (1981)

Parameters

- **Q** (*float*) – Q value array (A-1)
- **args** (*array*) – [float R, float VolFrac]: interparticle distance & volume fraction

Returns numpy array
S(Q)

GSASII.GSASIIsasd.**IPG**(*datum*, *sigma*, *G*, *Bins*, *Dbins*, *IterMax*, *Qvec=[]*, *approach=0.8*, *Power=-1*, *report=False*)

An implementation of the Interior-Point Gradient method of Michael Merritt & Yin Zhang, Technical Report TR04-08, Dept. of Comp. and Appl. Math., Rice Univ., Houston, Texas 77005, U.S.A. found on the web at http://www.caam.rice.edu/caam/trs/2004/TR04-08.pdf Problem addressed: Total Non-Negative Least Squares (TNNLS)

Parameters

- **datum** (*float[]*)
- **sigma** (*float[]*)
- **G** (*float[][]*) – transformation matrix
- **IterMax** (*int*)
- **Qvec** (*float*) – data positions for Power = 0-4
- **approach** (*float*) – 0.8 default fitting parameter
- **Power** (*int*) – 0-4 for Q^Power weighting, -1 to use input sigma

GSASII.GSASIIsasd.**InterPrecipitateSF**(*Q*, *args*)

Computes structure factor for precipitates in a matrix Refs.: E-Wen Huang, Peter K. Liaw, Lionel Porcar, Yun Liu, Yee-Lang Liu, Ji-Jung Kai, and Wei-Ren Chen,APPLIED PHYSICS LETTERS 93, 161904 (2008) R. Giordano, A. Grasso, and J. Teixeira, Phys. Rev. A 43, 6894 1991

Parameters

- **Q** (*float*) – Q value array (A-1)
- **args** (*array*) – [float R, float VolFr]: “radius” & volume fraction

Returns numpy array
S(Q)

GSASII.GSASIIsasd.**LSWCume**(*x*, *pos*, *args=[]*)

Lifshitz-Slyozov-Wagner Ostwald ripening cumulative distribution - numpy friendly on x axis

Parameters

- **x** (*float*) – independent axis (can be numpy array)
- **pos** (*float*) – location of distribution
- **scale** (*float*) – not used
- **shape** (*float*) – not used

Returns float
LSW cumulative distribution

GSASII.GSASIIsasd.**LSWDist**(*x*, *pos*, *args=[]*)

Lifshitz-Slyozov-Wagner Ostwald ripening distribution - numpy friendly on x axis ref:

Parameters

- **x** (*float*) – independent axis (can be numpy array)
- **pos** (*float*) – location of distribution
- **scale** (*float*) – not used
- **shape** (*float*) – not used

Returns float
LSW distribution

GSASII.GSASIIsasd.**LogNormalCume**(*x*, *pos*, *args*)

Standard LogNormal cumulative distribution - numpy friendly on x axis ref: http://www.itl.nist.gov/div898/handbook/index.htm 1.3.6.6.9

Parameters

- **x** (*float*) – independent axis (can be numpy array)
- **pos** (*float*) – location of distribution
- **scale** (*float*) – width of distribution (sigma)
- **shape** (*float*) – shape parameter

Returns float
LogNormal cumulative distribution

GSASII.GSASIIsasd.**LogNormalDist**(*x*, *pos*, *args*)

Standard LogNormal distribution - numpy friendly on x axis ref: http://www.itl.nist.gov/div898/handbook/index.htm 1.3.6.6.9

Parameters

- **x** (*float*) – independent axis (can be numpy array)
- **pos** (*float*) – location of distribution
- **scale** (*float*) – width of distribution (m)
- **shape** (*float*) – shape - (sigma of log(LogNormal))

Returns float
LogNormal distribution

exception GSASII.GSASIIsasd.**MaxEntException**

Any exception from this module

GSASII.GSASIIsasd.**MaxEnt_SB**(*datum*, *sigma*, *G*, *base*, *IterMax*, *image_to_data=None*, *data_to_image=None*, *report=False*)

do the complete Maximum Entropy algorithm of Skilling and Bryan

Parameters

- **datum[]** (*float*)
- **sigma[]** (*float*)
- **G** (*float[][]*) – transformation matrix
- **base[]** (*float*)
- **IterMax** (*int*)
- **image_to_data** (*obj*) – opus function (defaults to opus)
- **data_to_image** (*obj*) – tropus function (defaults to tropus)

Returns float[]

$f(r)dr$

GSASII.GSASIIsasd.**SchulzZimmCume**(*x*, *pos*, *args*)

Schulz-Zimm cumulative distribution - numpy friendly on x axis

Parameters

- **x** (*float*) – independent axis (can be numpy array)
- **pos** (*float*) – location of distribution
- **scale** (*float*) – width of distribution (sigma)
- **shape** (*float*) – not used

Returns float

Normal distribution

GSASII.GSASIIsasd.**SchulzZimmDist**(*x*, *pos*, *args*)

Schulz-Zimm macromolecule distribution - numpy friendly on x axis ref: http://goldbook.iupac.org/S05502.html

Parameters

- **x** (*float*) – independent axis (can be numpy array)
- **pos** (*float*) – location of distribution
- **scale** (*float*) – width of distribution (sigma)
- **shape** (*float*) – not used

Returns float

Schulz-Zimm distribution

GSASII.GSASIIsasd.**SphereFF**(*Q*, *R*, *args=()*)

Compute hard sphere form factor - can use numpy arrays

Parameters

- **Q** (*float*) – Q value array (usually in A-1)
- **R** (*float*) – sphere radius (Usually in A - must match Q-1 units)

- **args** (*array*) – ignored

Returns
form factors as array as needed (float)

GSASII.GSASIIsasd.**SphereVol**(*R*, *args=()*)

Compute volume of sphere - numpy array friendly

Parameters

- **R** (*float*) – sphere radius
- **args** (*array*) – ignored

Returns float
volume

GSASII.GSASIIsasd.**SphericalShellFF**(*Q*, *R*, *args=()*)

Compute spherical shell form factor - can use numpy arrays

Parameters

- **Q** (*float*) – Q value array (usually in A-1)
- **R** (*float*) – sphere radius (Usually in A - must match Q-1 units)
- **args** (*array*) – [float r]: controls the shell thickness: R_inner = min(r*R,R), R_outer = max(r*R,R)

Returns float
form factors as array as needed

Contributed by: L.A. Avakyan, Southern Federal University, Russia

GSASII.GSASIIsasd.**SphericalShellVol**(*R*, *args*)

Compute volume of spherical shell - numpy array friendly

Parameters

- **R** (*float*) – sphere radius
- **args** (*array*) – [float r]: controls shell thickness, see SphericalShellFF description

Returns float
volume

GSASII.GSASIIsasd.**SpheroidFF**(*Q*, *R*, *args*)

Compute form factor of cylindrically symmetric ellipsoid (spheroid) - can use numpy arrays for R & AR; will return corresponding numpy array

Parameters

- **Q** (*float*) – Q value array (usually in A-1)
- **R** (*float*) – radius along 2 axes of spheroid
- **args** (*array*) – [float AR]: aspect ratio so 3rd axis = R*AR

Returns float
form factors as array as needed

GSASII.GSASIIsasd.**SpheroidVol**(*R*, *args*)

Compute volume of cylindrically symmetric ellipsoid (spheroid) - numpy array friendly

Parameters

- **R** (*float*) – radius along 2 axes of spheroid
- **args** (*array*) – [float AR]: aspect ratio so radius of 3rd axis = R*AR

Returns float
volume

GSASII.GSASIIsasd.**SquareWellSF**(*Q*, *args*)

Computes structure factor for not dilute monodisperse hard sphere with a square well potential interaction. Refs.: SHARMA,SHARMA, PHYSICA 89A,(1977),213-

Parameters

- **Q** (*float*) – Q value array (A-1)
- **args** (*array*) – [float R, float VolFrac, float depth, float width]: interparticle distance, volume fraction (<0.08), well depth (e/kT<1.5kT), well width

Returns
numpy array: S(Q) well depth > 0 attractive & values outside above limits nonphysical cf. Monte Carlo simulations

GSASII.GSASIIsasd.**StickyHardSpheresSF**(*Q*, *args*)

Computes structure factor for not dilute monodisperse hard spheres Refs.: PERCUS,YEVICK PHYS. REV. 110 1 (1958),THIELE J. CHEM PHYS. 39 474 (1968), WERTHEIM PHYS. REV. LETT. 47 1462 (1981)

Parameters

- **Q** (*float*) – Q value array (A-1)
- **args** (*array*) – [float R, float VolFrac]: sphere radius & volume fraction

Returns numpy array
S(Q)

GSASII.GSASIIsasd.**UniDiskFF**(*Q*, *R*, *args*)

Compute form factor for unified disk - can use numpy arrays

Parameters

- **Q** (*float*) – Q value array (A-1)
- **R** (*float*) – cylinder radius (A)
- **args** (*array*) – [float T]: disk thickness (A)

Returns float
form factor

GSASII.GSASIIsasd.**UniDiskVol**(*R*, *args*)

Compute disk volume for radius & thickness - numpy array friendly

Parameters

- **R** (*float*) – diameter (A)
- **args** (*array*) – [float T]: thickness

Returns float
volume (A^3)

GSASII.GSASIIsasd.**UniRodARFF**(*Q*, *R*, *args*)

Compute form factor for unified rod of fixed aspect ratio - can use numpy arrays

Parameters

- **Q** (*float*) – Q value array (A-1)
- **R** (*float*) – cylinder radius (A)
- **args** (*array*) – [float AR]: cylinder aspect ratio = L/D = L/2R

Returns float
form factor

GSASII.GSASIIsasd.**UniRodARVol**(*R*, *args*)

Compute rod volume for radius & aspect ratio - numpy array friendly

Parameters

- **R** (*float*) – diameter (A)
- **args** (*array*) – [float AR]: =L/D=L/2R aspect ratio

Returns float
volume (A^3)

GSASII.GSASIIsasd.**UniRodFF**(*Q*, *R*, *args*)

Compute form factor for unified rod - can use numpy arrays

Parameters

- **Q** (*float*) – Q value array (A-1)
- **R** (*float*) – cylinder radius (A)
- **args** (*array*) – [float R]: cylinder radius (A)

Returns float
form factor

GSASII.GSASIIsasd.**UniRodVol**(*R*, *args*)

Compute cylinder volume for radius & length - numpy array friendly

Parameters

- **R** (*float*) – diameter (A)
- **args** (*array*) – [float L]: length (A)

Returns float
volume (A^3)

GSASII.GSASIIsasd.**UniSphereFF**(*Q*, *R*, *args=0*)

Compute form factor for unified sphere - can use numpy arrays

Parameters

- **Q** (*float*) – Q value array (A-1)
- **R** (*float*) – cylinder radius (A)
- **args** (*array*) – ignored

Returns float
form factor

GSASII.GSASIIsasd.**UniSphereVol**(*R*, *args=()*)

Compute volume of sphere - numpy array friendly

Parameters

- **R** (*float*) – sphere radius

- **args** (*array*) – ignored

Returns float
volume

GSASII.GSASIIsasd.**UniTubeFF**(*Q*, *R*, *args*)

Compute form factor for unified tube - can use numpy arrays assumes that core of tube is same as the matrix/solvent so contrast is from tube wall vs matrix

Parameters

- **Q** (*float*) – Q value array (A-1)
- **R** (*float*) – cylinder radius (A)
- **args** (*array*) – [float L,T]: tube length & wall thickness(A)

Returns float
form factor

GSASII.GSASIIsasd.**UniTubeVol**(*R*, *args*)

Compute tube volume for radius, length & wall thickness - numpy array friendly

Parameters

- **R** (*float*) – diameter (A)
- **args** (*array*) – [float L,T]: tube length & wall thickness(A)

Returns float
volume (A^3) of tube wall

GSASII.GSASIIsasd.**print_arr**(*text*, *a*)

print the contents of an array to the console

GSASII.GSASIIsasd.**print_vec**(*text*, *a*)

print the contents of a vector to the console

17.2 *Substances: Define Materials*

Defines materials commonly found in small angle & reflectometry experiments. GSASII substances as a dictionary "Substances.Substances" with these materials.

Each entry in "Substances" consists of:

```
'key':{'Elements':{element:{'Num':float number in formula},...},'Density':value,
↪'Volume':,value}
```

Density & Volume are optional, if one missing it is calculated from the other; if both are missing then Volume is estimated from composition & assuming 10 AA^3 for each atom. Density is calculated from that Volume. See examples below for what is needed.

CHAPTER

EIGHTEEN

GSASIISCRIPTABLE: SCRIPTING INTERFACE

18.1 *Summary/Contents*

GSASIIscriptable provides routines to use an increasing amount of GSAS-II's capabilities from Python 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, peak fits, refinements… The .gpx files are completely compatible with the GUI, so one can move back and forth between the GUI and scripting when developing scripts. This mode of code development is encouraged to get started with GSAS-II scripting.

GSASIIscriptable is normally used by writing Python commands via this module's application programming interface (API). (There is also an older mechanism where GSASIIscriptable can be accessed via shell/batch commands, see *GSASIIscriptable Command-line Interface*, called command-line mode.) Access to GSASIIscriptable via the API is used more widely than via command-line mode and offers many more features. The material below introduces and summarizes use of GSASIIscriptable via the API. Following that, detailed descriptions of all routines are provided 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.

GSASIIscriptable is designed around the hierarchical structure of .gpx files, that is seen in the GUI as the GSAS-II data tree. The module defines wrapper classes (inheriting from `G2ObjectWrapper`) for most GSAS-II data tree items, so most scripting is done with object-oriented code that operate on different types of data tree objects. At the top level one has a project (`G2Project`) which contains phases (`G2Phase`) powder diffraction histograms (`G2PwdrData`).

Scripting Documentation Contents

18.2 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 will use scripting extensively will still need access to the GUI for some activities, since the scripting API does not cover all features of GSAS-II. Even if that were to be 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 project.

Nonetheless, there may be times where it does make sense to install GSAS-II without all of the Python packages needed for running the GUI, for example on a compute server or cluster. The minimal requirements for use of GSASIIscriptable are:

- python
- numpy
- scipy

GSASIIscriptable can be used without these packages, but with significantly reduced functionality, so their inclusion is highly recommended:

- PyCifRW
- requests

There are a few other packages that may be used in GSASIIscriptable, for example to import specific types of powder or image data or perform specific types of computations. They are not commonly needed, but if access is attempted, it should be obvious from error messages:

- h5py
- xmltodict
- pybaselines
- seekpath
- matplotlib
- pillow

More information on Python packages used in GSAS-II is provided in the *Scripting Requirements* section of the Requirements chapter, which also provides some installation instructions.

18.3 Accessing the GSASIIscriptable Module

When GSAS-II is installed with GSAS2MAIN or GSAS2PKG, or with the `gitstrap.py` script, or directly with git, the GSAS-II software is installed outside of Python. When the GUI is invoked, a small script or Windows batch file is used to invoke Python with a reference to a file (named `G2.py`) that starts the GSAS-II GUI. It is also possible to install GSAS-II inside Python (for example with pixi), but work on this is not yet complete or documented.

When GSASIIscriptable is used, and GSAS-II has been installed outside of Python, some mechanism is needed to provide 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 the Python installation.

The latter method requires an extra installation step, but has the advantage that it allows writing portable GSAS-II scripts.

18.3.1 Explicit GSASIIscriptable Specification

MacOS/Linux:

The example below is for MacOS/Linux and assumes that GSAS-II has been installed in location `/Users/toby/g2main/` such that there is a directory `/Users/toby/g2main/GSAS-II/GSASII` that contains all the GSAS-II Python files such as `GSASIIscriptable.py`. Place code like this into the beginning of your script so that the location of GSAS-II files can be found:

```
import sys
sys.path.insert(0,'/Users/toby/g2main/GSAS-II') # needed to find GSASII␣
↪package
from GSASII import GSASIIscriptable as G2sc
```

Windows:

A similar example, but for Windows, assumes that GSAS-II has been installed in location `C:\Users\toby\gsas2main` such that there is a directory `C:\Users\toby\gsas2main\GSAS-II\GSASII` that contains all the GSAS-II Python files such as `GSASIIscriptable.py`. Place code like this into the beginning of your script (note that use of forward slashes is deliberate; to use back slashes, they must be doubled or placed in a raw-string, `'C:\\Users\\...'` or `r'C:\Users\...'`):

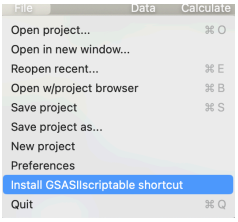
```
import sys
sys.path.insert(0,'/Users/toby/gsas2main/GSAS-II') # needed to find GSASII␣
↪package
from GSASII import GSASIIscriptable as G2sc
```

Note that the directory that is placed in the path is the one that contains the `GSASII` directory. Previously, this path contained this directory, but now is its parent.

18.3.2 Install GSASIIscriptable Location Into Python

As an alternative to defining the location of GSAS-II in every script, you can define the location of GSAS-II inside Python *once*, but note that this must be done for each version of Python, if you plan to use GSAS-II scripting with more that one. If you have more than one version of GSAS-II installed, only one can be defined for a Python installation, but the previous method, where `sys.path` is modified, can be used with all of the GSAS-II installions.

There are three different ways to use GSAS-II to define a location for GSASIIscriptable. You can choose the method that is easiest for you.



1. The most easy option is to invoke the "Install GSASIIscriptable shortcut" command in the GSAS-II GUI File menu.

   This performs the same actions as below, but since the location of both Python and the GSAS-II files are defined within the GUI, no additional input is needed.

2. Alternatively, using the commands modeled after the ones above, add the Python command `G2sc.installScriptingShortcut()` into your script as:

```
import sys
sys.path.insert(0,'/Users/toby/gsas2main/GSAS-II') # needed to find␣
↪GSASII package
from GSASII import GSASIIscriptable as G2sc
G2sc.installScriptingShortcut()
```

   This only needs to be done once. After the script has been run, remove the command or comment it out.

3. The third choice is to run two command-line (bash/zsh/DOS,…) commands. This assumes that the intended Python interpreter is already in the path. (If not use a conda activate command.) Note that the directory that is used is the parent of the `GSASII` directory (the directory that contains `GSASII`.)

   Here are the commands on **MacOS/Linux**:

```
% cd /Users/toby/gsas2main/GSAS-II
% python -c "import GSASII.GSASIIscriptable as G2sc; G2sc.
↪installScriptingShortcut()"
```

   On **Windows** the commands in a cmd.exe window will be similar:

```
>cd \Users\toby\gsas2main\GSAS-II
>python -c "import GSASII.GSASIIscriptable as G2sc; G2sc.
↪installScriptingShortcut()"
```

   The output from this process on Windows is shown below.

```
C:\Users\toby>gsas2main\Scripts\activate

(base) C:\Users\toby>cd gsas2main\GSAS-II

(base) C:\Users\toby\gsas2main\GSAS-II>python -c "import GSASII.GSASIIscriptable as G2sc; G2sc.installScriptingShortcut()"
3 values read from C:\Users\toby\.GSASII\config.ini
GSAS-II binary directory: C:\Users\toby\gsas2main\GSAS-II\GSASII-bin\win_64_p3.13_n2.2
Created file C:\Users\toby\gsas2main\Lib\site-packages\G2script.py
setting up GSASIIscriptable from C:\Users\toby\gsas2main\GSAS-II\GSASII
success creating C:\Users\toby\gsas2main\Lib\site-packages\G2script.py
```

Once any of the above three choices has been completed, a good test to see if GSASIIscriptable is working will be commands:

```
import G2script as G2sc
print(G2sc.ShowVersions())
```

as is shown in the image below.

```
(base) C:\Users\toby\gsas2main\GSAS-II>python
Python 3.13.3 | packaged by conda-forge | (main, Apr 14 2025, 20:31:24) [MSC v.1943 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import G2script as G2sc
setting up GSASIIscriptable from C:\Users\toby\gsas2main\GSAS-II\GSASII
3 values read from C:\Users\toby\.GSASII\config.ini
GSAS-II binary directory: C:\Users\toby\gsas2main\GSAS-II\GSASII-bin\win_64_p3.13_n2.2
>>> print(G2sc.ShowVersions())
  Python      3.13.3:  from C:\Users\toby\gsas2main\python.exe
  numpy       2.2.5:
  scipy       1.15.2:
  GSAS-II:    6c26ef37, 25-Apr-2025 15:26 (0.0 days old). Tag: #5806, v5.3.3

GSAS-II location: C:\Users\toby\gsas2main\GSAS-II\GSASII
Binary location:  C:\Users\toby\gsas2main\GSAS-II\GSASII-bin\win_64_p3.13_n2.2 (5805, v5.2.0)

>>>
```

18.3.3 When GSAS-II is Installed Inside Python

If GSAS-II is installed inside of Python, the location of the GSAS-II software is established without any changes to the path so this command should work without using any of the above:

```
from GSASII import GSASIIscriptable as G2sc
```

18.4 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.

18.4.1 Overview of Classes

Scripting class name	Description
G2Project	`G2Project`: A GSAS-II project file; provides references to objects below, each corresponding to a tree item (exception is `G2AtomRecord`)
G2Phase	`G2Phase`: Provides phase information access (also provides access to atom info via `G2AtomRecord`)
G2AtomRecord	`G2AtomRecord`: Access to an atom within a phase
G2PwdrData	`G2PwdrData`: Access to powder histogram info
G2Single	`G2Single`: Access to single crystal histogram info
G2Image	`G2Image`: Access to image info
G2PDF	`G2PDF`: PDF histogram info
G2SmallAngle	`G2SmallAngle`: Small Angle scattering histogram info
G2SeqRefRes	`G2SeqRefRes`: The sequential results table

18.4.2 Independent Functions

A small number of Scriptable routines do not require existence of a G2Project object.

method	Use
`ShowVersions()`	Shows Python and GSAS-II version information
`GenerateReflections()`	Generates a list of unique powder reflections
`SetPrintLevel()`	Sets the amount of output generated when running a script
`installScriptingShortcut()`	Installs GSASIIscriptable within Python as G2script

18.4.3 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 table below is not complete but does contain the most commonly used methods in this object:

method	Use
`save()`	Writes the current project to disk.
`add_powder_histogram()`	Used to read in powder diffraction data into a project file.

continues on next page

Table 1 – continued from previous page

method	Use
add_simulated_powder_histogram()	Defines a "dummy" powder diffraction data that will be simulated after a refinement step.
add_image()	Reads in an image into a project.
add_phase()	Adds a phase to a project
add_PDF()	Adds a PDF entry to a project (does not compute it)
add_single_histogram()	Used to read in a single crystal diffraction dataset into a project file.
add_SmallAngle()	Adds a small-angle scattering histogram to a project
histograms()	Provides a list of histograms in the current project, as *G2PwdrData* or as *G2Single* objects.
histogram()	Finds a histogram from an object, name or random id reference, returning a a *G2PwdrData* or *G2Single* object.
histType()	Determines the histogram type from an object, name or random id reference.
phases()	Provides a list of phases defined in the current project, as *G2Phase* objects
phase()	Finds a phase from an object, name or random id reference, returning a a *G2Phase* object.
images()	Provides a list of images in the current project, as *G2Image* objects
image()	Finds an image from an object, name or random id reference, returning a a *G2Image* object.
pdfs()	Provides a list of PDFs in the current project, as *G2PDF* objects
seqref()	Returns a *G2SeqRefRes* object if there are Sequential Refinement results
do_refinements()	This is passed a list of dictionaries, where each dict defines a refinement step. Passing a list with a single empty dict initiates a refinement with the current parameters and flags. A refinement dict sets up a single refinement step (as described in *Project-level Parameter Dict*). Also see *Refinement recipe*.
set_refinement()	This is passed a single dict which is used to set parameters and flags. These actions can be performed also in *do_refinements()*.
get_Variable()	Retrieves the value and esd for a parameter
get_Covariance()	Retrieves values and covariance for a set of refined parameters
set_Controls()	Set overall GSAS-II control settings such as number of cycles and to set parameter limits. This is also used to set up a sequential fit. (Also see *get_Controls()* to read values.)
get_LastFitResults()	Retrieves the shifts and sigma values from the last least-squares cycle
imageMultiDistCalib()	Performs a global calibration fit with images at multiple distance settings.
get_Constraints()	Retrieves *constraint definition* entries.
add_HoldConstr()	Adds a hold constraint on one or more variables
add_EquivConstr()	Adds an equivalence constraint on two or more variables

continues on next page

Table 1 – continued from previous page

method	Use
add_EqnConstr()	Adds an equation-type constraint on two or more variables
add_NewVarConstr()	Adds an new variable as a constraint on two or more variables
ComputeWorstFit()	Determines the parameters that will have the greatest impact on the fit if refined
get_Frozen()	Find variables where parameters have refined out of the parameter limit ranges. Note that parameter limits are set using *set_Controls()*.
set_Frozen()	Adds or removes variables from the list where parameters have refined outside of their limits. Note that parameter limits are set using *set_Controls()*.

18.4.4 Class `G2Phase`

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

method	Use
set_refinements()	Provides a mechanism to set values and refinement flags for the phase. See *Phase parameters* for more details. This information also can be supplied within a call to *do_refinements()* or *set_refinement()*.
clear_refinements()	Unsets refinement flags for the phase.
set_HAP_refinements()	Provides a mechanism to set values and refinement flags for parameters specific to both this phase and one of its histograms. See *Histogram-and-phase parameters*. This information also can be supplied within a call to *do_refinements()* or *set_refinement()*.
clear_HAP_refinements()	Clears refinement flags specific to both this phase and one of its histograms.
getHAPvalues()	Returns values of parameters specific to both this phase and one of its histograms.
copyHAPvalues()	Copies HAP settings between from one phase/histogram and to other histograms in same phase.
HAPvalue()	Sets or retrieves values for some of the parameters specific to both this phase and one or more of its histograms.
atoms()	Returns a list of atoms in the phase
atom()	Returns an atom from its label
add_atom()	Adds an atom to a phase
histograms()	Returns a list of histograms linked to the phase
get_cell()	Returns unit cell parameters (also see *get_cell_and_esd()*)
export_CIF()	Writes a CIF for the phase
setSampleProfile()	Sets sample broadening parameters
clearDistRestraint()	Clears any previously defined bond distance restraint(s) for the selected phase
addDistRestraint()	Finds and defines new bond distance restraint(s) for the selected phase
setDistRestraintWeight()	Sets the weighting factor for the bond distance restraints
Origin1to2Shift()	Shifts the atom coordinates from an Origin 1 setting to the Origin 2 setting

18.4.5 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
set_refinements()	Provides a mechanism to set values and refinement flags for the powder histogram. See *Histogram parameters* for details.
clear_refinements()	Unsets refinement flags for the powder histogram.
residuals()	Reports R-factors etc. for the powder histogram (also see *get_wR()*)
add_back_peak()	Adds a background peak to the histogram. Also see *del_back_peak()* and *ref_back_peak()*.
fit_fixed_points()	Fits background to the specified fixed points.
set_background()	Sets a background histogram that will be subtracted (point by point) from the current histogram.
calc_autobkg()	Estimates the background and sets the fixed background points from that.
getdata()	Provides access to the diffraction data associated with the histogram.
reflections()	Provides access to the reflection lists for the histogram.
Export()	Writes the diffraction data or reflection list into a file
add_peak()	Adds a peak to the peak list. Also see *Peak Fitting*.
set_peakFlags()	Sets refinement flags for peaks
refine_peaks()	Starts a peak/background fitting cycle, returns refinement results
Peaks	Provides access to the peak list data structure
PeakList	Provides the peak list parameter values
Export_peaks()	Writes the peak parameters to a text file
Limits()	Reads or sets the region of data used in fitting (histogram limits)
Excluded()	Reads or sets regions of powder data that will be ignored
ComputeMassFracs()	Reports mass (weight) fractions and their uncertainties

18.4.6 Class `G2Single`

A less commonly-used object in GSASIIscriptable scripts is *G2Single*, which will encapsulate each single crystal diffraction histogram in a project. At present, very few methods are provided:

method	Use
set_refinements()	Provides a mechanism to set refinement flags for the single crystal histogram. See *Histogram parameters* for details.
clear_refinements()	Unsets refinement flags for the single crystal powder histogram.
Export()	Writes the reflection list into a file

18.4.7 Class `G2Image`

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

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

18.4.8 Class G2PDF

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

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

18.4.9 Class G2SmallAngle

To work with Small Angle (currently only SASD entries), object *G2SmallAngle*, encapsulates a SASD entry. At present no methods are provided.

18.4.10 Class G2SeqRefRes

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

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

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

18.5 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.

18.5.1 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': 'chebyschev-1', 'refine': True,
                                  'peaks':[[0,True],[1,1,1]] }},
          'clear': {'Instrument Parameters': ['U', 'V', 'W']}}
my_project.set_refinement(pardict)
```

18.5.2 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': 'chebyschev-1', 'refine': True}},
          'clear': {'Instrument Parameters': ['U', 'V', 'W']}}
my_project.do_refinements([pardict])
```

However, in addition to setting a number of parameters, this example will perform a refinement as well, after setting the parameters. More than one refinement can be performed by including more than one dict in the list.

In this example, two refinement steps will be performed:

```
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 *Specifying Refinement Parameters* 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 is 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 *Histogram parameters* or *Histogram-and-phase parameters*. Note that this will be ignored for *Phase parameters*. 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 *Phase parameters* or *Histogram-and-phase parameters*. Note that this will be ignored for *Histogram parameters*. 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 `iter_refinements()` where `self` 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 <tt>G2Project</tt> 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.

18.5.3 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*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.

18.6 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.

18.6.1 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 (Debye-Scherrer)
	DisplaceY	Sample displacement along the Y direction (Debye-Scherrer)
	Shift	Bragg-Brentano sample displacement
	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 `ref_back_peak()` instead. When value is a dict, supply any of the following keys:
	type	The background model, e.g. 'chebyschev-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

18.6.2 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.

18.6.3 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
PhaseFraction		PhaseFraction can also be used in place of Scale for the routines that access HAP parameters: *HAPvalue()*, *setHAPvalues()*, *copyHAPvalues()*, *set_refinement()*, *do_refinements()*, *clear_HAP_refinements()* and *set_HAP_refinements()*.

18.6.4 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': 'chebyschev-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).

18.7 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:

```
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 Ban
k 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 Ban
k 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
↪'])
---
> , '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.

18.8 Code Examples

Contents for Scripting Examples

18.8.1 Accessing GSASIIscriptable

As discussed in the *Accessing the GSASIIscriptable Module* section of this chapter, GSAS-II is commonly installed outside of a Python installation, which means that Python must be instructed on how to access the GSASII package. In *that section* three methods are provided for defining G2sc as the location of the GSASIIscriptable module. The scripting examples below all assume that one of choices for import statements has been executed to provide access to GSASIIscriptable.

18.8.2 Status Information

To find information on Python, Python packages and the GSAS-II version, one can call the `ShowVersions()` function. This will show versions and install locations.

```
print(f'Version information:\n{G2sc.ShowVersions()}')
```

which produces output like this:

```
setting up GSASIIscriptable from /Users/toby/G2/git/g2full/GSAS-II/GSASII
Version information:
  Python      3.11.9:  from /Users/toby/py/mf3/envs/py311/bin/python
  numpy       1.26.4:
  scipy       1.13.0:
  IPython     8.22.2:
  GSAS-II:    641a65, 24-May-2024 10:16 (0.5 days old). Last tag: #5789

GSAS-II location: /Users/toby/G2/git/g2full/GSAS-II/GSASII
Binary location:  /Users/toby/G2/git/g2full/GSAS-II/GSASII-bin/mac_arm_p3.11_n1.26
```

18.8.3 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/).

```
import os
datadir = os.path.expanduser("~/Scratch/peakfit")
PathWrap = lambda fil: os.path.join(datadir,fil)
gpx = G2sc.G2Project(newgpx=PathWrap('pkfit.gpx'))
hist = gpx.add_powder_histogram(PathWrap('FAP.XRA'), PathWrap('INST_XRY.PRM'),
                                fmthint='GSAS powder')
hist.set_refinements({'Limits': [16.,24.],
      'Background': {"no. coeffs": 2,'type': 'chebyschev-1', 'refine': True}
                     })
peak1 = hist.add_peak(1, ttheta=16.8)
peak2 = hist.add_peak(1, ttheta=18.9)
peak3 = hist.add_peak(1, ttheta=21.8)
peak4 = hist.add_peak(1, ttheta=22.9)
hist.set_peakFlags(area=True)
hist.refine_peaks()
hist.set_peakFlags(area=True,pos=True)
hist.refine_peaks()
hist.set_peakFlags(area=True, pos=True, sig=True, gam=True)
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
```

18.8.4 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
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
```

(continues on next page)

(continued from previous page)

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

18.8.5 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.

```
datadir = "/Users/Scratch/"
gpx = G2sc.G2Project(os.path.join(datadir,'test2.gpx'))
gpx.histogram(0).add_back_peak(4.5,30000,5000,0)
pardict = {'set': {'Sample Parameters': ['Absorption', 'Contrast', 'DisplaceX'],
                   'Background': {'type': 'chebyschev-1', 'refine': True,
                                  'peaks':[[0,True]]}}}
gpx.set_refinement(pardict)
```

18.8.6 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,glob
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},
```

(continues on next page)

(continued from previous page)

```
     "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'])
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()
```

18.8.7 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
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'))
pwdrList = 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,glob
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)
```

(continues on next page)

(continued from previous page)

```
    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'])
```

18.8.8 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,glob
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
```

(continues on next page)

(continued from previous page)

```
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()
```

18.8.9 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.

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
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.imctrl')
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['intTAmap'] = 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[
```

(continues on next page)

(continued from previous page)

```
↪'FrameMask'])
            for pwdr in img.Integrate(MaskMap=cache['intMaskMap'],ThetaAzimMap=cache[
↪'intTAmap']):
                pwdr.Export(os.path.join(pathxye,os.path.split(tifname)[1]),'.xye')
↪# '.tif in name ignored
            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
```

18.8.10 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 preceded 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 hyper-threading. 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')

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
```

(continues on next page)

(continued from previous page)

```
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
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['intTAmap'] = 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 pwdr in img.Integrate(MaskMap=cache['intMaskMap'],ThetaAzimMap=cache[
↪'intTAmap']):
            pwdr.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)
```

(continues on next page)

(continued from previous page)

```
    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
            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')
```

18.8.11 Access the Image Pixel-Mask

In this example, a pixel mask has already been computed and has been saved with the image in a .gpx file. This example then reads the .gpx file, locates an image and then pulls the spot mask (an array of True and False values for every pixel) from the data structure. As an extra check (and demo) the image is reread and the dimensions of the image are confirmed to match those of the image. Note that the *GeneratePixelMask()* routine could also have been used to compute the mask.

This also provides an example showing how a result that is not made directly available from the GSASIIscriptable API can still be accessed from the GSAS-II data structures, but this requires some care to determine where values are stored.

```
import os
import G2script as G2sc
datadir = os.path.expanduser("~/Scratch/MPE_H5")
PathWrap = lambda fil: os.path.join(datadir,fil)
gpx = G2sc.G2Project(PathWrap('pixelMask.gpx'))
img0 = gpx.image(0) # access 1st image
spotMask = img0.data['Masks']['SpotMask'].get('spotMask')
if spotMask is not None:
    assert spotMask.shape == img0.getImage().shape   # diagnostic to confirm sizes␣
↪match
```

18.8.12 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,glob
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')
```

18.8.13 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 Chebyschev 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 Chebyschev 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 Chebyschev polynomial terms is increased to six and the refinement is repeated.

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

18.8.14 Specify Instrument Parameters Directly

Rather than read instrument parameters from a file, it is also possible to specify them directly in a script. See the documentation on instrument parameter file contents, *CW Instrument Parameters* and *TOF Instrument Parameters* for more information on the parameters supplied here.

```
import G2script as G2sc
import os
```

(continues on next page)

(continued from previous page)

```
datadir = os.path.expanduser("~/Scratch/peakfit")
PathWrap = lambda fil: os.path.join(datadir,fil)
gpx = G2sc.G2Project(newgpx=PathWrap('pkfit.gpx'))
# specify instrmental parameters dictionaries
inst_params = [
    {
        "Type": ["PXC", "PXC", 0],
        "Lam": [1.5405, 1.5405, 0],
        "Zero": [0.0, 0.0, 0],
        "Polariz.": [0.7, 0.7, 0],
        "U": [2.0, 2.0, 0],
        "V": [-2.0, -2.0, 0],
        "W": [5.0, 5.0, 0],
        "X": [0.0, 0.0, 0],
        "Y": [0.0, 0.0, 0],
        "Z": [0.0, 0.0, 0],
        "SH/L": [0.002, 0.002, 0],
        "Azimuth": [0.0, 0.0, 0],
        "Bank": [1, 1, 0],
    },
    {},
]
hist = gpx.add_powder_histogram(PathWrap('FAP.XRA'), fmthint='GSAS powder',
                                iparams=inst_params)
```

18.9 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.

18.9.1 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.

18.10 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 GSASII.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 'I' 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 GSASII.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)
>>> pwdrList = 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*, *ttmin=0.0*, *ttmax=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. The image must be properly calibrated so that radial averaging is possible. (See GSASIIimage.AutoPixelMask() and scipy.stats.median_abs_deviation() for more details.)

The 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 retained if the .gpx file is to 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 *MaskFrameMask()*). 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 implementatruion 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* or *G2SmallAngle*) 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 calbration information to be deleted

getImage()

Returns the image, even if not already cached

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.

loadPixelMask(*mask*, *tag='loaded in G2sc.loadPixelMask'*)

Loads a pixel map from an array supplied by the user

Parameters

- **mask** (*np.array*) – An array that has a True or False value for each pixel. True means that the pixel should be masked. The array dimensions must match the current image.
- **tag** (*str*) – provides a name that is saved to indicate the source of the mask. At present this name is not used.

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 GSASII.GSASIIscriptable.**G2ImportException**

class GSASII.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 GSASII.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 specifed together.

optimize (*showFit=True*, *maxCycles=5*, *xydata=None*, *limits=None*, *inst=None*)

Optimize the low R portion of G(R) to minimize selected parameters. Note that this updates the parameters in the settings (self.data['PDF Controls']) but does not update the PDF object arrays (self.data['PDF Controls']['G(R)'] etc.) with the computed values, use *calculate()* after a fit to do that.

Parameters

- **showFit** (*bool*) – if True (default) the optimized parameters are shown before and after the fit, as well as the RMS value in the minimized region.
- **maxCycles** (*int*) – the maximum number of least-squares cycles; defaults to 5.
- **xydata** (*dict*) – an array containing the Sample's I vs Q, and any or none of the Sample Background, the Container scattering and the Container Background. If xydata is None (default), the values are taken from histograms, as named in the PDF's self.data['PDF Controls'] entries with keys 'Sample', 'Sample Bkg.','Container Bkg.' & 'Container'.
- **limits** (*list*) – upper and lower Q values to be used for PDF computation. If None (default), the values are taken from the Sample histogram's .data['Limits'][1] values.
- **inst** (*dict*) – The Sample histogram's instrument parameters to be used for PDF computation. If None (default), the values are taken from the Sample histogram's .data['Instrument Parameters'][0] values.

Returns

the result from the optimizer as True or False, depending on if the refinement converged.

set_background (*btype*, *histogram*, *mult=-1.0*, *refine=False*)

Sets a histogram to be used as the 'Sample Background', the 'Container' or the 'Container Background.'

Parameters

- **btype** (*str*) – Type of background to set, must contain the string 'samp' for Sample Background', 'cont' and 'back' for the 'Container Background' or only 'cont' for the 'Container'. Note that capitalization and extra characters are ignored, so the full strings (such as 'Sample Background' & 'Container Background') can be used.
- **histogram** – A reference to a histogram, which can be reference by object, name, or number.
- **mult** (*float*) – a multiplier for the histogram; defaults to -1.0
- **refine** (*bool*) – a flag to enable refinement (only implemented for 'Sample Background'); defaults to False

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 GSASII.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*, *newValue=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' or 'PhaseFraction' (either can be used for phase fraction), 'Use', 'Extinction', 'LeBail', 'PO' (for Preferred Orientation). If not specified or invalid an exception is generated showing the list of valid parameters. At present, only these HAP parameters cannot be accessed with this function: 'Size', 'Mustrain', 'HStrain', 'Babinet'. 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. When param='PO' then this value is interpreted as the following:

  if the value is 0 or an even integer, then the preferred orientation model is set to "Spherical harmonics". If the value is 1, then "March-Dollase" is used. Any other value generates an error.

- **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 (i.e. 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('PhaseFraction',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.

InitDisAgl(*useAll=True*)

Create the default controls used for distance and angle searching. Perform distance and angle searching by passing the results from this to `GSASIIstrMain.RetDistAngle()` or `GSASIIstrMain.PrintDistAngle()` At present, this does not populate the values needed for uncertainty computations.

Parameters

useAll (*bool*) – when True (default) all atoms are included in the origin atom list. When False, only atoms with full occupancy are included. All atoms are included in the target atom list.

Returns

DisAglCtls, DisAglData. See the *Distance/Angle controls documentation* for a description of these.

Origin1to2Shift()

Applies an Origin 1 to Origin 2 shift to the selected phase, if defined. Note that GSAS-II only uses Origin 2 settings when both are offered in the International Tables. (These are space groups where the centre of symmetry is not at the highest symmetry site in the cell.) If the structure is not in the Origin 1 setting, this routine will create garbage.

A copy of the phase is made where the new phase name has the string "_shifted" added to it. The routine returns a reference to the new *G2Phase* object for the new phase.

If the phase is not one of the space groups that has Origin 1 & Origin 2 settings, None is returned.

Returns

the newly created phase object or None

ShortDistances(*useAll=False*)

Looks for unrealistic distances in a structure, which are atom-atom distances < 1.1 A for non-H(D) atoms. To reduce the likelihood of distances between disordered fragments being noted, set useAll=False (the default) so that disordered atoms are ignored.

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 A 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 A. 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 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()*. Don't use with `use`.
- **use** (*list*) – specifies the items in the HAP dict should be copied. The default is None, which causes all items to be copied. Don't use with `skip`.

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]
>>> 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 A. 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])
```

(continues on next page)

(continued from previous page)

```
>>> 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()*. Don't use with `use`.
- **use** (*list*) – specifies the items in the HAP dict should be copied. The default is None, which causes all items to be copied. Don't use with `skip`.

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

Example for Size and Mustrain with LG_mix:

```
phase.set_HAP_refinements({
    'Size': {'type':'isotropic', 'refine': True,
             'LGmix': {'value': 0.5, 'refine': False}},
    'Mustrain': {'type':'uniaxial',
                 'LGmix': {'value': 0.8, 'refine': True}}
})
```

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 GSASII.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 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(Chi**2)/d(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.

SAS (*sasRef*)

Gives an object representing the specified SAS entry in this project.

Parameters

sasRef – A reference to the desired SASD entry. Either the SASD tree name (str), the SASD's index (int) or a SASD object (*G2SmallAngle*)

Returns

A *G2SmallAngle* object

Raises

KeyError

See also

SASs() G2PDF

SASs ()

Returns a list of all the Small Angle histograms in the project.

Returns

A list of *G2SmallAngle* objects

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

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.

- **override** (*bool*) – This routine looks up variables using `GSASIIobj.getDescr()` (which is not comprehensive). If not found, the routine will throw an exception, unless override=True is specified.

Example:

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

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

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.
- **override** (*bool*) – This routine looks up variables using `GSASIIobj.getDescr()` (which is not comprehensive). If not found, the routine will throw an exception, unless override=True is specified.

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*, *override=False*)

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.
- **override** (*bool*) – This routine looks up variables using `GSASIIobj.getDescr()` (which is not comprehensive). If not found, the routine will throw an exception, unless override=True is specified.

Example:

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

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

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.
- **override** (*bool*) – This routine looks up variables using `GSASIIobj.getDescr()` (which is not comprehensive). If not found, the routine will throw an exception, unless override=True is specified.

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(*prmfile*, *histogram*)

Creates a PDF entry that can be used to compute a PDF. Note that this command places an entry in the project, but *G2PDF.calculate()* must be used to actually perform the computation.

Parameters

- **datafile** (*str*) – The powder data file to read, a filename.
- **histogram** – A reference to a histogram, which can be reference by object, name, or number.

Returns

A *G2PDF* object for the PDF entry

add_SmallAngle(*datafile*)

Placeholder for an eventual routine that will read a small angle dataset from a file.

Parameters

datafile (*str*) – The SASD data file to read, a filename.

Returns

A *G2SmallAngle* object for the SASD 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*, *URL=False*, *download_loc=None*, *imageKey=None*)

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. Note that with this option, all images are read from the file, but only the specified image(s) are retained. Do not use imageKey and indexList together.
- **cacheImage** (*bool*) – When True, the image is cached to save in rereading it later. Default is False (no caching).
- **URL** (*bool*) – if True, the contents of imagefile is a URL and the file will be downloaded and saved. The file will be written in the specified directory (see *download_loc*) or a temporary location, if not specified. Note that if a temporary location, if the proiject (.gpx) file is saved, the image may not be accessible if the .gpx file is later reopened. Default is False. If URL is specified and the Python requests package is not installed, a *ModuleNotFoundError* Exception will occur. will occur.
- **download_loc** (*str*) – a location or file name where the image will be saved. Note that for almost all image types, the image cannot be read if the file extension does not match what is expected for the format. (This can be determined by looking at the importer code; if *strictExtension=True*, the extension must be in the *extensionlist* list.) If only a directory is specified, the file name will be taken from the URL, which will likely cause problems if it does not match the needed extension. If URL is specified and the default download_loc value is used (None), the image will be saved in a temporary location that will persist until the OS removes it.
- **imageKey** – This can be a single image number (int) to read a specific image (numbered starting with 1) or for files that have images in named sections, (right now this is only HDF5), it can be a tuple of form (‘section’,0) where ‘section’ is the section name (such as ‘/exchange/data’) and 0 is the image number in that section. If imageKey is specified, only one image is read. Do not use imageKey and indexList together.

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*, *URL=False*, *useNet=False*, *PhaseRef=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). Specifying this is optional but is strongly encouraged.
- **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).
- **URL** (*bool*) – if True, the contents of phasefile is a URL and the file will be downloaded to a temporary location and read. The downloaded file will not be saved. If URL is specified and the Python requests package is not installed, a *ModuleNotFoundError* Exception will occur. will occur.
- **useNet** (*bool*) – if True, when an incompatible space group setting is detected (at present this is only tested with CIFs, where symmetry operators are supplied), which is most likely to occur with origin-1 settings, where allowed, the importer will call Bilbao "CIF to Standard Setting" web service. (Default is False).
- **PhaseRef** – Used for magnetic phases only, a reference to the (aka chemical/nuclear) phase. This can be the phase name (str), the phase's ranId, the phase's index (both int) or a phase object (*G2Phase*)

Returns
A *G2Phase* object representing the new phase.

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

- **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.
- **URL** (*bool*) – if True, the contents of datafile is a URL and if not a dict, the contents of iparams is also a URL. both files will be downloaded to a temporary location and read. The downloaded files will not be saved. If URL is specified and the Python requests package is not installed, a *ModuleNotFoundError* Exception will occur. will occur.

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 specifiying 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, where parameters are frozen after refining outside the range where their values are allowed due to parameter limits, when these limits are set. (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_LastFitResults()

Returns the shifts on refined variables and their uncertainties in the last refinement cycle

Returns

a dict with the last least-squares shifts and a dict of sigma 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.

Returns
A list of *G2Phase* objects

See also

histogram() histograms() phase()

refine(*newfile=None*, *printFile=None*, *makeBack=False*)

Invoke a refinement for the project. The project is written to the currently selected gpx file and then either a single or sequential refinement is performed depending on the setting of 'Seq Data' in Controls (set in *get_Controls()*).

reload()

Reload self from self.filename

save(*filename=None*)

Saves the project, either to the current filename, or to a new file.

Updates self.filename if a new filename provided

seqref()

Returns a sequential refinement results object, if present

Returns

A *G2SeqRefRes* object or None if not present

set_Controls(*control*, *value*, *variable=None*)

Set project controls.

Controls determine how refinements are performed, including setting lower (parmMin) or upper limits (parmMax) values for parameters where you choose to set refinement limits. Note that use of this with to 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 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 minimum or maximum 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), where parameters are frozen after refining outside the range where their values are allowed due to parameter limits, when these limits are set. (See *Parameter Limits* description and *G2Project.set_Controls()* for setting limits.) 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 refinment 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 GSASII.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.

Note that the returned list is a reference to the actual list as stored in the .gpx file; use care not to modify this unless intended.

Returns

list containing a list and dict with background values

ComputeMassFracs()

Computes the mass fractions (or equivalently the weight fractions) for the phases linked to the current histogram with uncertainties from the results of the last refinement, if the phase fractions were refined.

Returns

a dict where the keys are phase names and the values associated with is a tuple where the first value is the phase's mass fraction and the second value is the s.u. on that value.

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 nuber of points used may differ by +-1 from Npoints. Must be below 25,000.

Excluded(*value=None*)

Used to obtain or set the excluded regions for a histogram. When a value is specified, the excluded regions are set. Otherwise, the list of excluded region pairs is returned. Note that excluded regions may be an empty list or a list of regions to be excluded, where each region is provided as pair of numbers, where the lower limit comes first. Some sample excluded region lists are:

```
[[4.5, 5.5], [8.0, 9.0]]

[[130000.0, 140000.0], [160000.0, 170000.0]]

[]
```

The first above describes two excluded regions from 4.5-5.5 and 8-9 degrees 2-theta. The second is for a TOF pattern and also describes two excluded regions, for 130-140 and 160-170 milliseconds. The third line would be the case where there are no excluded regions.

Parameters

value (*list*) – A list of pairs of excluded region numbers (as two-element lists). Some error checking/reformatting is done, but users are expected to get this right. Use the GUI to create examples or check input. Numbers in the list are in units of degrees or TOF (microsec.).

If a value is not specified, the command returns the list of excluded regions.

Returns
The list of excluded regions (when `value=None`). Units are 2-theta (degrees) or TOF (microsec).

Example 1:

```
h = gpx.histogram(0)  # adds an excluded region (11-13 degrees)
h.Excluded(h.Excluded() + [[11,13]])
```

Example 2:

```
h = gpx.histogram(0) # changes the range of the first excluded region
excl = h.Excluded()
excl[0] = [120000.0, 160000.0]  # microsec
h.Excluded(excl)
```

Example 3:

```
h = gpx.histogram(0)  # deletes all excluded regions
h.Excluded([])
```

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.

Note that the returned dict is a reference to the actual dict as stored in the .gpx file; use care not to modify this unless intended.

Limits (*typ*, *value=None*)

Used to obtain or set the histogram limits. When a value is specified, the appropriate limit is set. Otherwise, the value is returned. Note that this provides an alternative to setting histogram limits with the `G2Project:do_refinements()` or *G2PwdrData.set_refinements()* methods.

Parameters

- **typ** (*str*) – a string which must be either 'lower' (for 2-theta min or TOF min) or 'upper' (for 2theta max or TOF max). Anything else produces an error.
- **value** (*float*) – the number to set the limit (in units of degrees or TOF (microsec.). If not specified, the command returns the selected limit value rather than setting it.

Returns

The current value of the requested limit (when `value=None`). Units are 2-theta (degrees) or TOF (microsec).

Examples:

```
h = gpx.histogram(0)
val = h.Limits('lower')
h.Limits('upper',75)
```

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.

Note that the returned dict is a reference to the actual dict as stored in the .gpx file; use care not to modify this unless intended.

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

Background in diffraction patterns is usually fit with a slowly varying smooth function, such as a Chebyschev polynomial, but when the background contains broad peaks (for example from a Kapton sample container) those peaks are usually better fit by adding extra peaks to the smooth background function rather that providing enough parameters to the smooth function in order fit the peak(s). Note that background peaks are typically treated as Gaussian only (`gam``=0) with very large ``sig` values (>1000). Normally one should refine `int` and then `sig` and only after the background peak is well fit can one refine the `pos` value.

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 not used (small)
- **refflags** (*list*) – a list of 1 to 4 boolean refinement flags for pos,int,sig & gam, respectively (e.g. 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

Parameters

- **area** (*float*) – peak area
- **dspace** (*float*) – peak position as d-space (A)
- **Q** (*float*) – peak position as Q (A-1)
- **ttheta** (*float*) – peak position as 2Theta (deg)

Note: only one of the parameters: dspace, Q or ttheta may be specified. See *Peak Fitting* for an example.

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*) – log_10 of the Lambda value used in the pybaselines.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.

It should be noted that for TOF data, GSAS-II expects input where the TOF value is the minimum for the bin, but does computations using the TOF value for the center of each bin. The values reported using the 'X' option here are the values used in computation, while the 'X-orig' option reverses the shift applied when TOF values are read in.

Parameters
datatype (*str*) – must be one of the following values (case is ignored):

- 'X': the 2theta or TOF values for the pattern
- **'X-orig': for TOF, time values for the pattern shifted**
  as used as input for GSAS-II. For everything else, the values are the same as with the 'X' option (see above.)
- 'Q': the 2theta or TOF values for the pattern transformed to Q
- 'd': the 2theta or TOF values for the pattern transformed to d-space
- '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
a numpy MaskedArray with data values of the requested type. Note that the returned values are a copy of the GSAS-II histogram array, not a reference to the actual data as stored in the .gpx file.

ref_back_peak (*peaknum*, *refflags=[]*)

Sets refinement flag for a background peak

Parameters

- **peaknum** (*int*) – the number of the peak (starting from 0)
- **refflags** (*list*) – a list of 1 to 4 boolean refinement flags for pos,int,sig & gam, respectively. If a flag is not specified it defaults to False (use [0,1] to refine int only). Defaults to [] which means nothing is refined.

refine_peaks (*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).

Note that the returned array is a reference to the actual data as stored in the .gpx file; use care not to modify this.

property residuals

Provides a dictionary with with the R-factors for this histogram. Includes the weighted and unweighted profile terms (R, Rb, wR, wRb, wRmin) as well as the Bragg R-values for each phase (ph:H:Rf and ph:H:Rf^2).

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, str, None 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()*.

Note that the returned array is a reference to the actual data as stored in the .gpx file; use care not to modify this.

exception GSASII.GSASIIscriptable.**G2ScriptException**

class GSASII.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:

```
import os
PathWrap = lambda fil: os.path.join('/Users/toby/Scratch/SeqTut2019Mar',fil)
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 in the project tree starting from 0.

Returns

cell,cellESD,uniqCellIndx where cell (list) with the unit cell parameters (a,b,c,alpha,beta,gamma,Volume); cellESD are the standard uncertainties on the 7 unit cell parameters; and uniqCellIndx is a tuple with indicies for the unique (non-symmetry determined) unit parameters (e.g. [0,2] for a,c in a tetragonal cell)

histograms()

returns a list of histograms in the squential fit

class GSASII.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()

Export(*fileroot*, *extension*, *fmthint=None*)

Write the HKLF 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['single'] 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

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})
```

class GSASII.GSASIIscriptable.**G2SmallAngle**(*data*, *proj*, *name*)

Wrapper for SASD histograms (and hopefully, in the future, other small angle histogram types).

Note that in a GSASIIscriptable script, instances of G2SmallAngle will be created by calls to *SAS()*, *SASs()*, or by `G2Project.Integrate()`. Also, someday `G2Project.add_SAS()`. Scripts should not try to create a *G2SmallAngle* object directly.

This object contains these class variables:

- G2SmallAngle.proj: contains a reference to the *G2Project* object that contains this histogram
- G2SmallAngle.name: contains the name of the histogram
- G2SmallAngle.data: contains the histogram's associated data in a dict with keys 'Comments', 'Limits', 'Instrument Parameters', 'Substances', 'Sample Parameters' and 'Models'. Further documentation on SASD entries needs to be written.

See also

`add_SAS()` *SAS()* *SASs()* `Integrate()`

GSASII.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:

```
>>> refs = G2sc.GenerateReflections('P 1',
...                     (5.,6.,7.,90.,90.,90),
...                     TTmax=20,wave=1)
>>> for r in refs: print(r)
...
[0, 0, 1, 7.0]
[0, 1, 0, 6.0]
[1, 0, 0, 5.0]
[0, 1, 1, 4.55553961419178]
[0, 1, -1, 4.55553961419178]
[1, 0, 1, 4.068667356033675]
[1, 0, -1, 4.068667356033674]
[1, 1, 0, 3.8411063979868794]
[1, -1, 0, 3.8411063979868794]
```

GSASII.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
```

GSASII.GSASIIscriptable.**LoadDictFromProjFile**(*ProjFile*)

Read a GSAS-II project file and load items to dictionary

Parameters

ProjFile (*str*) – GSAS-II project (name.gpx) full file name

Returns

Project,nameList, where

- Project (dict) is a representation of gpx file following the GSAS-II tree structure for each item: key = tree name (e.g. 'Controls','Restraints',etc.), data is dict data dict = {'data':item data whch may be list, dict or None,'subitems':subdata (if any)}
- nameList (list) has names of main tree entries & subentries used to reconstruct project file

Example for fap.gpx:

```
Project = {                 #NB:dict order is not tree order
  'Phases':{'data':None,'fap':{phase dict}},
  'PWDR FAP.XRA Bank 1':{'data':[histogram data list],'Comments':comments,'Limits
↪':limits, etc},
  'Rigid bodies':{'data': {rigid body dict}},
  'Covariance':{'data':{covariance data dict}},
  'Controls':{'data':{controls data dict}},
  'Notebook':{'data':[notebook list]},
  'Restraints':{'data':{restraint data dict}},
  'Constraints':{'data':{constraint data dict}}]
  }
nameList = [                #NB: reproduces tree order
  ['Notebook',],
  ['Controls',],
  ['Covariance',],
  ['Constraints',],
  ['Restraints',],
  ['Rigid bodies',],
  ['PWDR FAP.XRA Bank 1',
       'Comments',
       'Limits',
       'Background',
       'Instrument Parameters',
       'Sample Parameters',
       'Peak List',
       'Index Peak List',
       'Unit Cells List',
       'Reflection Lists'],
  ['Phases', 'fap']
  ]
```

GSASII.GSASIIscriptable.**LoadG2fil**()

Setup GSAS-II importers. Delay importing this module when possible, it is slow. Multiple calls are not. Only the first does anything.

GSASII.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)

GSASII.GSASIIscriptable.**Readers = {'Image': [], 'Phase': [], 'Pwdr': [], 'importErrpkgs': []}**

Readers by reader type

GSASII.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)

GSASII.GSASIIscriptable.**SetDebugMode**(*mode*)

Set the debug configuration mode on (mode=True) or off (mode=False). This will provide some additional output that may help with tracking down problems in the code.

GSASII.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'

GSASII.GSASIIscriptable.**SetupGeneral**(*data*, *dirname*)

Initialize phase data.

GSASII.GSASIIscriptable.**ShowVersions**()

Show the versions all of required Python packages, etc.

GSASII.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.
```

(continues on next page)

(continued from previous page)

```
                        Example: -pf CIF
-l HISTLIST [HISTLIST ...], --histlist HISTLIST [HISTLIST ...]
                        list of histgram indices to associate with added
                        phases. If not specified, phases are associated with
                        all previously loaded histograms. Example: -l 2 3 4
```

GSASII.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 histogram3d)

GSASII.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()*.

GSASII.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()*.

GSASII.GSASIIscriptable.**create**(*args*)

Implements the create command-line subcommand. This creates a GSAS-II project, optionally adding histograms and/or phases:

```
usage: GSASIIscriptable.py create [-h] [-d HISTOGRAMS [HISTOGRAMS ...]]
                                  [-i IPARAMS [IPARAMS ...]]
                                  [-p PHASES [PHASES ...]]
                                  filename
```

positional arguments:

```
filename              the project file to create. should end in .gpx
```

optional arguments:

```
-h, --help            show this help message and exit
-d HISTOGRAMS [HISTOGRAMS ...], --histograms HISTOGRAMS [HISTOGRAMS ...]
                      list of datafiles to add as histograms
-i IPARAMS [IPARAMS ...], --iparams IPARAMS [IPARAMS ...]
                      instrument parameter file, must be one for every
                      histogram
-p PHASES [PHASES ...], --phases PHASES [PHASES ...]
                      list of phases to add. phases are automatically
                      associated with all histograms given.
```

GSASII.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 constains 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)
```

GSASII.GSASIIscriptable.**downloadFile**(*URL*, *download_loc=None*)

Download the URL

GSASII.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
```

GSASII.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
```

GSASII.GSASIIscriptable.**exportersByExtension = {}**

Specifies the list of extensions that are supported for Powder data export

GSASII.GSASIIscriptable.**import_generic**(*filename*, *readerlist*, *fmthint=None*, *bank=None*, *URL=False*, *download_loc=None*, *useNet=True*, *buffer=None*, *imageKey=None*)

Attempt to import a filename, using a list of reader objects.

This is not intended to be called directly in scripting, only by routines like G2Project.add_phase() and G2Project.add_image but this may be used to read CIFs, as is done with OnISODISTORT_kvec in GSASIIpwdGUI.

Returns the first reader object which is read successfully.

GSASII.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 *Accessing the GSASIIscriptable Module*.

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.

GSASII.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

GSASII.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)

GSASII.GSASIIscriptable.**main**()

The command-line interface for calling GSASIIscriptable as a shell command, where it is expected to be called as:

```
python GSASIIscriptable.py <subcommand> <file.gpx> <options>
```

The following subcommands are defined:

- create, see *create()*
- add, see *add()*
- dump, see *dump()*
- refine, see *refine()*
- export, *export()*
- browse, see *IPyBrowse()*

See also

create() add() dump() refine() export() IPyBrowse()

GSASII.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)

GSASII.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
```

CHAPTER

NINETEEN

GSAS-II MISC SCRIPTS

19.1 testDeriv: Check derivative computation

Use this to check derivatives used in structure least squares refinement against numerical values computed in this script. To use set `DEBUG=True` in GSASIIstrMain.py (line 40, as of version 2546); run the least squares - zero cycles is sufficient. Do the "Save Results"; this will write the file testDeriv.dat in the local directory.

Then run this program to see plots of derivatives for all parameters refined in the last least squares. Shown will be numerical derivatives generated over all observations (including penalty terms) and the corresponding analytical ones produced in the least squares. They should match. Profiling is also done for function calculation & for the 1st selected derivative (rest should be the same).

`GSASII.testDeriv.`**`main`**`()`

Starts main application to compute and plot derivatives

class `GSASII.testDeriv.`**`testDeriv`**(*parent*)

19.2 GSASIItestplot: Plotting for testDeriv

Plotting module used for script testDeriv.

class `GSASII.GSASIItestplot.`**`Plot`**(*parent*, *id=-1*, *dpi=None*, ***kwargs*)

Creates a plotting window

class `GSASII.GSASIItestplot.`**`PlotNotebook`**(*id=-1*)

creates a Wx application and a plotting notebook

`Show`(*show=True*) → bool

Shows or hides the window.

19.3 scanCCD: reduce data from scanning CCD

Quickly prototyped routine for reduction of data from detector described in B.H. Toby, T.J. Madden, M.R. Suchomel, J.D. Baldwin, and R.B. Von Dreele, "A Scanning CCD Detector for Powder Diffraction Measurements". Journal of Applied Crystallography. 46(4): p. 1058-63 (2013). This is no longer being updated.

`GSASII.scanCCD.`**`main`**`()`

starts main application to merge data from scanning CCD

class `GSASII.scanCCD.`**`scanCCD`**(*parent*)

PlotXY(*XY*, *newPlot=False*, *type=''*)

simple plot of xy data, used for diagnostic purposes

class GSASII.scanCCD.**scanCCDmain**(*redirect=False*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)

OnInit(*self*) → bool

19.4 *makeMacApp: Create MacOS Applet*

This script creates an AppleScript app bundle to launch GSAS-II. It is called by bootstrap.py during the GSAS-II installation process. It creates a "copy" of Python that is able to run wx.Python programs and names this version of Python as GSAS-II so that items in the menus are named correctly. This routine creates an app bundle named GSAS-II.app. Inside the bundle is a symbolic link to the Python executable named "GSAS-II" that will be used to run GSAS-II. Having this link named that way causes the name of the app to shows in the menu bar as "GSAS-II" rather than "Python". Also used by the app, is another symbolic link named GSAS-II.py, which must be placed in the same directory as the app bundle. This file is linked to the G2.py script and the link is run using the link to Python. This also causes other items in the app to be labeled as GSAS-II (but not with the right capitalization, alas).

The original contents of the app bundle was created interactively and, after some manual edits, the contents of that was placed into a tar file distributed with GSAS-II, and is expanded in this script. This method seems to be needed for MacOS 11.0+ (Big Sur and later) where Apple's security constraints seem to prevent creation of the app directly. Older code (not currently in use) created the app from "scratch" using the osacompile utility, but that no longer seems to work.

Three different paths are needed to run this script:

```
path2GSAS:  The location where the GSAS-II Python files are found.
installLoc: The location where the GSAS-II.app app bundle and
   the GSAS-II.py will be placed. This will be the parent of path2GSAS
pythonLoc:  The location of the Python executable.
```

Under normal circumstances, the locations for all of these paths can be determined from the location of the makeMacApp.py file. Note that when GSAS-II is installed from git using gitstrap.py, the git repository is placed at <loc>/GSAS-II and the GSAS-II Python scripts are placed in the <loc>/GSAS-II/GSASII child directory. GSAS-II is started from the <loc>/GSAS-II/GSAS-II.py script created here and the current script (makeMacApp.py) will be found in <loc>/GSAS-II/GSASII/install/.

When the GSAS-II conda installers are used, the git repository is placed at $CONDA_HOME/GSAS-II so that <loc> above is $CONDA_HOME. Also, the Python executable will be found in $CONDA_HOME/bin/Python. Thus, if this file is in makePath (typically <loc>/GSAS-II/GSASII/install), then

- path2GSAS will be makePath/.. and
- installLoc will be path2GSAS/.. and
- pythonLoc will be installLoc/../bin/python,

but these locations can be overridden from the command-line arguments. If a Python location is not supplied and is not at the default location (installLoc/../bin/python) then the Python executable currently running this script (from sys.executable) is used.

Run this script with no arguments or with one or two arguments.

The first argument, if supplied, provides the path to be used for the app bundle will be created. Note that GSAS-II.app and GSAS-II.py will be created in this directory.

The second argument, if supplied, is path2GSAS, a path to the location G2.py script, which can be a relative path (the absolute path is determined). If not supplied, the G2.py script is expected to be located in the directory above where this (makeMacApp.py) script is found.

The third argument, if supplied, provides the full path for the Python installation to be used inside the app bundle that will be created. If not supplied, and Python exists at installLoc/../bin/python, that will be used. If that does not exist, then the location of the current Python executable (from sys.executable) will be used.

`GSASII.install.makeMacApp.`**`AppleScript = ''`**

Will be set to contain an AppleScript to start GSAS-II by launching Python and the GSAS-II Python script. Not currently used.

19.5 *makeBat: Create GSAS-II Batch File*

This script performs Windows specific installation steps to allow for easy launching of GSAS-II. It is called by bootstrap.py during the GSAS-II installation process. This script creates a file named `RunGSASII.bat` and a desktop shortcut to that file. It registers the filetype .gpx so that the GSAS-II project files exhibit the GSAS-II icon and so that double-clicking on them opens them in GSAS-II.

Run this script with no arguments; the path to the `G2.py` file is assumed to be in the parent directory to the one where this file (`makeBat.py`) is found.

The contents of this file may also be run from inside the gitstrap.py installation script. In that case, the following variables are already defined:

- path2GSAS2 is the directory with all GSAS-II Python code
- G2script has the location of the G2.py file
- path2repo is the location of the GSAS-II git repository

The path to Python is determined from the version of Python used to run this script.

19.6 *makeLinux: Create Linux Shortcuts*

This script performs Linux specific installation steps that allowscreates files allowing GSAS-II to be launched from a desktop icon or desktop manager menu. Not all desktop managers will recognize these files. It is called by bootstrap.py during the GSAS-II installation process. This script creates a menu entry and dektop shortcut for Gnome (and perhaps KDE) desktop managers. The most recent testing has been on Raspberry Pi OS. My hope is to improve this further to work conveniently with a wider range of Linux desktop managers.

Run this script with one optional argument, the location of the G2.py file. That location may be specified relative to the current path or given an absolute path, but will be accessed via an absolute path. If no arguments are supplied, the path to the `G2.py` file is assumed to be in the parent directory to the one where this file (`makeLinux.py`) is found.

The contents of this file may also be run from inside the gitstrap.py installation script. In that case, the following variables are already defined:

- path2GSAS2 is the directory with all GSAS-II Python code
- G2script has the location of the G2.py file
- path2repo is the location of the GSAS-II git repository

The path to Python is determined from the version of Python used to run this script.

19.7 *makeVarTbl: Make Table of Variable Names*

This creates a table of variable names from the definitions supplied in `GSASIIobj.CompileVarDesc()`. This table is used in the Sphinx documentation as the *GSAS-II Variable Names table*. This is run as part of the Sphinx build from inside `docs/source/conf.py`.

19.8 *testSytSym: Test Site Symmetry*

A GUI program for testing the site symmetry generation routines.

GSASII.testSytSym.**main**()

Starts main application to compute and plot derivatives

class GSASII.testSytSym.**testSytSmain**(*redirect=False*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)

OnInit(*self*) → bool

class GSASII.testSytSym.**testSytSym**(*parent*)

19.9 *testSSymbols: Test Superspace Group Symbols*

A GUI program for testing the 3+1 superspace group symmetry generation routines.

GSASII.testSSymbols.**main**()

Starts main application to compute and plot derivatives

class GSASII.testSSymbols.**testSSmain**(*redirect=False*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)

OnInit(*self*) → bool

class GSASII.testSSymbols.**testSSymbols**(*parent*)

19.10 *Self-test Modules*

A set of scripts that can be run to test a series of self-tests in GSAS-II.

19.10.1 test_diffax.py

Performs a DIFFaX computation using the GSAS-II interface to the pydiffax.so [.pyd] module

tests.test_diffax.**test_diffax**()

tests DIFFaX

19.10.2 test_elm.py

Performs a very simple test on the element tables.

tests.test_elm.**test_get_xsection**()

Test that a cross-section for a single element is read

19.10.3 test_image.py

Performs tests that image integration and pixel masking are working. Also tests the imports that require GSAS-II-compiled binaries.

tests.test_image.**test_CBF**()

Read in a CBF image (tests unpack_cbf module)

tests.test_image.**test_image**()

Read in a MAR image (tests pack_f binary module) and then integrate it using GeneratePixelMask (tests fmask binary module)

19.10.4 test_kvec.py

Performs a series of tests on the k-vector search code.

tests.test_kvec.**test_AtomID**()

self-test #0: test the unique ID generation routine

tests.test_kvec.**test_CriticalRoutines**()

self-test #2: test the critical routines

tests.test_kvec.**test_KVecCandidateUpdate**()

self-test #3: test the updating of the list of alternative k vectors

tests.test_kvec.**test_KVecSearch**()

self-test #4: test the k vector search routine

tests.test_kvec.**test_LatConstruct**()

self-test #1: test the lattice vectors construction routine

19.10.5 test_lattice.py

Performs comprehensive unit tests for many but by no means all routines in *GSASIIlattice*.

tests.test_lattice.**selftestlist = [<function test_gmat>, <function test_Avec>, <function test_2cell>, <function test_invcell>, <function test_V>, <function test_2AB>, <function test_Brav>, <function test_Laue>]**

Defines a list of self-tests

tests.test_lattice.**test_2AB**()

test cell2AB

tests.test_lattice.**test_2cell**()

test Gmat2A, A2cell, A2Gmat, Gmat2cell

tests.test_lattice.**test_Avec**()

test cell2A and A2Gmat

tests.test_lattice.**test_Brav**()

test GetBraviasNum() and GenHBravais()

tests.test_lattice.**test_Laue**()

test GenHLaue

tests.test_lattice.**test_V**()

test calc_rVsq, calc_rV, calc_V

tests.test_lattice.**test_gmat**()

tests cell2Gmat, fillgmat, Gmat2cell

tests.test_lattice.**test_invcell**()

test invcell2Gmat

19.10.6 test_nistlat.py

Tests the two NIST*LATTICE binaries, LATTIC and convcell using their Python wrappers in module nistlat.py

tests.test_nistlat.**test_CellSymSearch**()

test_CellSymSearch(): nistlat.CellSymSearch

tests.test_nistlat.**test_CompareCell**()

test_CompareCell(): nistlat.CompareCell

tests.test_nistlat.**test_ConvCell**()

test_ConvCell(): nistlat.ConvCell

tests.test_nistlat.**test_ReduceCell**()

test_ReduceCell(): nistlat.ReduceCell

19.10.7 test_scriptref.py

Performs a CW GSAS-II refinement using GSASIIscriptable and tutorial data. Tests texture code as well.

tests.test_scriptref.**test_refine**()

Performs a GSAS-II refinement using GSASIIscriptable and tutorial data

19.10.8 test_spg.py

Unit tests for code in GSASIIspc.py. Also exercises the pyspg Fortran routine.

tests.test_spg.**selftestlist = [<function test_MoveToUnitCell>, <function test_SpcGroup1>, <function test_SpcGroup2>, <function test_SytSym>]**

Defines a list of self-tests

tests.test_spg.**test_MoveToUnitCell**()

self-test #0: exercise MoveToUnitCell

tests.test_spg.**test_SpcGroup1**()

self-test #1: SpcGroup against previous results

tests.test_spg.**test_SpcGroup2**()

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

tests.test_spg.**test_SytSym**()

self-test #3: exercise SytSym against selected data from IT Volume A

19.10.9 test_tofref.py

Perform a TOF GSAS-II refinement using GSASIIscriptable and tutorial data. This tests the TOF diffraction code and that Marquardt damping works.

tests.test_tofref.**test_refine**()

Tests a TOF refinement with POWGEN data

19.11 *Other scripts*

A few scripts are also placed in the GSAS-II auxiliary repositories

19.11.1 gitstrap.py

`gitstrap.py` in `GSASII-buildtools/install/` is used to install the GSAS-II package, including the appropriate binary files. May be used directly to install GSAS-II from inside Python in an appropriately configured Python installation, or is also used to obtain or update the GSAS-II files in a conda installation.

19.11.2 gitcompile.py

`gitcompile.py` in `GSASII-buildtools/install/` is used to install the GSAS-II package, but also compiles the binary files. May be used directly to install GSAS-II from inside Python in an appropriately configured Python installation.

19.11.3 makeGitTutorial.py

`makeGitTutorial.py` in `GSASII-tutorials/scripts/` provides a script to creates the HTML page (`GSASII/help/Tutorials.html`) that lists all the tutorials defined in variable `GSASIIctrlGUI.tutorialIndex`. Run this after adding new tutorials to that catalog.

19.11.4 tag-version.py

`tag-version.py` in `GSASII/install/` creates a new numerical tag number (advancing from `5898` to `5899`) for the most recent git check in and records that in the `git_version.py` file. This also advances the minor version number for the GSAS-II version number (from `5.x.y` to `5.(x+1).0`). Use this when there is a significant change to GSAS-II functionality.

19.11.5 incr-mini-version.py

`incr-mini-version.py` in `GSASII/install/` creates a new numerical tag number (advancing from `5898` to `5899`) for the most recent git check in and records that in the `git_version.py` file. This also advances the "mini" version number for the GSAS-II version number (from `5.x.y` to `5.x.(y+1)`). Use this to note a minor by noteworthy change to GSAS-II functionality, such as a bug fix where users should be aware that something has changed.

19.11.6 macStartScript.py

`macStartScript.py` in `GSASII/install/` creates a MacOS applet to start GSAS-II similar to what is in `makeMacApp.py` (where the app is taken from a tar file). This is not in regular use, as it seems to have some permissions problems, but may be needed to update `makeMacApp.py`.

CHAPTER

TWENTY

GSAS-II WEB MODULES

These modules are used to access external web sites.

20.1 SUBGROUPS: Interface Bilbao SUBGROUPS & k-SUBGROUPSMAG web pages

Extraction of space subgroups for a given space group and a propagation vector from the GSAS version of SUBGROUPS & k-SUBGROUPSMAG web page on the Bilbao Crystallographic server. Note that the web pages are special to GSAS-II. This uses the `GSASIIpath.postURL()` function for web access.

20.1.1 Module SUBGROUPS

Routines that access the Bilbao Crystallographic Server.

Note that there is a test for some of these routines in file `tests/run_bilbao.py`.

`GSASII.SUBGROUPS.`**`BCS_init`**(*threadCallback=None*)

Initialize access to the Bilbao Crystallographic Server.

Returns
True if initialization fails due to the lack of a key

`GSASII.SUBGROUPS.`**`BilbaoGetStructure`**(*csdict*, *rowdict*, *stru*, *pagelist=None*, *dlg=None*, *dlgNum=None*, *prefix=''*)

Get the structure for a specified supergroup using the results from BilbaoSymSearch1

`GSASII.SUBGROUPS.`**`BilbaoLowSymSea1`**(*formDict*, *row*, *pagelist=None*)

Using a candidate higher symmetry unit cell from *`BilbaoSymSearch1()`* for monoclinic and triclinic cells, create a list of possible supergroups. Those that match the possible lattice types are marked for potential follow-up to see if coordinates can be are consistent with that symmetry.

Returns

latticeList,valsdict,tbl where

- latticeList: a list of the possible Bravais lattice types
- valsdict: a dict with values needed for the next web form
- **tbl a list of supergroups with four values per entry,**
  - True/False if the lattice type matches,
  - a label with the space group number and the index (sg@ind),
  - the space group number and a lattice type (cell & centering)

GSASII.SUBGROUPS.**BilbaoLowSymSea2**(*num*, *valsdict*, *row*, *savedcookies*, *pagelist=None*)

For a selected cell & supergroup from *BilbaoLowSymSea1()*, see if the coordinates are consistent with the supergroup

GSASII.SUBGROUPS.**BilbaoReSymSearch**(*key*, *result*, *maxdelta=2*, *pagelist=None*)

Perform a supergroup search on a result from previously identified supergroup that was found in *find2SearchAgain()* from the returned web pages. Provides results about the same as from *BilbaoSymSearch1()*

Returns

formDict,csdict,rowdict where formDict will contain values to be used in the next call to Bilbao

csdict will be used to select which entries will be used in the next search and rowdict contains possible supergroup settings.

GSASII.SUBGROUPS.**BilbaoSymSearch1**(*sgnum*, *phase*, *maxdelta=2*, *angtol=None*, *pagelist=None*, *keepCell=False*)

Perform a search for a supergroup consistent with a phase using the Bilbao Pseudosymmetry search (PSEUDO) program, see C. Capillas, E.S. Tasci, G. de la Flor, D. Orobengoa, J.M. Perez-Mato and M.I. Aroyo. "A new computer tool at the Bilbao Crystallographic Server to detect and characterize pseudosymmetry". Z. Krist. (2011), 226(2), 186-196 DOI:10.1524/zkri.2011.1321.

The phase must be in a standard setting.

Parameters

- **sgnum** (*int*) – A space group number (1-230)
- **phase** (*dict*) – a GSAS-II phase object (see *Phase Information*). Note that the phase must be in a standard setting (see *GetStdSGset()*).
- **maxdelta** (*float*) – Allowed distance tolerance in pseudosym search (default 2)
- **angtol** (*float*) – Allowed tolerance for cell angles, used for finding possible unit cells in from triclinic or monoclinic cells, ignored otherwise. Defaults to None, which will cause 5 degrees to be used.
- **pagelist** (*list*) – a list to contain references to the text of web pages created by the Bilbao web site. If None (default) the web pages are not saved.
- **keepCell** (*bool*) – if False (default) and the cell is monoclinic or triclinic, a search is made for higher symmetry cells. If True, the search is made with the current cell.

Returns

formDict,csdict,rowdict,stru where the contents will change depending on the space group, but formDict will contain values to be used in the next call to Bilbao

- For monoclinic and triclinic unit cells: csdict will be None and rowdict (rowlist) will be a list containing unit cells of higher symmetry matching the input unit cell to be used for searching for supergroups.
- For higher symmetry unit cells, csdict will be used to select which entries will be used in the next search and rowdict contain possible supergroup settings.

GSASII.SUBGROUPS.**GetNonStdSubgroups**(*SGData*, *kvec*, *star=False*, *landau=False*, *maximal=False*)

Run Bilbao's SUBGROUPS for a non-standard space group. This requires doing a post to the Bilbao site, which returns all subgroups of the entered space group as the text of a web page with a table containing the space group symbol, the transformation matrix and index for each subgroup.

Params list kvec

propogation vector as a list of nine string fractions or blank

Params SGData
space group object (see *Space Group object*)

Returns
(error,text) error: if True no error or False; where text containts a possible web page text

GSASII.SUBGROUPS.**GetNonStdSubgroupsmag**(*SGData*, *kvec*, *star=False*, *landau=False*, *maximal=False*)

Run Bilbao's k-Subgroupsmag for a non-standard space group. This requires doing a post to the Bilbao site, which returns all magnetic subgroups of the entered subgroup as the text of a web page with a table containing the BNS magnetic space group symbol, the transformation matrix and index for each subgroup.

Params list kvec
propogation vector as a list of three numbers

Params SGData
space group object (see *Space Group object*)

Returns
(error,text) error: if True no error or False; where text containts a possible web page text

GSASII.SUBGROUPS.**GetStdSGset**(*SGData=None*, *oprList=[]*)

Determine the standard setting for a space group from either a list of symmetry operators or a space group object using the Bilbao Crystallographic Server utility IDENTIFY GROUP

Parameters

- **oprList** (*list*) – a list of symmetry operations (example: ['x,y,z', '-x,-y,-z']). Supply either this or SGData, not both.
- **SGData** – from `GSASIIspc.SpcGroup()` Supply either this or oprList, not both.

Returns

(sgnum, sgnam, xformM, offsetV) where:

- sgnum is the space group number,
- sgnam is the space group short H-M name (as used in GSAS-II)
- xformM is 3x3 numpy matrix relating the old cell & coordinates to the new
- offsetV is the vector of offset to be applied to the coordinates

Note that the new cell is given by G2lat.TransformCell([a,b,…],xformM)

GSASII.SUBGROUPS.**RegisterProgressDialog**(*pgbar=None*)

Register a ProgressDialog so it is updated while waiting for the Bilbao server to respond. To reset, when the ProgressDialog is destroyed, call with no argument.

Returns
the result from BCS_init, True if initialization fails due to the lack of a key.

GSASII.SUBGROUPS.**createStdSetting**(*cifFile*, *rd*)

Use the Bilbao "CIF to Standard Setting" web service to obtain a structure in a standard setting. Then update the reader object with the space group, cell and atom positions from this. This is called from the CIF importer in `G2phase_CIF` when a structure is encountered that has different symmetry operators from what GSAS-II generates.

GSASII.SUBGROUPS.**find2SearchAgain**(*pagelist*, *req='@'*)

Scan through web pages from supergroup tests and pull out where coordinates pass the tests to be potentially used to search entries to be used to search for a higher symmetry setting.

GSASII.SUBGROUPS.**scanBilbaoMinsup**(*page0*)

Scan the output from Bilbao minsup search. Obtains a set of cells and symmetry.

Returns

formDict,csdict,rowdict where: * formDict: has the information to be submitted on the next form * csdict: a dict of True/False values with the defaults for continued

searching on each row

- rowdict: a dict with the information for each cell/sym that was found. The contents of each row are list elements:

HM symbol, sgnum, index, index i_k, TR mat, Trans Cell, WP valid, latt_valid

GSASII.SUBGROUPS.**scanBilbaoPseudocell**(*page0*)

Scan the output from Bilbao minsup search. Obtains a set of cells and symmetry.

Returns

formDict,csdict,rowdict where: * formDict: has the information to be submitted on the next form * csdict: is None

searching on each row

- rowdict: a dict with the information for each cell/sym that was found.

20.2 *ISODISTORT: Interface to BYU ISODISTORT web pages*

Uses the BYU ISODISTORT web site to search over all k-points for a structure or to relate a parent and child structure by irreps. This uses the mod:*requests* package.

GSASII.ISODISTORT.**GetISODISTORT**(*Phase*)

Run Stokes & Campbell ISODISTORT. This requires doing a post to the BYU upload site with a cif file, which returns a BYU local copy. This is then sent to the BYU form site with various options, which returns all subgroups of the entered space group as the text of a web page with a table containing the space group symbol, the transformation matrix and index for each subgroup. Selection of one of these is returned to the BYU form site which returns the text of a cif file to be used to create the new phase which can apply the distortion mode constraints

Params dict Phase
GSAS-II phase data

Returns
radio: dict of possible distortion structures

Returns
data2: list of str input for next run of isositortform for extracting cif file

GSASII.ISODISTORT.**GetISODISTORTcif**(*Phase*)

Run Stokes & Campbell ISODISTORT. Selection of one of the order parameter disrections is returned to the BYU form site which returns the text of a cif file to be used to create the new phase which can apply the distortion mode constraints

Params dict Phase
GSAS-II phase data; contains result of GetISODISTORT above & selection

Returns
CIFfile str: name of cif file created by this in local directory

CHAPTER

TWENTYONE

GSAS-II IMPORTER MODULES

Almost all the types of data files that GSAS-II can accept are read using a layer of routines called importers. (The one exception is the reading of powder peak positions.) Importers usually require quite simple code, so they can be written quickly for new data formats. The interface to the importers is self-configuring, so all supplied importers are available once the importer is added to the code base. This allows GSAS-II to be quite flexible in adapting to use many data formats without need for extensive coding.

Imports are implemented by deriving a class from `GSASIIobj.ImportPhase`, `GSASIIobj.ImportStructFactor`, `GSASIIobj.ImportPowderData`, `GSASIIobj.ImportSmallAngleData`, `GSASIIobj.ImportReflectometryData`, `GSASIIobj.ImportPDFData`, or `GSASIIobj.ImportImage`. These classes are in turn derived from `GSASIIobj.ImportBaseclass`. Module file names (*G2phase_*, *G2pwd_* and *G2sfact_*, etc.) are used to determine which type of data will be read and which menu an importer routine should be placed into. (N.B. in retrospect this naming was an unnecessary choice; importer types could have been determined from the base class as is done for exporters.)

To implement the import of a phase, a single crystal or a powder dataset, etc., create a file named with the appropriate file name prefix:

- 'img' (images),
- 'pdf' (pair distribution function),
- 'phase' (cell/coordinates),
- 'pwd' (powder diffraction),
- 'rfd' (reflectivity),
- 'sad' (small-angle scattering) or
- 'sfact' (single crystal).

The importer file can be placed either in the `GSASII/imports` directory (which requires modification of the `__init__.py` and the `meson.build` files or the importer file can be placed in the `~/.GSASII/imports` directory. (Note that `~` here is translated to the user's home directory; for Windows this is usually taken from the USERPROFILE setting or a combination of HOMEPATH and HOMEDRIVE, so this directory will usually have form `C:\\Users\\YourUsername\\.GSASII\\imports`. The next time GSAS-II is started, the file will be loaded with all the other GSAS-II files and the new data format(s) will appear in the appropriate importer menu. The importer file may contain a single importer class or several.

Importers are documented below, separated by type. Importers tend to be fairly simple files, where many are in the range of 50-100 lines, and where more than half of those lines are directly copied from other importers without any changes. Details on this are given in the *Writing a Importer Routine* section, immediately below.

21.1 Writing an Importer Routine

When writing a importer routine, one should create a new class derived from `GSASIIobj.ImportPhase`, `GSASIIobj.ImportStructFactor`, `GSASIIobj.ImportPowderData`, `GSASIIobj.ImportSmallAngleData`, `GSASIIobj.ImportReflectometryData`, `GSASIIobj.ImportPDFData`, or `GSASIIobj.ImportImage`. The name of the class is arbitrary, but if more than one class is placed in file, each class must have a different name. The same name can be repeated if it is in different files. As described below, to implement an importer class, you must implement an `__init__()` and a `Reader()` method, and many will supply a `ContentsValidator()` method, too. The purpose of each of these routines is described below. The easiest way to craft a new importer will be to use the other importers of the same data type as a model for what values should be set inside each routine, with most of the work needed to create a `Reader()` routine. The documentation for the parent class may also have some useful information. See the appropriate class for details on what values each type of `Reader()` should set.

21.1.1 __init__()

The `__init__` method will follow standard boilerplate largely independent of the data type:

```
def __init__(self):
    super(self.__class__,self).__init__( # fancy way to self-reference
        extensionlist=('.ext1','ext2'),
        strictExtension=True,
        formatName = 'example image',
        longFormatName = 'A longer description that this is an example image format'
        )
```

The first line in the `__init__` method calls the parent class `__init__` method with the following parameters:

- `extensionlist`: a list of extensions that may be used for this type of file.
- `strictExtension`: Should be True if only files with extensions in `extensionlist` are allowed; False if all file types should be offered in the file browser. Also if False, the importer class will be used on all files when "guess from format" is tried, though readers with matching extensions will be tried first. It is a very good idea to supply a *ContentsValidator* method when `strictExtension` is False.
- `formatName`: a string to be used in the menu. Should be short.
- `longFormatName`: a longer string to be used to describe the format in help.

Note that if an importer detects a condition which prevents its use, for example because a required Python package is not present, it can set the value of `self.UseReader` to False. Another possible use for this would be an importer that requires a network connection to a remote site. Setting `self.UseReader` to False must be done in the `__init__` method and will prevent the importer from being accessed or included in the appropriate GUI menu.

21.1.2 Reader()

The class must supply a `Reader` method that actually performs the reading. All readers must have at a minimum these arguments:

```
def Reader(self, filename, filepointer, ParentFrame, **unused):
```

where the arguments have the following uses:

- `filename`: a string with the name of the file being read
- `filepointer`: a file object (created by `open()`) that accesses the file and is points to the beginning of the file when Reader is called.

- `ParentFrame`: a reference to the main GSAS-II (tree) windows, for the unusual `Reader` routines that will create GUI windows to ask questions. For use with scripting, the Reader should do something reasonable such as assume a default if `ParentFrame` is None, which indicates that the GUI should not be accessed.

In addition, the following keyword parameters are defined that `Reader` routines may optionally use:

- `buffer`: a dict that can be used to retain information between repeated calls of the routine
- `blocknum`: counts the number of times that a reader is called, to be used with files that contain more than one set of data (e.g. GSAS .gsa/.fxye files with multiple banks or image files with multiple images.)
- `usedRanIdList`: a list of previously used random Id values that can be checked to determine that a value is unique.

Note that a Reader is used to read only a single phase, image, dataset, etc. and will be called repeatedly when used to read files that contain multiple datasets, etc. The `buffer` dict can be used to hold information that will speed repeated calls. As an example, the `buffer` dict is used in CIF reading to hold the parsed CIF file, so that when reading multiple datasets or phases from a multi-block CIF, the parsed information can be reused without having to reread and reparse the file for subsequent calls. For multi-image files, the indexing of where images are to be found is done once and saved. This greatly speeds the time needed to process image file.

Some additional information specific to on what a `Reader()` method should do for images and single-crystal datasets can be found in the documentation for `ImportImage` (images) and `ImportStructFactor`, respectively.

Reader return values

The `Reader` routine should return the value of True if the file has been read successfully. Optionally, use *self.warnings* to indicate any problems.

If the file cannot be read, the `Reader` routine should return False or raise an `GSASIIobj.ImportBaseclass.ImportException()` exception. (Why are both available? Sometimes an exception is the easiest way to bail out of a called routine.) Place text in *self.errors* and/or use:

```
ImportException('Error message')
```

to give the user information on what went wrong during the reading. The following variables are used to indicate results from the reader:

self.warnings

Use *self.warnings* to indicate any information that should be displayed to the user if the file is read successfully, but perhaps not completely or additional settings will need to be made.

self.errors

Use *self.errors* to give the user information on where and why a read error occurs in the file. Note that text supplied with the `raise` statement will be appended to `self.errors`.

self.repeat

Set *self.repeat* to True (the default is False) if a Reader should be called again to after reading to indicate that more data may exist in the file to be read. This is used for reading multiple powder histograms or multiple images from a single file. Variable *self.repeatcount* is used to keep track of the block numbers.

Reader support routines

Note that module *GSASIIctrlGUI* supplies three GUI routines, *BlockSelector()*, *MultipleBlockSelector()*, and *MultipleChoicesDialog()* that are useful for selecting amongst one or more datasets (and perhaps phases) or data items for `Reader()` routines that may encounter more than one set of information in a file.

21.1.3 ContentsValidator()

Defining a `ContentsValidator` method is optional, but is usually a good idea, particularly if the file extension is not a reliable identifier for the file type. The intent of this routine is to take a superficial look at the file to see if it has the expected characteristics of the expected file type. For example, are there numbers in the expected places?

This routine is passed a single argument:

- *filepointer*: a file object [created by Python intrinsic `open()`] that accesses the file and is points to the beginning of the file when ContentsValidator is called.

Note that `CIFValidator()` found in `GSASIIobj.ImportBaseclass` is a ContentsValidator for validating CIF files.

ContentsValidator return values

The `ContentsValidator` routine should return the value of True if the file appears to match the type expected for the class.

If the file cannot be read by this class, the routine should return False. Preferably one will also place text in *self.errors* to give the user information on what went wrong during the reading.

21.1.4 ReInitialize()

Importer classes are substantiated only once and are used as needed. This means that if something needs to be initialized before the `Reader()` will be called to read a new file, the initialization step must be coded. The `ReInitialize()` method is provided for this and it is always called before the `ContentsValidator` method is called. Use care to call the parent class `ReInitialize()` method, if this is overridden.

21.2 Phase Importer Routines

Phase importer routines are classes derived from `GSASIIobj.ImportPhase`. They must be found in files named *G2phase*.py* that are in the Python path and the class must override the `__init__` method and add a `Reader` method. The distributed routines are:

21.2.1 *Module G2phase* (PDB, PDF, .EXP & JANA m40,m50)

A set of short routines to read in phases in a number of formats, as created by a number of external programs. The PDB (Protein Data Bank) format uses coordinates in Angstroms. Other software includes GSAS/EXPGUI .EXP, JANA .m40-.m50 and Powder Diffaction File entries from the ICDD. There are several classes in *G2phase*. The documentation for them follows.

class `GSASII.imports.G2phase.`**EXP_ReaderClass**

Routine to import Phase information from GSAS .EXP files

ContentsValidator(*filename*)

Look for a VERSION tag in 1st line

ReadEXPPhase(*G2frame*, *filepointer*)

Read a phase from a GSAS .EXP file.

Reader(*filename*, *ParentFrame=None*, *usedRanIdList=[]*, ***unused*)

Read a phase from a GSAS .EXP file using *ReadEXPPhase()*

class GSASII.imports.G2phase.**JANA_ReaderClass**

Routine to import Phase information from a JANA2006 file

ContentsValidator(*filename*)

Taking a stab a validating a .m50 file (look for cell & at least one atom)

ReadJANAPhase(*filename*, *parent=None*)

Read a phase from a JANA2006 m50 & m40 files.

Reader(*filename*, *ParentFrame=None*, ***unused*)

Read a m50 file using *ReadJANAPhase()*

class GSASII.imports.G2phase.**PDB_ReaderClass**

Routine to import Phase information from a PDB file

ContentsValidator(*filename*)

Taking a stab a validating a PDB file (look for cell & at least one atom)

ReadPDBPhase(*filename*, *parent=None*)

Read a phase from a PDB file.

Reader(*filename*, *ParentFrame=None*, ***unused*)

Read a PDB file using *ReadPDBPhase()*

class GSASII.imports.G2phase.**PDF_ReaderClass**

Routine to import Phase information from ICDD Powder Diffraction File(r) Card, exported by their software.

ContentsValidator(*filename*)

Look for a str tag in 1st line

ReadPDFPhase(*G2frame*, *fp*)

Read a phase from a ICDD .str file.

Reader(*filename*, *ParentFrame=None*, ***unused*)

Read phase from a ICDD .str file using *ReadPDFPhase()*

21.2.2 *Module G2phase_GPX: Import phase from GSAS-II project*

Copies a phase from another GSAS-II project file into the current project. Class to read a phase from an existing GSAS-II project file

class GSASII.imports.G2phase_GPX.**PhaseReaderClass**

Opens a .GPX file and pulls out a selected phase

ContentsValidator(*filename*)

Test if the 1st section can be read as a pickle block, if not it can't be .GPX!

Reader(*filename*, *ParentFrame=None*, ***unused*)

Read a phase from a .GPX file. Does not (yet?) support selecting and reading more than one phase at a time.

21.2.3 *Module G2phase_CIF: Coordinates from CIF*

Parses a CIF using PyCifRW from James Hester (https://github.com/jamesrhester/pycifrw) and pulls out the structural information.

If a CIF generated by ISODISTORT is encountered, extra information is added to the phase entry and constraints are generated.

class GSASII.imports.G2phase_CIF.**CIFPhaseReader**

Implements a phase importer from a possibly multi-block CIF file

ContentsValidator(*filename*)

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

ISODISTORT_proc(*blk*, *atomlbllist*, *ranIdlookup*, *filename*)

Process ISODISTORT items to create constraints etc. Constraints are generated from information extracted from loops beginning with _iso_ and are placed into self.Constraints, which contains a list of *constraints tree items* and one dict. The dict contains help text for each generated ISODISTORT variable

At present only _iso_displacivemode…, _iso_occupancymode… and _iso_magneticmode… are processed. Not yet processed: _iso_rotationalmode… & _iso_strainmode…

ISODISTORT_test(*blk*)

Test if there is any ISODISTORT information in CIF

At present only _iso_displacivemode…, _iso_occupancymode… and _iso_magneticmode_… are tested.

GSASII.imports.G2phase_CIF.**ISODISTORT_shortLbl**(*lbl*, *shortmodelist*)

Shorten model labels and remove special characters

GSASII.imports.G2phase_CIF.**parse_parent2mag**(*s*)

Parse _iso_parent-to-child.transform_Pp_abc with strings like:
'a,-b,-c;0.5,0,0' 'a+b,-a+b,c;0,0,0'

into:
(matrix, vector)

Returns:
matrix: list[list[int]] vector: list[float]]

21.2.4 *Module G2phase_INS: Import phase from SHELX INS file*

Copies a phase from SHELX ins file into the current project.

class GSASII.imports.G2phase_INS.**PhaseReaderClass**

Opens a .INS file and pulls out a selected phase

ContentsValidator(*filename*)

Test if the ins file has a CELL record

ReadINSPhase(*filename*, *parent=None*)

Read a phase from a INS file.

Reader(*filename*, *filepointer*, *ParentFrame=None*, ***unused*)

Read a ins file using *ReadINSPhase()*

21.2.5 *Module G2phase_rmc6f: Import phase from RMCProfile*

Copies a phase from a file written by RMCProfile into the current GSAS-II project. Class to read a phase from a RMCprofile output file

class GSASII.imports.G2phase_rmc6f.**PhaseReaderClass**

Opens a .rmc6f file and pulls out the phase

ContentsValidator(*filename*)

Test if the rmc6f file has a CELL record

Reader(*filename*, *filepointer*, *ParentFrame=None*, ***unused*)

Read a rmc6f file using *Readrmc6fPhase()*

Readrmc6fPhase(*filename*, *parent=None*)

Read a phase from a rmc6f file.

21.2.6 *Module G2phase_xyz: read coordinates from an xyz file*

A short routine to read in a phase from an xyz Cartesian coordinate file

class GSASII.imports.G2phase_xyz.**XYZ_ReaderClass**

Routine to import Phase information from a XYZ file

ContentsValidator(*filename*)

Taking a stab a validating: 1st line should be a number

Reader(*filename*, *ParentFrame=None*, ***unused*)

Read a phase from an XYZ file.

21.2.7 *Module G2phase_RRUFF: read from RRUFF database*

A short routine to read in a phase from the RRUFF database in the native format it uses. Reads structure files downloaded from the RRUFF database either as ASCII text files or .rtf files that somehow are generated

class GSASII.imports.G2phase_RRUFF.**RRUFFReader**

A fairly quickly-written importer to pull out the phase info from a RRUFF database (https://rruff.info) text file.

ContentsValidator(*filename*)

Test if the file has CELL PARAMETERS: and SPACE GROUP: entries

Reader(*filename*, *ParentFrame=None*, ***unused*)

Read a DIF file from RRUFF

21.3 Powder Data Importer Routines

Powder data importer routines are classes derived from `GSASIIobj.ImportPowderData`. They must be found in files named *G2pwd*.py* that are in the Python path and the class must override the `__init__` method and add a `Reader` method.

The distributed powder data importers are:

21.3.1 *Module G2pwd_GPX: GSAS-II projects*

Routine to importer powder data from GSAS-II .gpx files

class GSASII.imports.G2pwd_GPX.**GSAS2_ReaderClass**

Routines to import powder data from a GSAS-II file This should work to pull data out from a out of date .GPX file as long as the details of the histogram data itself don't change

ContentsValidator(*filename*)

Test if the 1st section can be read as a pickle block, if not it can't be .GPX!

Reader(*filename*, *ParentFrame=None*, ***kwarg*)

Read a dataset from a .GPX file. If multiple datasets are requested, use self.repeat and buffer caching.

21.3.2 *Module G2pwd_fxye: GSAS data files*

Routine to read in powder data in a variety of formats that were defined in the original GSAS/EXPGUI software suite.

class GSASII.imports.G2pwd_fxye.**GSAS_ReaderClass**

Routines to import powder data from a GSAS files

ContentsValidator(*filename*)

Validate by checking to see if the file has BANK lines & count them

Reader(*filename*, *ParentFrame=None*, ***kwarg*)

Read a GSAS (old formats) file of type FXY, FXYE, ESD or STD types. If multiple datasets are requested, use self.repeat and buffer caching.

EDS data is only in the STD format (10 values per line separated by spaces); the 1st line contains at col 60 the word "Two-Theta " followed by the appropriate value. The BANK record contains the 3 values (4th not used) after 'EDS' for converting MCA channel number (c) to keV via E = A + Bc + Cc^2; these coefficients are generally predetermined by calibration of the MCA. They & 2-theta are transferred to the Instrument parameters data.

GSASII.imports.G2pwd_fxye.**sfloat**(*S*)

convert a string to a float, treating an all-blank string as zero

GSASII.imports.G2pwd_fxye.**sint**(*S*)

convert a string to an integer, treating an all-blank string as zero

21.3.3 *Module G2pwd_xye: Topas & Fit2D data*

Routine to read in powder data from a number of related formats including ones used in Topas and Fit2D. Typical file extensions are .xye, .qye, .chi, and .qchi. Importer for various two/three column formats with 2theta vs intensity or Q vs intensity with an optional 3rd column for s.u.(I)

class GSASII.imports.G2pwd_xye.**xye_ReaderClass**

Routines to import powder data from a .xye/.chi file

ContentsValidator(*filename*)

Look through the file for expected types of lines in a valid Topas Fit2D or BNL/pyFAI file. Alas the latter two formats are somewhat in conflict.

Reader(*filename*, *ParentFrame=None*, ***unused*)

Read a Topas file

21.3.4 *Module G2pwd_CIF: CIF powder data*

Routine to read in powder data from a CIF. Parses a CIF using PyCifRW from James Hester (https://github.com/jamesrhester/pycifrw). Class to read a phase from a CIF

class GSASII.imports.G2pwd_CIF.**CIFpwdReader**

Routines to import powder data from a CIF file

ContentsValidator(*filename*)

Use standard CIF validator

Reader(*filename*, *ParentFrame=None*, ***kwarg*)

Read powder data from a CIF. If multiple datasets are requested, use self.repeat and buffer caching.

21.3.5 *Module G2pwd_BrukerRAW: Bruker .raw*

Routine to read in powder data from most Bruker versions 2, 3, or 4 .raw files.

class GSASII.imports.G2pwd_BrukerRAW.**raw_ReaderClass**

Routines to import powder data from a binary Bruker .RAW file

ContentsValidator(*filename*)

Look through the file for expected types of lines in a valid Bruker RAW file

Reader(*filename*, *ParentFrame=None*, ***kwarg*)

Read a Bruker RAW file

21.3.6 *Module G2pwd_BrukerBRML: Bruker .brml*

Routine to read in powder data from a Bruker .brml file.

class GSASII.imports.G2pwd_BrukerBRML.**brml_ReaderClass**

Routines to import powder data from a zip Bruker .brml file

ContentsValidator(*filename*)

Validate by testing if the file can be opened by zip and if so, does it contain at least one file named "RawData*.xml"

Reader(*filename*, *ParentFrame=None*, ***kwarg*)

Read a Bruker brml file

21.3.7 *Module G2pwd_FP: FullProf .dat data*

Routine to read in powder data from a FullProf .dat file

class GSASII.imports.G2pwd_FP.**fp_ReaderClass**

Routines to import powder data from a FullProf 1-10 column .dat file

ContentsValidator(*filename*)

Look through the file for expected types of lines in a valid FullProf file

Reader(*filename*, *ParentFrame=None*, ***unused*)

Read a FullProf file

21.3.8 *Module G2pwd_Panalytical: Panalytical .xrdml data*

Routines to importer powder data from a Pananalytical (XML) .xrdm file.

class GSASII.imports.G2pwd_Panalytical.**Panalytical_ReaderClass**

Routines to import powder data from a Pananalytical.xrdm (xml) file.

ContentsValidator(*filename*)

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

Reader(*filename*, *ParentFrame=None*, ***kwarg*)

Read a Panalytical .xrdml (.xml) file; already in self.root

21.3.9 *Module G2pwd_csv: Read Excel .csv data*

Routine to read in powder data from Excel type comma separated variable column-oriented variable. The only allowed extensions for this are .csv, .xy, or .XY.

class GSASII.imports.G2pwd_csv.**csv_ReaderClass**

Routines to import powder data from a .xye file

ContentsValidator(*filename*)

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

Reader(*filename*, *ParentFrame=None*, ***unused*)

Read a csv file

21.3.10 *Module G2pwd_rigaku: powder data from a Rigaku .txt file*

class GSASII.imports.G2pwd_rigaku.**Rigaku_rasReaderClass**

Routines to import powder data from a Rigaku .ras file with multiple scans. All scans will be imported as individual PWDR entries

ContentsValidator(*filename*)

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

Reader(*filename*, *ParentFrame=None*, ***kwarg*)

Read a Rigaku .ras/.rasx file

class GSASII.imports.G2pwd_rigaku.**Rigaku_txtReaderClass**

Routines to import powder data from a Rigaku .txt file with an angle and then 1 or 11(!) intensity values on the line. The example file is proceeded with 10 of blank lines, but I have assumed they could be any sort of text. This code should work with an angle and any number of intensity values/line as long as the number is the same on each line. The step size may not change. The number of comment lines can also change, but should not appear to be intensity values (numbers only).

ContentsValidator(*filename*)

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

Reader(*filename*, *ParentFrame=None*, ***kwarg*)

Read a Rigaku .txt file

21.3.11 *Module G2pwd_MIDAS: Read integration results from MIDAS*

The MIDAS package can be used on to read and integrate images, particularly at APS Sector 1. This importer reads the Zarr container files created by MIDAS. Import a collection of "lineouts" from MIDAS from a zarr zip file

class GSASII.imports.G2pwd_MIDAS.**MIDAS_Zarr_Reader**

Routine to read multiple powder patterns from a Zarr file created by MIDAS. Perhaps in the future, other software might also use this file type as well.

For Midas, the main file is <file>.zip, but optionally sample and instrument parameters can be placed in <file>.samprm and <file>.instprm. Any parameters placed in those files will override values set in the .zip file.

ContentsValidator(*filename*)

Test if valid by seeing if the zarr module recognizes the file. Then get file type (currently Midas only)

Reader(*filename*, *ParentFrame=None*, ***kwarg*)

Scan file for sections needed by defined file types (currently only Midas) and then use appropriate routine to read the file. Most of the time the results are placed in the buffer arg (if supplied) so the file is not read on most calls.

For MIDAS, masking can eliminate some or all points in an azimuthal region. This will only return "lineouts" (aka diffractograms) that have 20 or more points in them.

Note that if Reader.selections is used to select individual "lineouts", the selections are numbered against all possible "lineouts" not the ones that have 20 or more points.

readMidas(*filename*, *fpbuffer={}*)

Read zarr file produced by Midas

21.4 Single Crystal Data Importer Routines

Single crystal data importer routines are classes derived from , `GSASIIobj.ImportStructFactor`. They must be found in files named *G2sfact*.py* that are in the Python path and the class must override the `__init__` method and add a `Reader` method. The distributed routines are:

21.4.1 *Module G2sfact: simple HKL import*

Read structure factors from a number of hkl file types. Routines are provided to read from files containing F or F^2 values from a number of sources. Classes to read single crystal reflection files in formats used by: Shelx, Jana, REMOS, TOPAS (SNS), HB-3A (HIFR)

GSASII.imports.G2sfact.**ColumnValidator**(*parent*, *filepointer*, *nCol=5*)

Validate a file to check that it contains columns of numbers

class GSASII.imports.G2sfact.**GSAS2_INT_ReaderClass**

Routines to import reflections from a GSAS-II gpx file

ContentsValidator(*filename*)

Test if the 1st section can be read as a pickle block, if not it can't be .GPX!

Reader(*filename*, *ParentFrame=None*, ***kwarg*)

Read an HKLF dataset from a .GPX file. If multiple datasets are requested, use self.repeat and buffer caching.

class GSASII.imports.G2sfact.**HKLF_ReaderClass**

Routines to import F, sig(F) reflections from a HKLF file

ContentsValidator(*filename*)
Make sure file contains the expected columns on numbers

Reader(*filename*, *ParentFrame=None*, ***unused*)
Read the file

class GSASII.imports.G2sfact.**HKLMF_ReaderClass**
Routines to import F, reflections from a REMOS HKLMF file

ContentsValidator(*filename*)
Make sure file contains the expected columns on numbers

Reader(*filename*, *ParentFrame=None*, ***unused*)
Read the file

class GSASII.imports.G2sfact.**M90_ReaderClass**
Routines to import F**2, sig(F**2) reflections from a JANA M90/M91 file

ContentsValidator(*filename*)
Discover how many columns are in the m90 file - could be 9-12 depending on satellites

Reader(*filename*, *filepointer*, *ParentFrame=None*, ***unused*)
Read the file

class GSASII.imports.G2sfact.**NT_HKLF2_ReaderClass**
Routines to import neutron TOF F**2, sig(F**2) reflections from a HKLF file

ContentsValidator(*filename*)
Make sure file contains the expected columns on numbers & count number of data blocks - "Banks"

Reader(*filename*, *ParentFrame=None*, ***unused*)
Read the file

class GSASII.imports.G2sfact.**NT_JANA2K_ReaderClass**
Routines to import neutron TOF F**2, sig(F**2) reflections from a JANA2000 file

ContentsValidator(*filename*)
Make sure file contains the expected columns on numbers & count number of data blocks - "Banks"

Reader(*filename*, *filepointer*, *ParentFrame=None*, ***unused*)
Read the file

class GSASII.imports.G2sfact.**SHELX4ED_ReaderClass**
Routines to import F**2, sig(F**2), exp# & Z-pos reflections from a Shelx HKLF 4 microED file

ContentsValidator(*filename*)
Make sure file contains the expected columns on numbers

Reader(*filename*, *ParentFrame=None*, ***unused*)
Read the file

class GSASII.imports.G2sfact.**SHELX4_ReaderClass**
Routines to import F**2, sig(F**2) reflections from a Shelx HKLF 4 file

ContentsValidator(*filename*)
Make sure file contains the expected columns on numbers

Reader(*filename*, *ParentFrame=None*, ***unused*)
Read the file

class GSASII.imports.G2sfact.**SHELX5_ReaderClass**

Routines to import F**2, sig(F**2) twin/incommensurate reflections from a fixed format SHELX HKLF5 file

ContentsValidator(*filename*)

Discover how many columns before F^2 are in the SHELX HKL5 file - could be 3-6 depending on satellites

Reader(*filename*, *ParentFrame=None*, ***unused*)

Read the file

class GSASII.imports.G2sfact.**SHELX6_ReaderClass**

Routines to import F**2, sig(F**2) twin/incommensurate reflections from a fixed format SHELX HKLF6 file

ContentsValidator(*filename*)

Discover how many columns before F^2 are in the SHELX HKL6 file - could be 3-6 depending on satellites

Reader(*filename*, *ParentFrame=None*, ***unused*)

Read the file

class GSASII.imports.G2sfact.**hb3a_INT_ReaderClass**

Routines to import neutron CW F**2, sig(F**2) reflections from a NIST hb3a int file

ContentsValidator(*filename*)

Make sure file contains the expected columns on numbers & count number of data blocks - "Banks"

Reader(*filename*, *filepointer*, *ParentFrame=None*, ***unused*)

Read the file

21.4.2 *Module G2sfact_CIF: CIF import*

Read structure factors from a CIF reflection table CIF using PyCifRW from James Hester (https://github.com/jamesrhester/pycifrw). Class to read single-crystal data from a CIF

class GSASII.imports.G2sfact_CIF.**CIFhklReader**

Routines to import Phase information from a CIF file

ContentsValidator(*filename*)

Use standard CIF validator

Reader(*filename*, *ParentFrame=None*, ***kwarg*)

Read single crystal data from a CIF. If multiple datasets are requested, use self.repeat and buffer caching.

21.5 Small Angle Scattering Data Importer Routines

Small angle scattering data importer routines are classes derived from , `GSASIIobj.ImportSmallAngleData`. They must be found in files named *G2sad*.py* that are in the Python path and the class must override the `__init__` method and add a `Reader` method. The distributed routines are in:

21.5.1 *Module G2sad_xye: read small angle data*

Routines to read in small angle data from an .xye type file, with two-theta or Q steps. Expected extensions are .xsad, .xdat, .nsad, or .ndat.

class GSASII.imports.G2sad_xye.**txt_NeutronReaderClass**

Routines to import neutron q SAXD data from a .nsad or .ndat file

ContentsValidator(*filename*)

Look through the file for expected types of lines in a valid q-step file

class GSASII.imports.G2sad_xye.**txt_XRayReaderClass**

Routines to import X-ray q SAXD data from a .xsad or .xdat file

ContentsValidator(*filename*)

Look through the file for expected types of lines in a valid q-step file

class GSASII.imports.G2sad_xye.**txt_nmNeutronReaderClass**

Routines to import neutron q in nm-1 SAXD data from a .nsad or .ndat file

ContentsValidator(*filename*)

Look through the file for expected types of lines in a valid q-step file

class GSASII.imports.G2sad_xye.**txt_nmXRayReaderClass**

Routines to import X-ray q SAXD data from a .xsad or .xdat file, q in nm-1

ContentsValidator(*filename*)

Look through the file for expected types of lines in a valid q-step file

21.6 Image Importer Routines

Image importer routines are classes derived from `GSASIIobj.ImportImage`. See *Writing a Importer Routine* for general information on importers and the `GSASIIobj.ImportImage` for information on what class variables a reader should set. Image importers must be found in files named *G2img*.py* that are in the Python path and the class must override the `__init__` method and add a `Reader` method.

The distributed routines are:

21.6.1 *Module G2img_ADSC: .img image file*

class GSASII.imports.G2img_ADSC.**ADSC_ReaderClass**

Reads an ADSC .img file

ContentsValidator(*filename*)

no test at this time

GSASII.imports.G2img_ADSC.**GetImgData**(*filename*, *imageOnly=False*)

Read an ADSC image file

21.6.2 *Module G2img_EDF: .edf image file*

class GSASII.imports.G2img_EDF.**EDF_ReaderClass**

Routine to read a Read European detector data .edf file. This is a particularly nice standard.

ContentsValidator(*filename*)

no test used at this time

GSASII.imports.G2img_EDF.**GetEdfData**(*filename*, *imageOnly=False*)

Read European detector data edf file

21.6.3 *Module G2img_SumG2: Python pickled image*

Routine to read an image from GSAS-II that has been pickled in Python. Images in this format are created by the "Sum image data" command. At least for now, only one image is permitted per file.

class GSASII.imports.G2img_SumG2.**G2_ReaderClass**

Routine to read an image that has been pickled in Python. Images in this format are created by the "Sum image data" command. At least for now, only one image is permitted per file.

ContentsValidator(*filename*)

test by trying to unpickle (should be quick)

Reader(*filename*, *ParentFrame=None*, ***unused*)

Read using pickle

21.6.4 *Module G2img_GE: summed GE image file*

Read data from General Electric angiography x-ray detectors, primarily as used at APS 1-ID. This shows an example of an importer that will handle files with more than a single image.

class GSASII.imports.G2img_GE.**GE_ReaderClass**

Routine to read a GE image, typically from APS Sector 1.

The image files may be of form .geX (where X is ' ', 1, 2, 3, 4 or 5), which is a raw image from the detector. These files may contain more than one image and have a rudimentary header. Files with extension .sum or .cor are 4 byte integers/pixel, one image/file. Files with extension .avg are 2 byte integers/pixel, one image/file.

ContentsValidator(*filename*)

just a test on file size

Reader(*filename*, *ParentFrame=None*, ***kwarg*)

Read using GE file reader, *GetGEsumData()*

class GSASII.imports.G2img_GE.**GEsum_ReaderClass**

Routine to read multiple GE images & sum them, typically from APS Sector 1.

The image files may be of form .geX (where X is ' ', 1, 2, 3, 4 or 5), which is a raw image from the detector. These files may contain more than one image and have a rudimentary header. Files with extension .sum or .cor are 4 byte integers/pixel, one image/file. Files with extension .avg are 2 byte integers/pixel, one image/file.

ContentsValidator(*filename*)

just a test on file size

Reader(*filename*, *ParentFrame=None*, ***kwarg*)

Read using GE file reader, *GetGEsumData()*

GSASII.imports.G2img_GE.**GetGEsumData**(*self*, *filename*, *imagenum=1*, *sum=False*)

Read G.E. detector images from various files as produced at 1-ID and with Detector Pool detector. Also sums multiple image files if desired

21.6.5 *Module G2img_MAR: MAR image files*

GSASII.imports.G2img_MAR.**GetMAR345Data**(*filename*, *imageOnly=False*)

Read a MAR-345 image plate image

class GSASII.imports.G2img_MAR.**MAR_ReaderClass**

Routine to read several MAR formats, .mar3450,.mar2300,.mar2560

ContentsValidator(*filename*)

no test at this time

21.6.6 *Module G2img_Rigaku: .stl image file*

GSASII.imports.G2img_Rigaku.**GetRigaku**(*filename*, *imageOnly=False*)

Read Rigaku R-Axis IV image file

class GSASII.imports.G2img_Rigaku.**Rigaku_ReaderClass**

Routine to read a Rigaku R-Axis IV image file.

ContentsValidator(*filename*)

Test by checking if the file size makes sense.

21.6.7 *Module G2img_1TIF: Tagged-image File images*

Routine to read an image in Tagged-image file (TIF) format as well as a variety of slightly incorrect pseudo-TIF formats used at instruments around the world. This uses a custom reader that attempts to determine the instrument and detector parameters from various aspects of the file, not always successfully alas. Note that the name G2img_1TIF is used so that this file will sort to the top of the image formats and thus show up first in the menu. (It is the most common, alas).

GSASII.imports.G2img_1TIF.**GetTifData**(*filename*, *imageOnly=False*)

Read an image in a pseudo-tif format, as produced by a wide variety of software, almost always incorrectly in some way.

GSASII.imports.G2img_1TIF.**TIFValidator**(*filename*)

Does the header match the required TIF header?

class GSASII.imports.G2img_1TIF.**TIF_ReaderClass**

Reads TIF files using a routine (*GetTifData()*) that looks for files that can be identified from known instruments and will correct for slightly incorrect TIF usage.

ContentsValidator(*filename*)

Does the header match the required TIF header?

Reader(*filename*, *ParentFrame=None*, ***unused*)

Read the TIF file using *GetTifData()* which attempts to recognize the detector type and set various parameters

21.6.8 *Module G2img_PILTIF: Std Tagged-image File images*

Routine to read an image in Tagged-image file (TIF) format using a standard image library function in Pillow or the now obsolete PIL package. This means that parameters such as the pixel size (which is in the TIFF header but is almost never correct) and distance to sample, etc. are not correct unless specified in a separate metadata file. See below for more information on metadata files. Read TIF files using the PIL/Pillow module.

The metadata can be specified in a file with the same name and path as the TIFF file except that the the extension is .metadata.

The contents of that file are a series of lines of form:

```
keyword = value
```

Note that capitalization of keywords is ignored. Defined keywords are in table below. Any line without one of these keywords will be ignored.

keyword	explanation
wavelength	Wavelength in Å
distance	Distance to sample in mm
polarization	Percentage polarized in horizontal plane
sampleChangerCoordinate	Used for sample changers to track sample
pixelSizeX	Pixel size in X direction (microns)
pixelSizeY	Pixel size in Y direction (microns)
CenterPixelX	Location of beam center as a pixel number (in X)
CenterPixelY	Location of beam center as a pixel number (in X)

class GSASII.imports.G2img_PILTIF.**TIF_LibraryReader**

Reads TIF files using a standard library routine. Metadata (such as pixel size) must be specified by user, either in GUI or via a metadata file. The library TIF reader can handle compression and other things that are not commonly used at beamlines.

ContentsValidator(*filename*)

Does the header match the required TIF header?

Reader(*filename*, *ParentFrame=None*, ***unused*)

Read the TIF file using the PIL/Pillow reader and give the user a chance to edit the likely wrong default image parameters.

21.6.9 *Module G2img_png: png image file*

Routine to read an image in .png (Portable Network Graphics) format. For now, the only known use of this is with converted Mars Rover (CheMin) tif files, so default parameters are for that.

class GSASII.imports.G2img_CheMin.**png_ReaderClass**

Reads standard PNG images; parameters are set to those of the Mars Rover (CheMin) diffractometer.

ContentsValidator(*filename*)

no test at this time

Reader(*filename*, *ParentFrame=None*, ***unused*)

Reads using standard scipy PNG reader

21.6.10 *Module G2img_CBF: .cbf cif image file*

class GSASII.imports.G2img_CBF.**CBF_ReaderClass**

Routine to read a Read cif image data .cbf file. This is used by Pilatus.

ContentsValidator(*filename*)

no test used at this time

Reader(*filename*, *ParentFrame=None*, ***unused*)

Read using Bob's routine *GetCbfData()*

GSASII.imports.G2img_CBF.**GetCbfData**(*self*, *filename*)

Read cif binary detector data cbf file

21.6.11 *Module G2img_HDF5: summed HDF5 image file*

Reads images found in a HDF5 file. If the file contains multiple images, all are read. A reader for HDF-5 files. This should be as generic as possible, but at present this is pretty much customized for XSD-MPE (APS) uses.

Note that for further development of this routine, as more types of HDF5 image files occur "in the wild," it is often helpful to map out the contents of a HDF5 file. If debug mode is on and the full file name/path contains either 'tmp' or 'scratch' (case is ignored) then the two files are created with filename + _HDF5Map.txt and + _NeXusMap.txt that use HDF5 and NeXus routines to outline the file contents.

class GSASII.imports.G2img_HDF5.**HDF5_Reader**

Routine to read one or more HDF-5 images from a HDF5 file, typically from APS Sectors 1, 6 or 20.

Initial version from Barbara Frosik/SDM.

ContentsValidator(*filename*)

Test if valid by seeing if the HDF5 library recognizes the file.

Reader(*filename*, *ParentFrame=None*, ***kwarg*)

Read an image from a HDF5 file. Note that images are using *readDataset()*.

When called the first time on a file, the file structure is scanned using *visit()* to map out locations of image(s). On subsequent calls, if more than one image is in the file, the map of file structure (in buffer arg) is reused. Depending on the Config setting for HDF5selection, a window may be opened to allow selection of which images will be read.

When an image is reread, the blocknum will be a list item with the location to be read, so the file scan can be skipped.

readDataset(*fp*, *imagenum=1*, *name=None*, *num=None*)

Read a specified image number from a file

visit(*fp*, *log=None*)

Recursively visit every node in an HDF5 file & look at dimensions of contents. If the shape is length 2, 3, or 4 assume an image and index in self.buffer['imagemap']. Optionally save an outline of the file contents on log, if defined.

Parameters

- **fp** – an HDF5 file object from h5py.File()
- **log** – an optional text file object [from open()]. If supplied, an outline of the file contents is placed here.

21.6.12 *Module G2img_SFRM: Brucker .sfrm image file*

GSASII.imports.G2img_SFRM.**GetGFRMData**(*self*, *filename*)

Read Bruker compressed binary detector data gfrm file

GSASII.imports.G2img_SFRM.**GetSFRMData**(*self*, *filename*)

Read cbf compressed binarydetector data sfrm file

class GSASII.imports.G2img_SFRM.**SFRM_ReaderClass**

Routine to read a Read Bruker Advance image data .sfrm/.grfm file.

ContentsValidator(*filename*)

GFRM files always begin with FORMAT, should also contain VERSION (also HDRBLKS, but not checked) No check for SRFM files

Reader(*filename*, *ParentFrame=None*, ***unused*)

Read using Bob's routine *GetSFRMData()*

21.6.13 Module G2img_pixirad_1ID_16bit: Pixirad detector

Reads images from the pixirad detector in use as APS Sector 1.

Module G2img_pixirad_1ID_16bit.py: 16-bit Pixirad TIF

Adaptation of G2img_1ID_32bit_TIFsum.py for use with the 1ID SAXS pixirad. Used at 1-ID with normalized 16bit Pixirad TIF for rapid access measurement

GSASII.imports.G2img_pixirad_1ID_16bit.**GetTifData**(*filename*)

Confirm that the file is in fact a 16bit tiff image with the correct parameters. Will nearly always throw an error unless used with the 1ID GSASII workflow, which assigns the proper tags and converts the Pixirad tiff images into the 16bit format with a different buffer in order to hold the larger amount of data in each pixel.

class GSASII.imports.G2img_pixirad_1ID_16bit.**TIF_ReaderClass**

Reads TIF files using a routine (*GetTifData()*) that looks for files that can be identified from known instruments and will correct for slightly incorrect TIF usage. If that routine fails, it will be read with a standard TIF reader, which can handle compression and other things less commonly used at beamlines.

ContentsValidator(*filename*)

Does the header match the required TIF header?

Reader(*filename*, *ParentFrame=None*, ***unused*)

Read the TIF file using *GetTifData()*.

21.7 Pair Distribution Function (PDF) Importer Routines

PDF importer routines are classes derived from GSASIIobj.ImportPDFData. See *Writing a Importer Routine* for general information on importers.

The distributed routines are in:

21.7.1 Module G2pdf_gr: read PDF G(R) data

Routines to read in G(R) data from a pdfGet/GSAS-II .gr or gudrun .dat file (with Å steps) or S(Q) data from a .fq file.

class GSASII.imports.G2pdf_gr.**txt_FSQReaderClass**

Routines to import S(Q) data from a .fq file

ContentsValidator(*filename*)

Look through the file for expected types of lines in a valid r-step file

class GSASII.imports.G2pdf_gr.**txt_PDFReaderClass**

Routines to import PDF G(R) data from a .gr file

ContentsValidator(*filename*)

Look through the file for expected types of lines in a valid r-step file

class GSASII.imports.G2pdf_gr.**txt_PDFReaderClassG**

Routines to import PDF G(R) data from a .dat file

ContentsValidator(*filename*)

Look through the file for expected types of lines in a valid r-step file

21.8 Reflectometry Importer Routines

Reflectometry importer routines are classes derived from `GSASIIobj.ImportReflectometryData`. See *Writing a Importer Routine* for general information on importers.

The distributed routines are:

21.8.1 *Module G2rfd_xye: read reflectometry data*

Routines to read in reflectometry data from an .xrfd, .xdat, .xtrfd, .xtdat, .nrfd or .ndat type file, with two-theta or Q steps.

class GSASII.imports.G2rfd_xye.**txt_NeutronReaderClass**

Routines to import neutron q REFD data from a .nrfd or .ndat file

ContentsValidator(*filename*)

Look through the file for expected types of lines in a valid q-step file

class GSASII.imports.G2rfd_xye.**txt_XRayReaderClass**

Routines to import X-ray q REFD data from a .xrfd or .xdat file

ContentsValidator(*filename*)

Look through the file for expected types of lines in a valid q-step file

class GSASII.imports.G2rfd_xye.**txt_XRayThetaReaderClass**

Routines to import X-ray theta REFD data from a .xtrfd or .xtdat file

ContentsValidator(*filename*)

Look through the file for expected types of lines in a valid q-step file

21.8.2 *Module G2rfd_Panalytical: read Panalytical reflectometry data*

Routine to importer reflectivity data from a Panalytical .xrdm (xml) file.

class GSASII.imports.G2rfd_Panalytical.**Panalytical_ReaderClass**

Routines to import reflectivity data from a Panalytical.xrdm (xml) file.

ContentsValidator(*filename*)

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

Reader(*filename*, *ParentFrame=None*, ***kwarg*)

Read a Panalytical .xrdml (.xml) file; already in self.root

21.8.3 *Module G2rdf_rigaku: reflectometry data from a Rigaku file*

class GSASII.imports.G2rfd_rigaku.**Rigaku_rasReaderClass**

Routines to import reflectometry data from a Rigaku .ras file with multiple scans. All scans will be imported as individual PWDR entries

ContentsValidator(*filename*)

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

Reader(*filename*, *ParentFrame=None*, ***kwarg*)

Read a Rigaku .ras/.rasx file

class GSASII.imports.G2rfd_rigaku.**Rigaku_txtReaderClass**

Routines to import powder data from a Rigaku .txt file with an angle and then 1 or 11(!) intensity values on the line. The example file is proceeded with 10 of blank lines, but I have assumed they could be any sort of text. This code should work with an angle and any number of intensity values/line as long as the number is the same on each line. The step size may not change. The number of comment lines can also change, but should not appear to be intensity values (numbers only).

ContentsValidator(*filename*)

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

Reader(*filename*, *ParentFrame=None*, ***kwarg*)

Read a Rigaku .txt file

CHAPTER

TWENTYTWO

GSAS-II EXPORT MODULES

Many of the data files written by GSAS-II for use by other software is written using a layer of routines called exporters. Exporters usually require quite simple code, so they can be written quickly to allow GSAS-II to provide output in new data formats. The interface to the exporters is self-configuring, so all supplied exporters are available once the exporter is added to the code base. This allows GSAS-II to be quite flexible in adapting to use many data formats without need for extensive coding.

Exports are implemented by deriving a class from *ExportBaseclass* in module *GSASIIfiles*. Export routines commonly access the GUI to determine information on the contents of the file(s) to be written, but for powder diffraction and phase data export, a routine named `Writer()` may be defined. If this is present, the exporter can be used from: mod:*~GSASII.GSASIIscriptable* without GUI access. Note that the arguments for the``Writer()`` method include a histogram tree name as well as a file name to be written.

A file containing one or more export routine can be placed either in the `GSASII/exports` directory (which requires modification of the `__init__.py` and the `meson.build` files) or the file can be placed in the `~/.GSASII/exports` directory. (Note that ~ here is translated to the user's home directory; for Windows this is usually taken from the USERPROFILE setting or a combination of HOMEPATH and HOMEDRIVE, so this directory will usually have form `C:\\Users\\YourUsername\\.GSASII\\exports`. The next time GSAS-II is started, the file will be loaded with all the other GSAS-II files and the new data format will appear in the appropriate exporter menu.

22.1 Writing an Exporter Routine

When writing a exporter routine, one should create a new class derived from class *~GSASII.GSASIIfiles.ExportBaseclass*. The name of the class is arbitrary, but if more than one class is placed in file, each class must have a different name. The same name can be repeated if it is in different files. As described below, this class will implement an `__init__()` and an `Exporter()` method, and many will supply a `Writer()` method, too. The purpose of each of these routines is described below. The easiest way to craft a new exporter will be to use the other exporters of the same data type as a model for where to find the data values that will be written, but the documentation for the parent class (*~GSASII.GSASIIfiles.ExportBaseclass*) provides useful information on support routines that pull information from the GSAS-II data structures into the exporter.

22.1.1 __init__()

The `__init__` method will follow standard boilerplate largely independent of the data type:

```
def __init__(self):
    super(self.__class__,self).__init__( # fancy way to self-reference
        G2frame=G2frame,
        formatName = 'Format name for menu',
        extension='.extn',
        longFormatName = 'Longer more detailed format name for status line'
       )
```

The first line in the __init__ method calls the parent class __init__ method with the following parameters:

- G2frame: a reference to the main GSAS-II GUI window or None when run scripted.
- formatName: a string to be used in the menu. Should be short.
- extension: a string to be used in the file name. All files produced by the exporter will have this extension.
- longFormatName: a longer string to be used to describe the format in help.

In addition, one instance variables must be defined:

```
self.exporttype = ['phase']
```

The value for self.exporttype determines the type of export that will be performed ('project', 'phase', 'single', 'powder', 'image', 'map', 'sasd', 'refd' or (someday) 'pdf') and the menu where the exporter will be placed. Note that 'project' exports are those that include all data from a .gpx file (all phases, histograms, etc.)

Another item is optional.

```
self.multiple = True
```

The value specified for self.multiple determines if only a single phase, data set, etc. can be exported at a time (when False) or when True, a file can be produced with multiple histograms, phases, etc.

22.1.2 Exporter()

The class must supply a Exporter method that will write a file. Depending on the settings for self.exporttype and self.multiple and the contents of the Data Tree, will dictate what dialogs will be presented to the user to select what will be written.

22.1.3 Writer()

For powder and phase exports, if the class supplies a Writer() module, then the export format will be available for scripted output (with GSASIIscriptable). These modules are supplied a histogram name or a phase name and should not attempt to access the GUI. It is not required that that this method be supplied, but usually it is not hard to do, unless information from the user is required.

Note that for phase exports the Writer() should be declared as follows:

```
def Writer(self,hist,phasenam,mode='w'):
```

while for histogram exports the Writer() should be declared as follows:

```
def Writer(self,hist,filename=None,mode='w'):
```

Module G2export_examples: Examples

Code to demonstrate how GSAS-II data export routines are created. The classes defined here, *ExportPhaseText*, *ExportSingleText*, *ExportPowderReflText*, and *ExportPowderText* each demonstrate a different type of export. Also see *ExportMapASCII* in *G2export_map* for an example of a map export.

22.1.4 G2export_examples Classes and Routines

Classes in *G2export_examples* follow:

class GSASII.exports.G2export_examples.**ExportPhaseText**(*G2frame*)

Used to create a text file for a phase

Parameters
G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export a phase as a text file

class GSASII.exports.G2export_examples.**ExportPowderReflHKLF**(*G2frame*)

Used to create a text file of reflections from a powder data set as shelx HKLF file

Parameters
G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export a set of powder reflections as a shelx HKLF file

class GSASII.exports.G2export_examples.**ExportPowderReflText**(*G2frame*)

Used to create a text file of reflections from a powder data set

Parameters
G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export a set of powder reflections as a text file

class GSASII.exports.G2export_examples.**ExportPowderText**(*G2frame*)

Used to create a text file for a powder data set

Parameters
G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export a set of powder data as a text file

class GSASII.exports.G2export_examples.**ExportSingleText**(*G2frame*)

Used to create a text file with single crystal reflection data skips user rejected & space group extinct reflections

Parameters
G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export a set of single crystal data as a text file

Module G2export_csv: Spreadsheet export

Code to create .csv (comma-separated variable) files for GSAS-II data export to a spreadsheet program, etc. Defines a number of .csv exports:

- *ExportPhaseCSV*: phases
- *ExportPowderCSV*: powder data, includes instrument parameters as well as obs & calc patterns, etc.
- *ExportMultiPowderCSV*: multiple powder datasets in a single spreadsheet
- *ExportPowderReflCSV*: reflections from a powder fit
- *ExportSASDCSV*: small angle data set
- *ExportREFDCSV*: reflectometry data set

- *ExportSingleCSV*: single crystal reflection data
- *ExportStrainCSV*: reflectometry datasets

22.1.5 G2export_csv Classes and Routines

Classes in *G2export_csv* follow:

class GSASII.exports.G2export_csv.**ExportMultiPowderCSV**(*G2frame*)

Used to create a csv file for a stack of powder data sets suitable for display purposes only; no y-calc or weights are exported only x & y-obs

Parameters
G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export a set of powder data as a single csv file

class GSASII.exports.G2export_csv.**ExportPhaseCSV**(*G2frame*)

Used to create a csv file for a phase

Parameters
G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export a phase as a csv file

class GSASII.exports.G2export_csv.**ExportPowderCSV**(*G2frame*)

Used to create a csv file for a powder data set

Parameters
G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export a set of powder data as a csv file

class GSASII.exports.G2export_csv.**ExportPowderReflCSV**(*G2frame*)

Used to create a csv file of reflections from a powder data set

Parameters
G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export a set of powder reflections as a csv file

class GSASII.exports.G2export_csv.**ExportREFDCSV**(*G2frame*)

Used to create a csv file for a reflectometry data set

Parameters
G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export a set of reflectometry data as a csv file

class GSASII.exports.G2export_csv.**ExportSASDCSV**(*G2frame*)

Used to create a csv file for a small angle data set

Parameters
G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export a set of small angle data as a csv file

class GSASII.exports.G2export_csv.**ExportSingleCSV**(*G2frame*)

Used to create a csv file with single crystal reflection data

Parameters

G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export a set of single crystal data as a csv file

class GSASII.exports.G2export_csv.**ExportStrainCSV**(*G2frame*)

Used to create a csv file with single crystal reflection data

Parameters

G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export a set of single crystal data as a csv file

GSASII.exports.G2export_csv.**WriteList**(*obj*, *headerItems*)

Write a CSV header

Parameters

- **obj** (*object*) – Exporter object
- **headerItems** (*list*) – items to write as a header

Module G2export_PDB: Macromolecular export

Code to export a phase into the venerated/obsolete (pick one) ASCII PDB format. Also defines exporter *ExportPhaseCartXYZ* which writes atom positions in orthogonal coordinates for a phase.

22.1.6 G2export_PDB Classes and Routines

Classes in *G2export_PDB* follow:

class GSASII.exports.G2export_PDB.**ExportDrawPhaseCartXYZ**(*G2frame*)

Used to create a Cartesian XYZ file for a phase draw atoms

Parameters

G2frame (*wx.Frame*) – reference to main GSAS-II frame

class GSASII.exports.G2export_PDB.**ExportPhaseCartXYZ**(*G2frame*)

Used to create a basic ORCA6 Cartesian inp file for a phase

Parameters

G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export as a XYZ file

class GSASII.exports.G2export_PDB.**ExportPhasePDB**(*G2frame*)

Used to create a PDB file for a phase

Parameters

G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export as a PDB file

Module G2export_image: 2D Image data export

Demonstrates how an image is retrieved and written. Uses a SciPy routine to write a PNG format file.

22.1.7 G2export_image Classes and Routines

Classes in *G2export_image* follow:

class GSASII.exports.G2export_image.**ExportImagePNG**(*G2frame*)

Used to create a PNG file for a GSAS-II image

Parameters

G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export an image

Module G2export_map: Map export

Code to write Fourier/Charge-Flip atomic density maps out in formats that can be read by external programs. At present a GSAS format that is supported by FOX and DrawXTL (*ExportMapASCII*) and the CCP4 format that is used by COOT (*ExportMapCCP4*) are implemented.

22.1.8 G2export_map Classes and Routines

Classes in *G2export_map* follow:

class GSASII.exports.G2export_map.**ExportMapASCII**(*G2frame*)

Used to create a text file for a phase

Parameters

G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export a map as a text file

class GSASII.exports.G2export_map.**ExportMapCCP4**(*G2frame*)

Used to create a text file for a phase

Parameters

G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export a map as a text file

Write(*data*, *dtype*)

write a line of output, attaching a line-end character

Parameters

line (*str*) – the text to be written.

Module G2export_shelx: Examples

Code to export coordinates in the SHELX .ins format (as best as we can make sense of it).

22.1.9 G2export_shelx Classes and Routines

Classes in *G2export_shelx* follow:

class GSASII.exports.G2export_shelx.**ExportPhaseShelx**(*G2frame*)

Used to create a SHELX .ins file for a phase

Parameters

G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export as a SHELX .ins file

Module G2export_CIF: CIF Exports

This implements a complex set of CIF (Crystallographic Information Framework) exporters. The base class, *ExportCIF*, implement a variety of export capabilities, where extra parameters for *ExportCIF.MasterExporter()* determine if a project, single phase or data set are written. The subclasses of *ExportCIF*, as listed below, supply these different parameters when calling that method.

- *ExportProjectCIF*: writes an entire project in a complete CIF intended for submission as a publication,
- *ExportPhaseCIF*: writes a single phase in CIF
- *ExportPwdrCIF*: writes a one powder diffraction dataset CIF
- ExportHKLFCIF: writes a single crystal dataset

22.1.10 G2export_CIF Classes and Routines

Classes in *G2export_CIF* follow:

GSASII.exports.G2export_CIF.**CIF2dict**(*cf*)

copy the contents of a CIF out from a PyCifRW block object into a dict

Returns

cifblk, loopstructure where cifblk is a dict with CIF items and loopstructure is a list of lists that defines which items are in which loops.

class GSASII.exports.G2export_CIF.**CIFdefHelp**(*parent*, *msg*, *helpwin*, *helptxt*)

Create a help button that displays help information on the current data item

Parameters

- **parent** – the panel which will be the parent of the button
- **msg** (*str*) – the help text to be displayed
- **helpwin** (*wx.Dialog*) – Frame for CIF editing dialog
- **helptxt** (*wx.TextCtrl*) – TextCtrl where help text is placed

class GSASII.exports.G2export_CIF.**CIFtemplateSelect**(*frame*, *panel*, *tmplate*, *G2dict*, *repaint*, *title*, *defaultname=''*, *cifKey='CIF_template'*)

Create a set of buttons to show, select and edit a CIF template

Parameters

- **frame** – wx.Frame object of parent
- **panel** – wx.Panel object where widgets should be placed
- **tmplate** (*str*) – one of 'publ', 'phase', or 'instrument' to determine the type of template
- **G2dict** (*dict*) – GSAS-II dict where CIF should be placed. The key specified in cifKey (defaults to "CIF_template") will be used to store either a list or a string. If a list, it will contain a dict and a list defining loops. If an str, it will contain a file name.
- **repaint** (*function*) – reference to a routine to be called to repaint the frame after a change has been made
- **title** (*str*) – A line of text to show at the top of the window
- **defaultname** (*str*) – specifies the default file name to be used for saving the CIF.
- **cifKey** (*str*) – key to be used for saving the CIF information in G2dict. Defaults to "CIF_template"

class GSASII.exports.G2export_CIF.**EditCIFpanel**(*parent*, *cifblk*, *loopstructure*, *cifdic={}*, *OKbuttons=[]*, ***kw*)

Creates a scrolled panel for editing CIF template items

Parameters

- **parent** (*wx.Frame*) – parent frame where panel will be placed
- **cifblk** – dict or PyCifRW block containing values for each CIF item
- **loopstructure** (*list*) – a list of lists containing the contents of each loop, as an example:

```
[ ["_a","_b"], ["_c"], ["_d_1","_d_2","_d_3"]]
```

this describes a CIF with this type of structure:

```
loop_ _a _b <a1> <b1> <a2> ...
loop_ _c <c1> <c2>...
loop _d_1 _d_2 _d_3 ...
```

Note that the values for each looped CIF item, such as _a, are contained in a list, for example as cifblk["_a"]

- **cifdic** (*dict*) – optional CIF dictionary definitions
- **OKbuttons** (*list*) – A list of wx.Button objects that should be disabled when information in the CIF is invalid
- **(other)** – optional keyword parameters for wx.ScrolledPanel

CIFEntryWidget(*dct*, *item*, *dataname*)

Create an entry widget for a CIF item. Use a validated entry for numb values where int is required when limits are integers and floats otherwise. At present this does not allow entry of the special CIF values of "." and "?" for numerical values and highlights them as invalid. Use a selection widget when there are specific enumerated values for a string.

ControlOKButton(*setvalue*)

Enable or Disable the OK button(s) for the dialog. Note that this is passed into the ValidatedTxtCtrl for use by validators.

Parameters

setvalue (*bool*) – if True, all entries in the dialog are checked for validity. The first invalid control triggers disabling of buttons. If False then the OK button(s) are disabled with no checking of the invalid flag for each control.

DoLayout()

Update the Layout and scroll bars for the Panel. Clears self.LayoutCalled so that next change to panel can request a new update

OnAddRow(*event*)

add a row to a loop

OnLayoutNeeded(*event*)

Called when an update of the panel layout is needed. Calls self.DoLayout after the current operations are complete using CallAfter. This is called only once, according to flag self.LayoutCalled, which is cleared in self.DoLayout.

class GSASII.exports.G2export_CIF.**EditCIFtemplate**(*parent*, *cifblk*, *loopstructure*, *defaultname*)

Create a dialog for editing a CIF template. The edited information is placed in cifblk. If the CIF is saved as a file, the name of that file is saved as `self.newfile`.

Parameters

- **parent** (*wx.Frame*) – parent frame or None
- **cifblk** – dict or PyCifRW block containing values for each CIF item
- **loopstructure** (*list*) – a list of lists containing the contents of each loop, as an example:

```
[ ["_a","_b"], ["_c"], ["_d_1","_d_2","_d_3"]]
```

this describes a CIF with this type of structure:

```
loop_ _a _b <a1> <b1> <a2> ...
loop_ _c <c1> <c2>...
loop _d_1 _d_2 _d_3 ...
```

Note that the values for each looped CIF item, such as _a, are contained in a list, for example as cifblk["_a"]

- **defaultname** (*str*) – specifies the default file name to be used for saving the CIF.

Post()

Display the dialog

Returns

True unless Cancel has been pressed.

class GSASII.exports.G2export_CIF.**ExportCIF**(*G2frame*, *formatName*, *extension*, *longFormatName=None*)

Base class for CIF exports. Not used directly. Exporters are defined in subclasses that call *MasterExporter()*.

MasterExporter(*event=None*, *phaseOnly=None*, *histOnly=None*)

Basic code to export a CIF. Export can be full or simple, as set by phaseOnly and histOnly which skips distances & angles, etc.

Parameters

- **phaseOnly** (*bool*) – used to export only one phase
- **histOnly** (*bool*) – used to export only one histogram

ShowHstrainCells(*phasenam*, *datablockidDict*)

Displays the unit cell parameters for phases where Dij values create mutiple sets of lattice parameters. At present there is no way defined for this in CIF, so local data names are used.

ValidateAscii(*checklist*)

Validate items as ASCII

class GSASII.exports.G2export_CIF.**ExportHKLCIF**(*G2frame*)

Used to create a simple CIF containing diffraction data only. Uses exact same code as *ExportCIF* except that *histOnly* is set for the Exporter Shows up in menu as Quick CIF.

Parameters
G2frame (*wx.Frame*) – reference to main GSAS-II frame

class GSASII.exports.G2export_CIF.**ExportPhaseCIF**(*G2frame*)

Used to create a simple CIF with one phase. Uses exact same code as *ExportCIF* except that *phaseOnly* is set for the Exporter Shows up in menu as Quick CIF.

also called directly in OnISOSearch in GSASIIphsGUI

Parameters
G2frame (*wx.Frame*) – reference to main GSAS-II frame

class GSASII.exports.G2export_CIF.**ExportProjectCIF**(*G2frame*)

Used to create a CIF of an entire project

also called directly in *ExportSequentialFullCIF()* in *GSASIImiscGUI*

Parameters
G2frame (*wx.Frame*) – reference to main GSAS-II frame

class GSASII.exports.G2export_CIF.**ExportPwdrCIF**(*G2frame*)

Used to create a simple CIF containing diffraction data only. Uses exact same code as *ExportCIF* except that *histOnly* is set for the Exporter Shows up in menu as Quick CIF.

Parameters
G2frame (*wx.Frame*) – reference to main GSAS-II frame

Writer(*hist*, *mode='w'*)

Used for histogram CIF export of a sequential fit.

GSASII.exports.G2export_CIF.**FmtAtomType**(*sym*)

Reformat a GSAS-II atom type symbol to match CIF rules

GSASII.exports.G2export_CIF.**HillSortElements**(*elmlist*)

Sort elements in "Hill" order: C, H, others, (where others are alphabetical).

Params list elmlist
a list of element strings

Returns
a sorted list of element strings

GSASII.exports.G2export_CIF.**LoadCIFdic**()

Create a composite core+powder CIF lookup dict containing information about all items in the CIF dictionaries, loading pickled files if possible. The routine looks for files named cif_core.cpickle and cif_pd.cpickle in every directory in the path and if they are not found, files cif_core.dic and/or cif_pd.dic are read.

Returns
the dict with the definitions

GSASII.exports.G2export_CIF.**PickleCIFdict**(*fil*)

Loads a CIF dictionary, cherry picks out the items needed by local code and sticks them into a python dict and writes that dict out as a pickle file for later reuse. If the write fails a warning message is printed, but no exception occurs.

Parameters

fil (*str*) – file name of CIF dictionary, will usually end in .dic

Returns

the dict with the definitions

GSASII.exports.G2export_CIF.**WriteAtomsMM**(*fp*, *phasedict*, *phasenam*, *parmDict*, *sigDict*, *RBparms={}*)

Write atom positions to CIF using mmCIF items

GSASII.exports.G2export_CIF.**WriteAtomsMagnetic**(*fp*, *phasedict*, *phasenam*, *parmDict*, *sigDict*, *labellist*)

Write atom positions to CIF

GSASII.exports.G2export_CIF.**WriteAtomsNuclear**(*fp*, *phasedict*, *phasenam*, *parmDict*, *sigDict*, *labellist*, *RBparms={}*, *RBsuDict={}*)

Write atom positions to CIF

GSASII.exports.G2export_CIF.**WriteCIFitem**(*fp*, *name*, *value=''*)

Helper function for writing CIF output. This gets used in different ways. The simplest use will be:

```
>>> WriteCIFitem(fp, '_some_cif_name', valstr)
```

For loops it will be used like this:

```
>>> WriteCIFitem(fp, 'loop_   _cif_name1  _cif_name2')
>>> for v1,v2 in values:
>>>     WriteCIFitem(fp, value=v1)
>>>     WriteCIFitem(fp, value=v2)
```

or if items will be aligned in a table (no spaces or new lines in the items)

```
>>> WriteCIFitem(fp, 'loop_   _cif_name1  _cif_name2')
>>> for v1,v2 in values:
>>>     s = PutInCol("{:.4f}".format(v1),10)
>>>     s += PutInCol(str(v2),8)
>>>     WriteCIFitem(fp, value=s)
```

It is occasionally used where a CIF value is passed as the name parameter. This works if no quoting is needed, but is not a good practice.

Parameters

- **fp** – file access object
- **name** (*str*) – a CIF data name
- **value** (*str*) – the value associated with the CIF data name. Written in different ways depending on what the contents contain, with respect to quoting.

GSASII.exports.G2export_CIF.**WriteComposition**(*fp*, *phasedict*, *phasenam*, *parmDict*, *quickmode=True*, *keV=None*)

determine the composition for the unit cell, crudely determine Z and then compute the composition in formula units.

If quickmode is False, then scattering factors are added to the element loop.

If keV is specified, then resonant scattering factors are also computed and included.

GSASII.exports.G2export_CIF.**WriteCompositionMM**(*fp*, *phasedict*, *phasenam*, *parmDict*, *quickmode=True*, *keV=None*)

determine the composition for the unit cell, crudely determine Z and then compute the composition in formula units.

If quickmode is False, then scattering factors are added to the element loop.

If keV is specified, then resonant scattering factors are also computed and included.

GSASII.exports.G2export_CIF.**WriteSeqAtomsNuclear**(*fp*, *cell*, *phasedict*, *phasenam*, *hist*, *seqData*, *RBparms*)

Write atom positions to CIF

GSASII.exports.G2export_CIF.**dict2CIF**(*dblk*, *loopstructure*, *blockname='Template'*)

Create a PyCifRW CIF object containing a single CIF block object from a dict and loop structure list.

Parameters

- **dblk** – a dict containing values for each CIF item
- **loopstructure** (*list*) – a list of lists containing the contents of each loop, as an example:

```
[ ["_a","_b"], ["_c"], ["_d_1","_d_2","_d_3"]]
```

this describes a CIF with this type of structure:

```
loop_ _a _b <a1> <b1> <a2> ...
loop_ _c <c1> <c2>...
loop _d_1 _d_2 _d_3 ...
```

Note that the values for each looped CIF item, such as _a, are contained in a list, for example as cifblk["_a"]

- **blockname** (*str*) – an optional name for the CIF block. Defaults to 'Template'

Returns

the newly created PyCifRW CIF object

GSASII.exports.G2export_CIF.**getCellwStrain**(*phasedict*, *seqData*, *pId*, *histname*)

Get cell parameters and their errors for a sequential fit

GSASII.exports.G2export_CIF.**mkSeqResTable**(*mode*, *seqHistList*, *seqData*, *Phases*, *Histograms*, *Controls*)

Setup sequential results table (based on code from *GSASII.GSASIIseqGUI.UpdateSeqResults()*)

TODO: This should be merged with the table build code in *GSASII.GSASIIseqGUI.UpdateSeqResults()* and moved to somewhere non-GUI like *GSASIIstrIO* to create a single routine that can be used in both places, but this means returning some of the code that has been removed from there.

Module G2export_pwdr: Export powder input files

Creates files used by GSAS (FXYE) & TOPAS (XYE) as input

22.1.11 G2export_pwdr Classes and Routines

Classes in *G2export_pwdr* follow:

class GSASII.exports.G2export_pwdr.**ExportPowderFXYE**(*G2frame*)

Used to create a FXYE file for a powder data set

Parameters
G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export one or more sets of powder data as FXYE file(s)

WriteInstFile(*hist*, *Inst*)

Write an instrument parameter file

Writer(*TreeName*, *filename=None*, *prmname=''*)

Write a single PWDR entry to a FXYE file

class GSASII.exports.G2export_pwdr.**ExportPowderXYE**(*G2frame*)

Used to create a Topas XYE file for a powder data set

Parameters
G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export one or more sets of powder data as XYE file(s)

Module G2export_FIT2D: Fit2D "Chi" export

Code to create .chi (Fit2D like) files for GSAS-II powder data export

22.1.12 G2export_FIT2d Classes and Routines

Classes in *G2export_FIT2D* follow:

class GSASII.exports.G2export_FIT2D.**ExportPowderCHI**(*G2frame*)

Used to create a CHI file for a powder data set

Parameters
G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export a set of powder data as a Fit2D .qchi file

class GSASII.exports.G2export_FIT2D.**ExportPowderQCHI**(*G2frame*)

Used to create a q-binned CHI file for a powder data set

Parameters
G2frame (*wx.Frame*) – reference to main GSAS-II frame

Exporter(*event=None*)

Export a set of powder data as a q-bin Fit2D .qchi file

Module G2export_JSON: ASCII .gpx Export

This implements a fairly simple exporter, *ExportJSON*, that can export the contents of an entire project as a sort-of human readable (JSON) ASCII file. This provides a way to see the contents of a GSAS-II project file. This does not provide a mechanism to change the contents of a .gpx file, since there are no provisions to read this file back into GSAS-II, as the likelihood of breaking a data structure is too high. If you want to change the contents of a .gpx file, use GSASIIscriptable where you can access the native Python data structures and change things, with a good chance of getting things to work.

22.1.13 G2export_JSON Classes and Routines

Classes in *G2export_JSON* follow:

This code is to honor my friend Robert Papoular, who wants to see what is inside a .gpx file.

class GSASII.exports.G2export_JSON.**ExportJSON**(*G2frame*)

Implement JSON export of entire projects

class GSASII.exports.G2export_JSON.**JsonEncoder**(** (Keyword-only parameters separator (PEP 3102))*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

This provides the ability to turn np arrays and masked arrays into something that json can handle.

default(*obj*)

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

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

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

Module G2export_Bracket: ASCII .gpx Export

This provides to methods for tabulating GSAS-II parameters from a project for use in manuscript preparation into an ASCII .csv (spreadsheet) file. The exporter, *Exportbracket*, creates a text file with standard uncertainties for values in crystallographic (e.g. "bracket") notation: *i.e.*: `1.234(5)`, which indicates a value of `1.234` with a standard uncertainty of `0.005`. A second method, *Export3col*, provides the standard uncertainties as a separate column.

This module initially written by Conrad Gillard. For any enquiries please contact conrad.gillard@gmail.com.

22.1.14 G2export_Bracket Classes and Routines

Classes in *G2export_Bracket* follow.

class GSASII.exports.G2export_Bracket.**Export3col**(*G2frame*)

Enables export of parameters that are commonly needed for publications, with esds in a separate column

ValEsd2col(*param*, *param_sig*)

Return two values with the formated value as the first number and the standard uncertainty (if provided) as the second value.

class GSASII.exports.G2export_Bracket.**Exportbracket**(*G2frame*)

Enables export of parameters that are commonly needed for publications, in bracket notation

CHAPTER

TWENTYTHREE

GSAS-II INDEPENDENT TOOLS

The modules here are used for independent programs to be used as tools within the GSAS-II package and run independently of the main GSAS-II program.

- *GSASIIIntPDFtool*: Parallelized auto-integration/PDF program
- *G2compare*: Project Comparison program

Both are under development.

23.1 GSASIIIntPDFtool: autointegration routines

An auto-integration program based on GSAS-II but with a minimal GUI and no visualization that runs independently from the main GSAS-II program . This is intended to implement significant levels of parallelization and require less of a memory footprint. Autointegration GSASIIimgGUI and GSASIIimage, hacked for stand-alone use

class GSASII.GSASIIIntPDFtool.**AutoIntFrame**(*G2frame*, *PollTime=30.0*)

Creates a wx.Frame window for the Image AutoIntegration. The intent is that this will be used as a non-modal dialog window.

Implements a Start button that morphs into a pause and resume button. This button starts a processing loop that is repeated every `PollTime()` seconds.

Parameters

- **G2frame** (*wx.Frame*) – main GSAS-II frame
- **PollTime** (*float*) – frequency in seconds to repeat calling the processing loop. (Default is 30.0 seconds.)

ArgGen(*PDFobj*, *imgprms*, *mskprms*, *xydata*)

generator for arguments for integration/PDF calc

OnPause()

Respond to Pause, changes text on button/Status line, if needed Stops timer self.Pause should already be True

OnTimerLoop(*event*)

A method that is called every `PollTime()` seconds that is used to check for new files and process them. Integrates new images. Also optionally sets up and computes PDF. This is called only after the "Start" button is pressed (then its label reads "Pause").

SetSourceDir(*event*)

Use a dialog to get a directory for image files

ShowMatchingFiles(*value*, *invalid=False*, ***kwargs*)

Find and image files matching the image file directory (self.params['readdir']) and the image file filter (self.params['filter']) and add this information to the GUI list box

StartLoop()

Prepare to start autointegration timer loop. Save current Image params for use in future integrations also label the window so users understand what is being used

GSASII.GSASIIIntPDFtool.**LookupFromTable**(*dist*, *parmList*)

Interpolate image parameters for a supplied distance value

Parameters

dist (*float*) – distance to use for interpolation

Returns

a list with 2 items: * a dict with interpolated parameter values, * the closest imctrl

GSASII.GSASIIIntPDFtool.**MapCache = {'ThetaAzimMap': {}, 'distanceList': [], 'maskMap': {}}**

caches for TA and Mask maps

GSASII.GSASIIIntPDFtool.**ProcessImage**(*newImage*, *imgprms*, *mskprms*, *xydata*, *PDFdict*, *InterpVals*, *calcModes*, *outputModes*)

Process one image that is read from file newImage and is integrated into one or more diffraction patterns and optionally each diffraction pattern can be transformed into a pair distribution function.

Parameters

- **newImage** (*str*) – file name (full path) for input image
- **imgprms** (*dict*) – dict with some nested lists & dicts describing the image settings and integration parameters
- **mskprms** (*dict*) – dict with areas of image to be masked
- **xydata** (*dict*) – contains histogram information with about background contributions, used for PDF computation (used if ComputePDF is True)
- **PDFdict** – contains PDF parameters (used if ComputePDF is True)
- **InterpVals** – contains interpolation table (used if TableMode is True)
- **calcModes** (*tuple*) – set of values for which computations are performed and how
- **outputModes** (*tuple*) – determines which files are written and where

GSASII.GSASIIIntPDFtool.**SetupInterpolation**(*dlg*)

Creates an object for interpolating image parameters at a given distance value

23.2 G2compare: Tool for project comparison

This is intended to read in multiple GSAS-II projects and provide graphics, tables of information and so on. Not much of this has been written at present.

class GSASII.G2compare.**MakeTopWindow**(*parent*)

Define the main frame and its associated menu items

LoadPhase(*fil*)

Load Phase entries from a .GPX file to the tree. see GSASIImiscGUI.ProjFileOpen()

LoadProject(*fil*)

Load the Covariance entry from a .GPX file to the tree. see `GSASIImiscGUI.ProjFileOpen()`

LoadPwdr(*fil*)

Load PWDR entries from a .GPX file to the tree. see `GSASIImiscGUI.ProjFileOpen()`

SelectGPX()

Select a .GPX file to be read

SelectMultGPX()

Select multiple .GPX files to be read

SetModeMenu()

Create the mode-specific menu and its contents

getMode()

returns the display mode (one of "Histogram","Phase","Project"). Could return '?' in case of an error.

loadFile(*fil*)

read or reread a file

onHistFilter(*event*)

Load a filter string via a dialog in response to a menu event

onHistPrinceTest(*event*)

Compare two histograms (selected here if more than two are present) using the statistical test proposed by Ted Prince in Acta Cryst. B35 1099-1100. (1982). Also see Int. Tables Vol. C (1st Ed.) chapter 8.4, 618-621 (1995).

onLoadGPX(*event*)

Initial load of GPX file in response to a menu command

onLoadMultGPX(*event*)

Initial load of multiple GPX files in response to a menu command

onLoadWildGPX(*event*, *wildcard=None*)

Initial load of GPX file in response to a menu command

onProjFtest(*event*)

Compare two projects (selected here if more than two are present) using the statistical F-test (aka Hamilton R-factor test), see:

- Hamilton, R. W. (1965), Acta Crystallogr. 18, 502-510.
- Prince, E., Mathematical Techniques in Crystallography and Materials Science, Second ed. (Springer-Verlag, New York, 1994).

onRefresh(*event*)

reread all files, in response to a change in mode, etc.

`GSASII.G2compare.`**RC2Ftest**(*npts*, *RChiSq0*, *nvar0*, *RChiSq1*, *nvar1*)

Compute the F-test probability that a model expanded with added parameters (relaxed model) is statistically more likely than the constrained (base) model :param int npts: number of observed diffraction data points :param float RChiSq0: Reduced Chi**2 for the base model :param int nvar0: number of refined variables in the base model :param float RChiSq0: Reduced Chi**2 for the relaxed model :param int nvar1: number of refined variables in the relaxed model

GSASII.G2compare.**RwFtest**(*npts*, *Rwp0*, *nvar0*, *Rwp1*, *nvar1*)

Compute the F-test probability that a model expanded with added parameters (relaxed model) is statistically more likely than the constrained (base) model :param int npts: number of observed diffraction data points :param float Rwp0: Weighted profile R-factor or GOF for the base model :param int nvar0: number of refined variables in the base model :param float Rwp1: Weighted profile R-factor or GOF for the relaxed model :param int nvar1: number of refined variables in the relaxed model

GSASII.G2compare.**main**(*application*)

Start up the GSAS-II GUI

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

PYTHON MODULE INDEX

g

t