GSAS-II Developers Documentation

Release 0.2.0

Robert B. Von Dreele and Brian H. Toby

May 15, 2013

CONTENTS

CHAPTER

ONE

GSAS-II MAIN MODULE

Main routines for the GSAS-II program

class GSASII.**GSASII**(*parent*)
Define the main GSAS-II frame and its associated menu items

CheckNotebook()
Make sure the data tree has the minimally expected controls. (BHT) correct?

class **ConstraintDialog**(*parent*, *title*, *text*, *data*, *separator='*'*)
Window to edit Constraint values

class GSASII.**CopyDialog**(*parent*, *title*, *text*, *data*)
Creates a dialog for copying control settings between data tree items

GSASII.**ErrorDialog**(*title*, *message*, *parent=None*, *wtype=4*)
Display an error message

GSASII.**ExitMain**(*event*)
Called if the main window is closed

GSASII.**FillMainMenu**(*menubar*)
Define contents of the main GSAS-II menu for the (main) data tree window in the mac, used also for the data item windows as well.

GSASII.**GetFileList**(*fileType*, *skip=None*)
Appears unused. Note routine of same name in GSASIIpwdGUI

GSASII.**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

GSASII.**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

GSASII.**GetPhaseData**()
Returns a list of defined phases. Used only in GSASIIgrid Note routine GSASIIstruct.GetPhaseData also exists.

GSASII.**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.

GSASII.**GetUsedHistogramsAndPhasesfromTree**()
Returns all histograms that are found in any phase and any phase that uses a histogram :returns: two dicts:

•Histograms = dictionary of histograms as {name:data,...}

•Phases = dictionary of phases that use histograms

GSASII.**OnAddPhase**(*event*)
Add a new, empty phase to the tree. Called by Data/Add Phase menu

GSASII.**OnDataDelete**(*event*)
Delete one or more histograms from data tree. Called by the Data/DeleteData menu

GSASII.**OnDeletePhase**(*event*)
Delete a phase from the tree. Called by Data/Delete Phase menu

GSASII.**OnFileClose**(*event*)
Clears the data tree in response to the File/Close Project menu button. User is given option to save the project.

GSASII.**OnFileExit**(*event*)
Called in response to the File/Quit menu button

GSASII.**OnFileOpen**(*event*)
Reads in a GSAS-II .gpx project file in response to the File/Open Project menu button

GSASII.**OnFileSave**(*event*)
Save the current project in response to the File/Save Project menu button

GSASII.**OnFileSaveas**(*event*)
Save the current project in response to the File/Save as menu button

GSASII.**OnImageRead**(*event*)
Called to read in an image in any known format

GSASII.**OnImageSum**(*event*)
Sum together image data(?)

GSASII.**OnImportGeneric**(*reader*, *readerlist*, *label*, *multiple=False*)
Used to import Phases, powder dataset or single crystal datasets (structure factor tables) using reader objects subclassed from GSASIIIO.ImportPhase, GSASIIIO.ImportStructFactor or GSASIIIO.ImportPowderData. If a reader is specified, only that will be attempted, but if no reader is specified, every one that is potentially compatible (by file extension) will be tried on the selected file(s).

Parameters

- **reader** (*readerobject*) – 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 or self.ImportPowderReaderlist (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.

Returns a list of reader objects (rd_list) that were able to read the specified file(s). This list may be empty.

GSASII.**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.

GSASII.**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.

GSASII.**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.

GSASII.**OnMakePDFs**(*event*)
Calculates PDFs

GSASII.**OnPatternTreeItemActivated**(*event*)
Called when a tree item is activated

GSASII.**OnPatternTreeItemCollapsed**(*event*)
Called when a tree item is collapsed

GSASII.**OnPatternTreeItemDelete**(*event*)
Called when a tree item is deleted – not sure what this does

GSASII.**OnPatternTreeItemExpanded**(*event*)
Called when a tree item is expanded

GSASII.**OnPatternTreeKeyDown**(*event*)
Not sure what this does

GSASII.**OnPatternTreeSelChanged**(*event*)
Called when a data tree item is selected

GSASII.**OnPwdrSum**(*event*)
Sum together powder data(?)

GSASII.**OnReadPowderPeaks**(*event*)
Bound to menu Data/Read Powder Peaks – still needed?

GSASII.**OnRefine**(*event*)
Perform a refinement. Called from the Calculate/Refine menu.

GSASII.**OnRenameData**(*event*)
Renames an existing phase. Called by Data/Rename Phase menu

GSASII.**OnSeqRefine**(*event*)
Perform a sequential refinement. Called from the Calculate/Sequential refine menu.

GSASII.**OnSize**(*event*)
Called when the main window is resized. Not sure why

GSASII.**OnViewLSParms**(*event*)
Displays a window showing all parameters in the refinement. Called from the Calculate/View LS Parms menu.

GSASII.**ReadPowderInstprm**(*instfile*)
Read a GSAS-II (new) instrument parameter file

Parameters **instfile** (*str*) – name of instrument parameter file

GSASII.**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, the user is asked to make a selection.
- **rd** (*obj*) – the raw data (histogram) data object. This sets rd.instbank.

class GSASII.**SumDialog**(*parent*, *title*, *text*, *dataType*, *data*)
Allows user to supply scale factor(s) when summing data

class GSASII.**ViewParmDialog**(*parent*, *title*, *parmDict*)
Window to show all parameters in the refinement. Called from OnViewLSParms

class GSASII.**GSASIImain**(*redirect=True*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)
Defines a wxApp for GSAS-II

Creates a wx frame (self.main) which contains the display of the data tree.

OnInit()
Called automatically when the app is created.

GSASII.**main**()
Start up the GSAS-II application

CHAPTER

TWO

GSASIIMAPVARS: PARAMETER CONSTRAINTS

Module to implements algebraic contraints, parameter redefinition and parameter simplification contraints.

Parameter redefinition (new vars) is done by creating one or more relationships between a set of parameters

```
Mx1 * Px + My1 * Py +...
Mx2 * Px + Mz2 * Pz + ...
```

where Pj is a parameter name and Mjk is a constant.

Constant constraint Relations can also be supplied in the form of an equation:

```
nx1 * Px + ny1 * Py +... = C1
```

where Cn is a constant. These equations define an algebraic constrant.

Parameters can also be "fixed" (held), which prevents them from being refined.

All of the above three cases are input using routines GroupConstraints and GenerateConstraints. The input consists of a list of relationship dictionaries:

```
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},
     {'0::A0': 0.0}]
fixedList = ['5.0', None, '0']
```

Where the dictionary defines the first part of an expression and the corresponding fixedList item is either None (for parameter redefinition) or the constant value, for a constant constraint equation. A dictionary that contains a single term defines a variable that will be fixed (held). The multiplier and the fixedList value in this case are ignored.

Parameters can also be equivalenced or "slaved" to another parameter, such that one (independent) parameter is equated to several (now dependent) parameters. In algebraic form this is:

```
P0 = M1 * P1 = M2 * P2 = ...
```

Thus parameters P0, P1 and P2,... are linearly equivalent. Routine StoreEquivalence is used to specify these equivalences.

Parameter redefinition (new vars) describes a new, independent, parameter, which is defined in terms of dependent parameters that are defined in the Model, while fixed constrained relations effectively reduce the complexity of the Model by removing a degree of freedom. It is possible for a parameter to appear in both a parameter redefinition expression and a fixed constraint equation, but a parameter cannot be used a parameter equivalance cannot be used elsewhere (not fixed, constrained or redefined). Likewise a fixed parameter cannot be used elsewhere (not equivalanced, constrained or redefined).

Relationships are grouped so that a set of dependent parameters appear in only one group (done in routine GroupConstraints.) Note that if a group contains relationships/equations that involve N dependent parameters, there must exist N-C new parameters, where C is the number of contraint equations in the group. Routine GenerateConstraints takes the output from GroupConstraints and generates the "missing" relationships and saves that information in the module's global variables. Each generated parameter is named sequentially using paramPrefix.

A list of parameters that will be varied is specified as input to GenerateConstraints (varyList). A fixed parameter will simply be removed from this list preventing that parameter from being varied. Note that all parameters in a relationship must specified as varied (appear in varyList) or none can be varied. This is checked in GenerateConstraints (as well as for generated relationships in SetVaryFlags).

- If all parameters in a parameter redefinition (new var) relationship are varied, the parameter assigned to this expression (::constr:n, see paramPrefix) newly generated parameter is varied. Note that any generated "missing" relations are not varied. Only the input relations are varied.
- If all parameters in a fixed constraint equation are varied, the generated "missing" relations in the group are all varied. This provides the N-C degrees of freedom.

2.1 *External 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. One can then use `CheckConstraints()` to check that the input is internally consistent and finally `GroupConstraints()` and `GenerateConstraints()` to generate the internally used tables. Routines `Map2Dict()` is used to initialize the parameter dictionary and `Dict2Map()`, `Dict2Deriv()`, and `ComputeDepESD()` are used to apply constraints. Routine `VarRemapShow()` is used to print out the constraint information, as stored by `GenerateConstraints()`.

`InitVars()` This is optionally used to clear out all defined previously defined constraint information

`StoreEquivalence()` To implement parameter redefinition, one calls StoreEquivalence. This should be called for every set of equivalence relationships. There is no harm in using StoreEquivalence with the same independent variable:

```
StoreEquivalence('x',('y',))
StoreEquivalence('x',('z',))
```

or equivalently

```
StoreEquivalence('x',('y','z'))
```

The latter will run more efficiently. Note that mixing independent and dependent variables is problematic. This is not allowed:

```
StoreEquivalence('x',('y',))
StoreEquivalence('y',('z',))
```

Use StoreEquivalence before calling GenerateConstraints or CheckConstraints

`CheckConstraints()` To check that input in internally consistent, use CheckConstraints

`Map2Dict()` To determine values for the parameters created in this module, one calls Map2Dict. This will not apply contraints.

`Dict2Map()` To take values from the new independent parameters and constraints, one calls Dict2Map. This will apply contraints.

`Dict2Deriv()` Use Dict2Deriv to determine derivatives on independent parameters from those on dependent ones

ComputeDepESD() Use ComputeDepESD to compute uncertainties on dependent variables

VarRemapShow() To show a summary of the parameter remapping, one calls VarRemapShow

2.2 Global Variables

dependentParmList: contains a list by group of lists of parameters used in the group. Note that parameters listed in dependentParmList should not be refined as they will not affect the model

indParmList: a list of groups of Independent parameters defined in each group. This contains both parameters used in parameter redefinitions as well as names of generated new parameters.

fixedVarList: a list of variables that have been 'fixed' by defining them as equal to a constant (::var: = 0). Note that the constant value is ignored at present. These variables are later removed from varyList which prevents them from being refined. Unlikely to be used externally.

arrayList: a list by group of relationship matrices to relate parameters in dependentParmList to those in indParmList. Unlikely to be used externally.

invarrayList: a list by group of relationship matrices to relate parameters in indParmList to those in dependentParmList. Unlikely to be used externally.

fixedDict: a dictionary containing the fixed values corresponding to parameter equations. The dict key is an ascii string, but the dict value is a float. Unlikely to be used externally.

2.3 Routines

Note that parameter names in GSAS-II are strings of form `<ph>:<hst>:<nam>`

GSASIImapvars.**CheckConstraints**(*varyList*, *constrDict*, *fixedList*)

Takes a list of relationship entries comprising a group of constraints and checks for inconsistencies such as conflicts in parameter/variable definitions and or inconsistently varied parameters.

Parameters

- **varyList** (*list*) – a list of parameters names that will be varied
- **constrDict** (*dict*) – a list of dicts defining relationships/constraints (as defined in GroupConstraints())
- **fixedList** (*list*) – a list of values specifying a fixed value for each dict in constrDict. Values are either strings that can be converted to floats or None if the constraint defines a new parameter rather than a constant.

Returns

two strings:

- the first lists conflicts internal to the specified constraints
- the second lists conflicts where the varyList specifies some parameters in a constraint, but not all

If there are no errors, both strings will be empty

GSASIImapvars.**ComputeDepESD**(*covMatrix*, *varyList*, *parmDict*)

Compute uncertainties for dependent parameters from independent ones returns a dictionary containing the esd values for dependent parameters

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** (*dict*) – a dict containing derivatives for dependent parameter computed from derivDict

GSASIImapvars.**Dict2Map**(*parmDict*, *varyList*)
Applies the constraints defined using `StoreEquivalence()`, `GroupConstraints()` and `GenerateConstraints()` by changing values in a dict containing the parameters. This should be done before the parameters are used for any computations

Parameters

- **parmDict** (*dict*) – a dict containing parameter values keyed by the parameter names. This will contain updated values for both dependent and independent parameters after Dict2Map is called. It will also contain some unexpected entries of every constant value {'0':0.0} & {'1.0':1.0}, which do not cause any problems.
- **varyList** (*list*) – a list of parameters names that will be varied

GSASIImapvars.**GenerateConstraints**(*groups*, *parmlist*, *varyList*, *constrDict*, *fixedList*)
Takes a list of relationship entries comprising a group of constraints and builds the relationship lists and their inverse and stores them in global variables Also checks for internal conflicts or inconsistencies in parameter/variable definitions.

Parameters

- **groups** (*list*) – a list of grouped contraints where each constraint grouped containts a list of indices for constraint constrDict entries, created in `GroupConstraints()` (returned as 1st value)
- **parmlist** (*list*) – a list containing lists of parameter names contained in each group, created in `GroupConstraints()` (returned as 1st value)
- **varyList** (*list*) – a list of parameters names (strings of form `<ph>:<hst>:<nam>`) that will be varied
- **constrDict** (*dict*) – a list of dicts defining relationships/constraints (as defined in `GroupConstraints()`)
- **fixedList** (*list*) – a list of values specifying a fixed value for each dict in constrDict. Values are either strings that can be converted to floats, float values or None if the constraint defines a new parameter
- **constrDict** – a list of dicts defining relationships/constraints

GSASIImapvars.**GetDependentVars**()
Return a list of dependent variables: e.g. variables that are constrained in terms of other variables

Returns a list of variable names

GSASIImapvars.**GetIndependentVars**()
Return a list of independent variables: e.g. variables that are created by constraints of other variables

Returns a list of variable names

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

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 {'param1': mult1, 'param2': 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

GSASIImapvars.**InitVars**()
Initializes all constraint information

GSASIImapvars.**Map2Dict**(*parmDict*, *varyList*)
Create (or update) the Independent Parameters from the original set of Parameters

Removes dependent variables from the varyList

This should be done once, after the constraints have been defined using `StoreEquivalence()`, `GroupConstraints()` and `GenerateConstraints()` and before any variable refinement is done to complete the parameter dictionary by defining independent parameters and satisfying the constraint equations.

Parameters

- **parmDict** (*dict*) – a dict containing parameter values keyed by the parameter names. This will contain updated values for both dependent and independent parameters after Dict2Map is called. It will also contain some unexpected entries of every constant value {'0':0.0} & {'1.0':1.0}, which do not cause any problems.
- **varyList** (*list*) – a list of parameters names that will be varied

GSASIImapvars.**PrintIndependentVars**(*parmDict*, *varyList*, *sigDict*, *PrintAll=False*, *pFile=None*)
Print the values and uncertainties on the independent variables

GSASIImapvars.**StoreEquivalence**(*independentVar*, *dependentList*)
Takes a list of dependent parameter(s) and stores their relationship to a single independent parameter (independentVar)

Parameters

- **independentVar** (*str*) – name of master parameter that will be used to determine the value to set the dependent variables
- **dependentList** (*list*) – a list of parameters that will set from independentVar. Each item in the list can be a string with the parameter name or a tuple containing a name and multiplier: `['parm1',('parm2',.5),]`

GSASIImapvars.**VarRemapShow**(*varyList*)

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

CHAPTER

THREE

GSASIIPHSGUI: PHASE GUI

Module to create the GUI for display of phase information in the data display window when a phase is selected. (Items displayed by some tabs is found in other modules.)

GSASIIphsGUI.**UpdatePhaseData**(*G2frame*, *Item*, *data*, *oldPage*)

Create the data display window contents when a phase is clicked on in the man (data tree) window. Called only from GSASIIgrid.MovePatternTreeToGrid, which in turn is called from GSASII.GSASII.OnPatternTreeSelChanged when a tree item is selected.

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
- **oldPage** (*int*) – This sets a tab to select when moving from one phase to another, in which case the same tab is selected to display first. The default is 0, which brings up the General tab.

Note that this program requires the Python wxPython, NumPy, SciPy, matplotlib packages, plus the PyOpenGL package (which is installed by GSAS-II if not found).

PYTHON MODULE INDEX

g

INDEX

R

S

U

V