MTFrame Developers Guide

Release

Andrew Supka

May 24, 2016

Contents:

CHAPTER 1

plot module

plot.__**bandPlot**(*oneCalc, yLim=[-10, 10], DOSPlot='', LSDA=False, postfix='', tight_banding=False*)

Function to take the data files generated by the sumpdos ppBands functions and plots the electronic band structure and the projected density of states with energy shifted relative to the Fermi Energy.

Parameters **oneCalc** (*tuple*) – Single calculation dictionary and the ID of the calculation NEEDS TO BE CHANGED TO oneCalc,ID FORMAT SOON

Keyword Arguments

- **yLim** (*list*) – List or tuple of two integers for max and min range in horizontal axis of DOS plot
- **LSDA** (*bool*) – To plot DOS as spin polarized or not (calculation must have been done as spin polarized)
- **DOSPlot** (*str*) – DOS or the PDOS plots next to the eletronic band structure plot (assuming you ran either ppDOS or ppPDOS) (default: NONE)
- **tight_banding** (*bool*) – Whether to treat the input data as from Quantum Espresso or WanT bands.x
- **postfix** (*str*) – Postfix to the filename of the plot

Returns None

plot.__**bands**(*oneCalc, ID, yLim=[-10, 10], DOSPlot='', LSDA=False, postfix='', tight_banding=False*)

Wrapper function for MTF.plot.__bandPlot OBSOLETE. NEEDS REMOVAL

Parameters

- **oneCalc** (*dict*) – Single calculation that is the value of the dictionary of dictionaries of calculations
- **ID** (*str*) – ID of the calculation

Keyword Arguments

- **yLim** (*list*) – List or tuple of two integers for max and min range in horizontal axis of DOS plot
- **LSDA** (*bool*) – To plot DOS as spin polarized or not (calculation must have been done as spin polarized)
- **DOSPlot** (*str*) – DOS or the PDOS plots next to the eletronic band structure plot (assuming you ran either ppDOS or ppPDOS) (default: NONE)
- **postfix** (*str*) – Postfix to the filename of the plot

- **tight_banding** (*bool*) – Whether to treat the input data as from Quantum Espresso or WanT bands.x

Returns None

plot.**__combinePDOS**(*dosfiles*)

plot.**__distortionEnergy**(*calcs1*, *xaxis*, *yaxis*, *zaxis='Energy'*, *calcs=None*, *colorbarUnits=None*, *titleArray=None*, *plotTitle=None*, *xAxisStr=None*, *yAxisStr=None*, *fileName='distortionEnergy.pdf'*, *percentage=False*)

Plots the change in energy of a certain chemistry when the structure is changed

Parameters

- **calcs1** (*dict*) – calculations of calculations with which to compare calcs2 and calcs3 with
- **xaxis** (*str*) – variable in the calculations you want as yaxis
- **yaxis** (*str*) – variable in the calculations you want as yaxis

Keyword Arguments

- **calcs** (*list*) – a set of calcs that has distorted parameters from calcs1 (default:None)
- **title** (*str*) – title of plot (default:None)
- **colorbarUnits** (*str*) – the units of the elements in the array.(default:same as xaxis)
- **xaxisStr** (*str*) – The label for the horizontal axis of the plot.(default:same as yaxis)
- **yaxisStr** (*str*) – The label for the vertical axis of the plot.(default:None)
- **plotTitle (str) The title of the plot.(default** – None)
- **titleArray** (*list*) – an array for the labels for the colorbar for the sets (default:None)
- **fileName** (*str*) – name (and path where default is directory where script is run from) of the output file (default: 'distortionEnergy.pdf')

Returns None

plot.**__dosPlot**(*oneCalc, ID, yLim=[-10, 10], LSDA=False*)

Function to take the data generated for the DOS and plot them with energy shifted relative to the Fermi Energy.

Parameters

- **oneCalc** (*dict*) – Single calculation that is the value of the dictionary of dictionaries of calculations
- **ID** (*str*) – ID of the calculation

Keyword Arguments

- **yLim** (*list*) – List or tuple of two integers for max and min range in horizontal axis of DOS plot
- **LSDA** (*bool*) – To plot DOS as spin polarized or not (calculation must have been done as spin polarized)

Returns None

plot.**__getPath_WanT**(*oneCalc*, *ID*)

Arguments:

Keyword Arguments:

Returns:

plot.**__opdos**(*oneCalc*, *ID*, *yLim*, *LSDA=False*, *postfix=''*, *scale=False*)

plot.**__plotByAtom**(*maxNum, speciesNum, fig, atom, oneCalc, ID, yLim=[-10, 10], LSDA=False, ax=None*)

Function to take the data files generated by the sumpdos function and plots them with energy shifted relative to the Fermi Energy.

Parameters

- **atom** (*str*) – Species of atom you are wanting to plot ('All' will plot the combined pdos for all atoms in the system)
- **oneCalc** (*dict*) – single calculation that is the value of the dictionary of dictionaries of calculations

Keyword Arguments **yLim** (*list*) – the limits of the ordinate on the plot (default: [-10,10]

Returns **ax2** – returns an axes object with the plotted proj. DOS

Return type matplotlib.pyplot.axis

plot.**__plot_phonon**(*oneCalc*, *ID*, *postfix=''*, *THz=True*)

Function to take the data files generated by the sumpdos ppBands functions and plots the electronic band structure and the projected density of states with energy shifted relative to the Fermi Energy.

Parameters

- **oneCalc** (*dict*) – Single calculation that is the value of the dictionary of dictionaries of calculations
- **ID** (*str*) – ID of the calculation

Keyword Arguments

- **postfix** (*str*) – Output filename postfix
- **THz** (*bool*) – Plot the frequencies in THz or cm^-1

Returns None

plot.**__radialPDF**(*oneCalc*, *ID*, *atomNum*, *filterElement=None*, *title=''*, *file_prefix=''*, *file_postfix=''*, *y_range=None*, ***kwargs*)

kwargs get passed onto the hist function inside

plot.**__smoothGauss**(*list*, *strippedXs=False*, *degree=2*)

Arguments:

Keyword Arguments:

Returns:

plot.**__sumpdos**(*oneCalc*, *ID*)

Takes the output files from projwfx.x that is called in the ppDOS function and sums the projected orbitals across the files of each orbital for each atomic species and outputs the summed data in files named <species>_<orbital>.sumpdos.

Parameters

- **oneCalc** (*dict*) – dictionary of one calculation
- **ID** (*str*) – ID of the calculation

Keyword Arguments **None**

Returns None

plot.**__transport_plot**(*oneCalc*, *ID*, *nm=False*, *postfix=''*)

Arguments:

Keyword Arguments:

plot.**bands**(*calcs, yLim=[-10, 10], DOSPlot='', LSDA=False, runlocal=False, postfix='', tight_banding=False*)

Generates electronic band structure plots for the calculations in the dictionary of dictionaries of calculations with the option to have a DOS or PDOS plot to accompany it.

Parameters **calcs** (*dict*) – dictionary of dictionaries representing the set of calculations

Keyword Arguments

- **yLim** (*list*) – List or tuple of two integers for max and min range in horizontal axis of DOS plot
- **LSDA** (*bool*) – To plot DOS as spin polarized or not (calculation must have been done as spin polarized)
- **DOSPlot** (*str*) – DOS or the PDOS plots next to the eletronic band structure plot (assuming you ran either ppDOS or ppPDOS) (default: NONE)
- **postfix** (*str*) – Postfix to the filename of the plot
- **tight_banding** (*bool*) – Whether to treat the input data as from Quantum Espresso or WanT bands.x

Returns None

plot.**distortionEnergy**(*calcs1, xaxis, yaxis, zaxis='Energy', calcs=None, colorbarUnits=None, titleArray=None, plotTitle=None, xAxisStr=None, yAxisStr=None, fileName='distortionEnergy.pdf', percentage=False, runlocal=True*)

plot.**dos**(*calcs, yLim=[-10, 10], LSDA=False, runlocal=False*)

Generates DOS plots for the calculations in the dictionary of dictionaries of calculations

Parameters **calcs** (*dict*) – dictionary of dictionaries representing the set of calculations

Keyword Arguments

- **yLim** (*list*) – List or tuple of two integers for max and min range in horizontal axis of DOS plot
- **LSDA** (*bool*) – To plot DOS as spin polarized or not (calculation must have been done as spin polarized)
- **runlocal** (*bool*) – Do the plotting right now or if False do it when the calculations are running
- **postfix** (*str*) – Postfix to the filename of the plot
- **tight_banding** (*bool*) – Whether to treat the input data as from Quantum Espresso or WanT bands.x

Returns None

plot.**grid_plot**(*calcs, xaxis, yaxis, zaxis='Energy', colorbarUnits=None, zaxis_title=None, plot_title=None, xAxisStr=None, yAxisStr=None, fileName='grid_plot.pdf', runlocal=True*)

plot.**interpolatePlot**(*calcs, variable1, variable2, zaxis='Energy', xaxisTitle=None, yaxisTitle=None, zaxisTitle=None, title=None, fileName='interpolatePlot.pdf', delta=False, text_min=False, vhline_min=False, circle_min=False, delta_min=True, rel_center=False, plot_color='jet', find_min=False*)

Takes a list of calculations and plots the energy of the calculations as a function of two input variables the first value is the baseline for the energy value and the energy plotted is the difference between that energy and the other energies in the grid

Parameters

- **calcs** (*dict*) – dictionary of dictionaries of calculations
- **variable1** (*str*) – a string of the variable in the calculations that you want as your x axis
- **variable2** (*str*) – a string of the variable in the calculations that you want to your y axis

Keyword Arguements: title (str): Title of plot (default: None) zaxis (str): Choice out of the keywords in each calc to plot in the Z axis (default: Energy) xaxisTitle (str): title of xaxis (default: same as variable1) yaxisTitle (str): title of yaxis (default: same as variable2) zaxisTitle (str): title of zaxis (default: same as zaxis) fileName (str): Name (and path where default is directory where script is run from) of the

output file (default: 'interpolatePlot.pdf')

delta (bool): Z-axis scale relative to its first value delta_min (bool): Z-axis scale relative to its lowest value text_min (bool): Display text of minimum value next to the minimum value if find_min=True vhline_min (bool): Display text of minimum value next to the minimum value if find_min=True circle_min (bool): Display text of minimum value next to the minimum value if find_min=True delta_min (bool): Display text of minimum value next to the minimum value if find_min=True rel_center (bool): Display text of minimum value next to the minimum value if find_min=True plot_color (str): string of the matplotlib colormap to be used find_min (bool): Interpolate between points and find value of

variable1 and variable2 for the minimum value of Z axis

Returns None

plot.**interpolatePlot1D**(*calcs*, *variable1*, *yaxis='Energy'*, *xaxisTitle=None*, *yaxisTitle=None*, *title=None*, *fileName='interpolatePlot.pdf'*, *delta=False*, *circle_min=False*)

Takes a list of calculations and plots the energy of the calculations as a function of two input variables the first value is the baseline for the energy value and the energy plotted is the difference between that energy and the other energies in the grid

Parameters

- **calcs** (*dict*) – dictionary of dictionaries of calculations
- **variable1** (*dict*) – a string of the variable in the calculations that you want as your x axis

Keyword Arguements: yaxis (str): Choice out of the keywords in each calc to plot in the Z axis (default: Energy) xaxisTitle (str): title of xaxis (default: same as variable1) yaxisTitle (str): title of yaxis (default: same as yaxis) title (str): Title of plot (default: None) fileName (str): Name (and path where default is directory where script is run from) of the

output file (default: 'interpolatePlot.pdf')

delta (bool): Z-axis scale relative to its first value circle_min (bool): Display text of minimum value next to the minimum value

plot.**opdos**(*calcs, yLim=[-10, 10], LSDA=False, runlocal=False, postfix='', scale=False*)

Generates electronic band structure plots for the calculations in the dictionary of dictionaries of calculations with the option to have a DOS or PDOS plot to accompany it.

Parameters **calcs** (*dict*) – dictionary of dictionaries representing the set of calculations

Keyword Arguments

- **yLim** (*list*) – List or tuple of two integers for max and min range in horizontal axis of DOS plot
- **LSDA** (*bool*) – To plot DOS as spin polarized or not (calculation must have been done as spin polarized)
- **runlocal** (*bool*) – Do the plotting right now or if False do it when the calculations are running

- **postfix** (*str*) – Postfix to the filename of the plot
- **tight_banding** (*bool*) – Whether to treat the input data as from Quantum Espresso or WanT bands.x

Returns None

plot.**phonon**(*calcs*, *runlocal=False*, *postfix=''*, *THz=True*)

plot.**radialPDF**(*calcs*, *atomNum*, *filterElement=None*, *runlocal=False*, *title=''*, ***kwargs*)
kwargs get passed to pyplot.hist

plot.**read_transport_datafile**(*ep_data_file*, *mult_x=1.0*, *mult_y=1.0*)
Arguments:

Keyword Arguments:

Returns:

plot.**transport_plots**(*calcs*, *runlocal=False*, *postfix=''*)

CHAPTER 2

prep module

prep.**ConfigSectionMap**(*section*, *option*, *configFile=None*)

prep.**__ConfigSectionMap**(*section*, *option*, *configFile=None*)

prep.**__addToAll**(*calcs*, *block=None*, *addition=None*)
Adds text to a particular command block in the _ID.py for all calculations in the set.

Parameters **calcs** (*dict*) – a dictionary of dicionaries representing the set of calculations

Keyword Arguments

- **block** (*str*) – the name of the command block that text will be added to
- **addition** (*str*) – the text to be added to the block

Returns None

prep.**__addToBlock**(*oneCalc*, *ID*, *block*, *addition*)
Writes the code to a single calculation's _ID.py

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step
- **block** (*str*) – string of the block in the _ID.py that the addition is to be added to for that step of workflow
- **addition** (*str*) – a string containing code to be written to the specific block in the _ID.py for each calculation

Returns None

prep.**__announcePrint**(*string*)
A debugging tool used to accentuate a string os it can be easily picked out from stdout

Parameters **string** (*str*) – a string

Returns None

prep.**__calcsFromCalcList**(*calcList*)
Takes a list of dictionaries representing a calculation and turns the list of dictionaries into a dictionary of dictionaries

Parameters **calcList** (*list*) – a list of dictionaries

Returns A dictionary of dictionaries

prep.**__checkSuccessCompletion**(*oneCalc*, *ID*, *faultTolerant=True*)
Allows one of the calculations to check the status of the other calculations and returns True if all calculations in the set have completed.

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments **faultTolerant** (*bool*) – a flag to choose if we return True if some of the calculations ran but did not complete successfully

Returns True if all calculations in the set are complete and False if not.

prep.**__check_lock**(*func*, *oneCalc*, *ID*, **args*, ***kwargs*)
A function that is wrapped on another function to check if the script has already run through to prevent certain functions in the <ID>.py from running again after a restart or a loop like scfuj.

Parameters

- **func** (*func*) – needed for the wrapper
- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step
- ***args** – additional arguments

Keyword Arguments ****kwargs** – additional keyword arguments

Returns Returns either False or the calculation dictionary and its ID hash

prep.**__check_restart**(*oneCalc*, *ID*)
Not used. Delete possibly

prep.**__cleanCalcs**(*ID*, *oneCalc*)
Removes all files from the directory tree with a "_" prefix.

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Returns None

prep.**__cleanInputStringSCF**(*inputString*)
Sanitizes QE input files by removing whitespace, commented out text, or any other unnecessary characters.

Parameters **inputString** (*str*) – a string of a QE input file

Returns a string of the sanitized QE input file

prep.**__copyConfig**(*config*, *dest*)
Copies the config file used to the MTF directory when MTFrame is initiated and resolves relative paths in the config file.

Parameters **config** (*str*) – location of the config file to be copied to "MTF" directory

prep.**__crc64digest**(*aString*)
Generates the CRC64 hash checksum of the inputted string.

From W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, "Numerical recipes in C

Parameters **aString** (*str*) – a string

Returns **hashed_str** – a 16 character long CRC64 hash of the inputted string

Return type str

prep.**__fillTemplate**(*oneCalc*, *ID*)
Fills in the blocks of each calculation's _ID.py with mechanism for starting the next step in the chain

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Returns None

prep.**__findInBlock**(*oneCalc*, *ID*, *block=None*, *string=''*)
Looks for a string via a regular expression inside one of the command blocks in the _ID.py. Used to check before writing something as to avoid unintentionally having it written it twice.

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments

- **block** (*str*) – a string of the name of the command block in the _ID.py to look in.
- **string** (*str*) – the string to search for which can be in plain text or as a regex.

Returns True if the regex finds the pattern or False if it does not

prep.**__forceGlobalConfigFile**(*configFile*)

prep.**__forceSubmitNodeIP**(*nodeName*)
Creates the global variable __submitNodeName__ and gives it the value of nodeName

Parameters **nodeName** (*str*) – a string containing the name of the submit node

Returns None

prep.**__freezeAtoms**(*oneCalc*, *ID*)
Modifies a QE input file to have all the atom movement flags in the ATOMIC_POSITIONS card be set to 0 which means they cannot move during a ionic or variable cell relax calculation.

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Returns oneCalc object representing the calculation but with the flags for atom movement all set to 0.

prep.**__from_local_scratch**(*oneCalc, ID, ext_list=['save', 'wfc', 'hub', 'mix', 'occup', 'update', 'bfgs', 'restart', 'restart_k', 'restart_scf', 'ewfcp', 'ewfc', 'ewfcm']*)

prep.**__getAMass**(*atom*)
Get the atomic mass for the specific atomic species in input

Parameters **atom** (*str*) – Atomic Species you want the mass of

Returns Mass of the atom's species

prep.**__getConfigFile**()

prep.**__getLoglevel**()
Checks config file for loggings level. If none specified it defaults to logging.INFO

Parameters **None** –

Returns

loglevel – loglevel object associated with the logging

level string found in the config file

Return type logging.loglevel

prep.**__getNextOneCalcVarName**(*oneCalc*, *ID*)

Gives increased the ID of one calculation an increase of one in its postfix

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Returns the identical calculation object that was inputted and its ID that has had its ID modified

prep.**__getPseudofilename**(*atom*, *pseudodir='./PSEUDOs'*)

Gets the pseudopotential filename for the specific atomic species in input. It will check in the MTFrame config file used when itiating the session for a pseudodir path.

Species names must be in the form of the element's symbol possibly followed by integers (i.e) "O1","Co6", or "Sn165" (Note that QE needs to be modified to accept species labels longer than 3 characters. Instructions on how are in the engine_mods directory.)

Parameters **atom** (*str*) – a string designating the atomic species you want the pseudofile name for

Keyword Arguments **pseudodir** (*str*) – the path of the directory containing pseudofiles

Returns

pseudofilename – name of the pseudopotential file in for

that species in the pseudodir specified

Return type str

prep.**__get_step**(*oneCalc*, *ID*, *step_type=None*, *last=True*)

prep.**__get_tempdir**()

prep.**__hash64String**(*hashString*)

Takes a string and returns a 16 character long CRC64 hash checksum of it.

Parameters **hashString** (*str*) – the string you want a hash checksum of

Returns **hashed_str** – a 16 character long CRC64 hash of the inputted string

Return type str

prep.**__incrementFileValue**(*oneCalc*, *ID*, *varName='uValue'*)

adds +1 to the value of a variable defined in the _ID.py files

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments **varName** (*str*) – a string of the name of the variable that is to be incremented

Returns None

prep.**__loadCalcsFromOneCalc**(*oneCalc*, *ID*)

Loads the entire calc set from one of the calculations at runtime. Used when all jobs in a set are needed to perform a task (i.e plotting data from all calculations in the set)

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Returns The dictionary of dictionaries representing the calculation set that oneCalc is a part of

prep.**__loadOneCalc**(*folder*, *ID*)

prep.**__lock_transform**(*oneCalc*, *ID*)

To be used with MTF.prep.newstepWrapper to stop a function from being run in the _<ID>.py python scripts generated by MTFrame to prohibit the wrapped function from running if it has already run.

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Returns **oneCalc** – a dictionary containing properties about the MTFrame calculation ID (str): ID string for the particular calculation and step

Return type dict

prep.**__modifyInputPrefixPW**(*oneCalc*, *ID*)

Assigns the new prefix to a the calculation inputs in the form _ID

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – The ID string for the particular calculation and the value of the the prefix without the leading "_" in front.

Returns None

prep.**__modifyNamelistPW**(*oneCalc*, *ID*, *namelist*, *parameter*, *value*)

A Wrapper function that is used to write the function MTF.prep.__modifyNameListPW to the _ID.py. If the value is intended to be a string in the QE input file, it must be dually quoted i.e. value="'scf"' will become 'scf' in the input file. If the value is equal to None (without quotes) it will remove that parameter from the namelist.

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step
- **namelist** (*str*) – a string of the fortran namelist that the parameter is in
- **parameter** (*str*) – a string of the parameter name
- **value** – the value of that parameter

Returns oneCalc object representing the calculation but with the paramater's value changed or added in oneCalc['_MTF_INPUT_'] and the same ID as the input to the function.

prep.**__modifyVarVal**(*oneCalc*, *ID*, *varName='uValue'*, *value=None*)

Modifies the value of a variable defined in the _ID.py files

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments

- **varName** (*str*) – a string of the name of the variable that is to be changed
- **value** – the value to change it to in the _ID.py

Returns None

prep.**__null__**(**args*, ***kwargs*)

prep.**__num_bands**(*oneCalc*)
attempt to find the number of kohn-sham orbitals after the scf calculation to find the value of the "nbnd" parameter in the "&system" namelist of QE input files.

Parameters **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation

Returns The value for the nbnd parameter in QE nscf calculations

prep.**__oneBands**(**args*, ***kwargs*)

prep.**__oneChangeCalcs**(**args*, ***kwargs*)

prep.**__oneDoss**(**args*, ***kwargs*)

prep.**__oneUpdateStructs**(**args*, ***kwargs*)

prep.**__one_phon**(*oneCalc*, *ID*, *subset_name='SUBSET'*)

prep.**__one_test_build**(*oneCalc*, *ID*, *build_command*, *subset_name='SUBSET'*, *merge_oneCalc=True*, *keep_name=False*, *config=None*)

prep.**__parseRef**(*refFile*, *dictFlag=False*)
Parses the input reference file and extract all of the information it then organizes it in a standard way and is used to create a unique hash checksum for the fortran namelist input. This is so spacing and order of namelist input data will not affect the checksum of each unique calculation input file

Parameters **refFile** (*str*) – a string of the input reference file

Keyword Arguments **dictFlag** (*bool*) – whether you want to have a string of the cleaned up text or a tokenized dictionary of it.

Returns Dictionary or String

prep.**__passGlobalVar**(*varname*, *value*)
Used to define the value of a global variable in this module's global namesapce usually from another module.

Parameters

- **varname** (*str*) – name of the global variable
- **value** – value of the global variable

Returns None

prep.**__removeComments**(*inputString*)
Removes any text from a string that follows either a # or ! on to the end of that line.

Parameters **inputString** (*str*) – input string with ! or # as comment characters

Returns the same string with the comments removed

prep.**__removeFromBlock**(*oneCalc*, *ID*, *block*, *removal*)
Searches for a regular expression pattern inside a specific block of a each calculation's _ID.py and removes the command from the block.

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

- **block** (*str*) – string of the block in the _ID.py that the addition is to be added to for that step of workflow
- **removal** (*str*) – a regular expression to find and delete a pattern of text in a single block of each calculation's _ID.py for a given step

Returns None

prep.**__resolveEqualities**(*inputString*)

Takes an input string that may or may not have text patterns of the form:

===EQN===

where EQN is some syntactically correct simple equations like 5+3. _MTFrame keywords can be used in the reference input. An example of this could be in to generate a set of calculations used to make an energy map of one layer of a layered material sliding across the other one might make their QE ATOMIC_POSITIONS card in their reference input like so:

ATOMIC_POSTIONS {crystal} _MTF_A_ ===0.00+_MTF_XSHIFT_=== ===0.00+_MTF_YSHIFT_=== 0.00 _MTF_A_ ===0.50+_MTF_XSHIFT_=== ===0.50+_MTF_YSHIFT_=== 0.00 _MTF_A_ ===0.00+_MTF_XSHIFT_=== ===0.00+_MTF_YSHIFT_=== 0.00 _MTF_A_ ===0.50+_MTF_XSHIFT_=== ===0.50+_MTF_YSHIFT_=== 0.00 _MTF_A_ 0.00 0.00 0.75 _MTF_A_ 0.00 0.50 0.75 _MTF_A_ 0.50 0.00 0.75 _MTF_A_ 0.50 0.50 0.75

With their user script containing a range of values for _MTF_XSHIFT_ and _MTF_YSHIFT_.

Parameters **inputString** (*str*) – a string of text

Returns a string of text that has the equations (i.e. ===EQN===) replaced with values

prep.**__return_ID**(*oneCalc*, *ID*, *step_type=None*, *last=True*, *straight=False*)

prep.**__saveOneCalc**(*oneCalc*, *ID*)

prep.**__scp_wrapper**(*from_path*, *to_path*, *node=''*)

prep.**__setup_local_scratch**(*oneCalc*, *ID*)

prep.**__temp_executable**(*oneCalc*, *ID*, *from_step=0*)

Creates the _ID.py for a single calculation for a given step.

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Argument ext (str): an optional postfix to the ID's to all calculations in the set for a given step

Returns The one calculation for the new set, its ID, the old calculation, and its ID

prep.**__to_local_scratch**(*oneCalc, ID, ext_list=['save', 'wfc', 'hub', 'mix', 'occup', 'update', 'bgfs', 'restart', 'restart_k', 'restart_scf', 'ewfcp', 'ewfc', 'ewfcm']*)

prep.**__transformInput**(*inputString*)

Standardizes the input format of the QE input files so MTFrame only has to account for one standard input type for atomic positions and lattice parameters.

Parameters **inputString** (*str*) – a string of a QE input file

Returns **inputString** – a string of a QE input file

Return type str

prep.**__transformParamsInput**(*inputString*)
Transforms the various styles of defining the lattice vectors into celldm form so MTFrame has a standard format to work with.

Parameters **inputString** (*str*) – a string of a QE input

Returns a string of a QE input that has its lattice vectors in celldm style

prep.**__transformPositionsInput**(*inputString*)
Transforms the various styles of defining the atomic positions into crystal fractional coordinates form so MTFrame has a standard format to work with.

Parameters **inputString** (*str*) – a string of a QE input

Returns **inputString** – a string of a QE input that has its atomic positions in crystal fractional coordinates

Return type str

prep.**__unfreezeAtoms**(*oneCalc*, *ID*)
Modifies a QE input file to have all the atom movement flags in the ATOMIC_POSITIONS card be set to 1 which means they are allowed to move during a ionic or variable cell relax calculation which is default in QE.

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Returns **oneCalc** – object representing the calculation but with the flags for atom movement all set to 1.

Return type dict

prep.**__updatecalclogs**(*calcs*, *inc=True*)
Save the log for aflowkeys

Parameters

- **aflowkeys** (*dict*) – keys from aflow
- **calcs** – Dictionary of dictionaries of calculations

prep.**__updatelogs**(*ID*, *oneCalc*, *filename*)

prep.**__writeLoggingBlock**(*oneCalc*, *ID*, *logfile*)
Writes the mechism to start the logging at runtime in the _ID.py

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step
- **logfile** (*str*) – string of the path of the logfile within the directory tree for that set of calculations

Returns None

prep.**__writeTemplate**(*finp*)
Writes the blocks of the skeleton _ID.py to file for one calculation.

Parameters **finp** (*str*) – filepath for skeleton _ID.py file to go

Returns None

prep.**__writeToScript**(*executable*, *oneCalc*, *ID*, *from_step=0*)
Generates calls on several functions to set up everything that is needed for a new step in the workflow. MTF.prep.writeToScript loops over the calculation set and calls this function to do the work on each calculation in the set.

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step
- **executable** (*str*) – <DEFUNCT OPTION: HERE FOR LEGACY SUPPORT>
- ***args** – <DEFUNCT OPTION: HERE FOR LEGACY SUPPORT>

Keyword Arguments ****kwargs** – <DEFUNCT OPTION: HERE FOR LEGACY SUPPORT>

Returns A single calculation dictionary for a new step and its ID.

prep.**__write_scratch_meta_data_file**(*oneCalc*, *ID*)
Not used. Delete possibly

prep.**addToBlockWrapper**(*oneCalc*, *ID*, *block*, *addition*)
Wraps MTF.prep.__addToBlock for use inside _calcs_container methods

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step
- **block** (*str*) – string of the block in the _ID.py that the addition is to be added to for that step of workflow
- **addition** (*str*) – a string containing code to be written to the specific block in the _ID.py for each calculation

Returns None

prep.**askMTFVars**(*refMTFVars*)
Cycle on the keys of the refMTFVars dictionary and ask to define them

Parameters **refMTFVars** (*dict*) – the variables in the ref files that you need to input to run the calculation

Returns None

prep.**bands**(*calcs*, *dk=None*, *nk=None*)
Wrapper function to write the function MTF.prep.__oneBands to the _ID.py

Parameters **calcs** (*dict*) – a dictionary of dicionaries representing the set of calculations

Keyword Arguments

- **dk** (*float*) – distance between points for Electronic Band Structure calculation
- **nk** (*int*) – approximate number of k points to be calculated along the path

Returns The identical "calcs" input variable

prep.**bandsAflow**(*dk*, *LAT*)
Query aflow for band structure path and generate the path for band structure calculation

Parameters

- **dk** (*float*) – distance between k points along path in Brillouin Zone
- **LAT** (*int*) – bravais lattice number from Quantum Espresso convention

Keyword Arguments None

Returns **info** – some information nks (str): number of k points in path stringk (str): kpoint path string

Return type str

prep.**build_calcs**(*PARAM_VARS*, *build_type='product'*)

prep.**calcFromFile**(*aflowkeys*, *fileList*, *reffile=None*, *pseudodir=None*, *build=True*, *workdir=None*, *keep_name=False*)

Reads in a string of an QE input file path, a string of an QE input, a file object of a QE input or a list of them and attempts to fill create a calculation from them. If they are missing things such as k_points card, they are automtically generated.

Parameters

- **aflowkeys** (*dict*) – a dictionary generated by MTF.prep.init
- **fileList** (*list*) – a string of an QE input file path, a string of an QE input, a file object of a QE input or a list of them

Keyword Arguments

- **reffile** (*str*) – a partially filled QE input file used in case the input(s) in fileList are missing. i.e. wfc cutoff. If the names of the Pseudopotential files are not included in the input(s) in fileList, they are chosen depending on the pseudodir chosen and included when the calculation set is formed.
- **workdir** (*str*) – a string of the workdir path that be used to override what is in the config file used when initating the MTFrame session
- **pseudodir** (*str*) – a string of the pseudodir path that be used to override what is in the config file used when initating the MTFrame session
- **build** (*bool*) – <DEFUNCT OPTION. NEEDS REMOVAL>

class prep.**calcs_container**(*dictionary*)

__delitem__(*index*)

__getitem__(*index*)

__init__(*dictionary*)

__len__()

__module__ = 'prep'

__repr__()

__str__()

_calcs_container__addToInit(*block=None*, *addition=None*)

_calcs_container__getInitInputs()

_calcs_container__saveOneCalc(*oneCalc*, *ID*)

addToAll(*block=None*, *addition=None*)

bands(*dk=None*, *nk=100*)

Wrapper method to write call MTF.prep.bands for calculating the Electronic Band Structure.

Parameters **calcs** (*dict*) – a dictionary of dicionaries representing the set of calculations

Keyword Arguments

- **dk** (*float*) – the density in the Brillouin zone of the k point sampling along the entirety of the path between high symmetry points.
- **nk** (*int*) – the approximate number of sampling points in the Brillouin Zone along the entirety of the path between high symmetry points. Points are chosen so that they are equidistant along the entirety of the path. The actual number of points will be slightly different than the inputted value of nk. nk!=None will override any value for dk.

Returns None

change_input(*namelist=None*, *parameter=None*, *value=None*)

change_pseudos(*directory*)

conventional_cell_input()

crawl_min(*mult_jobs=False*, *grid_density=10*, *initial_variance=0.02*, *thresh=0.01*, *constraint=None*, *final_minimization='relax'*)

Wrapper method to call MTF.pseudo.crawlingMinimization in the high level user interface. Adds a new step to the workflow.

Parameters **self** – the _calcs_container object

Keyword Arguments

- **mult_jobs** (*bool*) – if True split the individual scf jobs into separate cluster jobs if False run them serially
- **grid_density** (*int*) – controls the number of calculations to generate for the minimization num_{calcs}=grid_density^{num_{parameters}-num_{constraints}}
- **initial_variance** (*float*) – amount to vary the values of the parameters from the initial value. i.e. (0.02 = +/-2% variance)
- **thresh** (*float*) – threshold for $DeltaX$ of the lattice parameters between brute force minimization iterations.
- **constraint** (*list*) – a list or tuple containing two entry long list or tuples with the first being the constraint type and the second the free parameter in params that its constraining for example in a orthorhombic cell: constraint=(["volume",'c'],) allows for A and B to move freely but C is such that it keeps the cell volume the same in all calculations generated by the input oneCalc calculation.
- **final_minimization** (*str*) – calculation to be run at the end of the brute force minimization options include "scf", "relax", and "vcrelax"

Returns None

dos(*kpFactor=2*, *project=True*)

Wrapper method to call MTF.prep.doss in the high level user interface. Adds a new step to the workflow.

Parameters **self** – the _calcs_container object

Keyword Arguments

- **kpFactor** (*float*) – factor to which the k-point grid is made denser in each direction
- **project** (*bool*) – if True: do the projected DOS after completing the DOS

Returns None

evCurve_min(*pThresh=25*, *final_minimization='relax'*)

get_initial_inputs()

increase_step(*func*)

items()

iteritems()

keys()

new_step(*update_positions=True*, *update_structure=True*, *new_job=True*, *ext=''*)

phonon(*nrx1=2*, *nrx2=2*, *nrx3=2*, *innx=2*, *de=0.01*, *mult_jobs=False*, *raman=False*, *LOTO=False*, *disp_sym=True*, *atom_sym=True*)

pseudo_test_brute(*ecutwfc*, *dual=[]*, *sampling=[]*, *conv_thresh=0.01*, *constraint=None*, *initial_relax=None*, *min_thresh=0.01*, *initial_variance=0.05*, *grid_density=7*, *mult_jobs=False*, *options=None*)

relax()

resubmit(*reset=True*)

scf()

scfuj(*thresh=0.1*, *nIters=20*, *paodir=None*, *relax='scf'*, *mixing=0.0*)

Wrapper method to call MTF.scfuj.scfPrep and MTF.scfuj.run in the high level user interface. Adds a new step to the workflow.

Parameters **self** – the _calcs_container object

Keyword Arguments

- **thresh** (*float*) – threshold for self consistent hubbard U convergence
- **niters** (*int*) – max number of iterations of the acbn0 cycle
- **paodir** (*string*) – the path of the PAO directory. This will override an entry of the paodir in the MTFrame config file used for the session
- **mixing** (*float*) – the amount of the previous acbn0 U iteration to mix into the current (only needed when there is U val oscillation)

Returns None

split_chain(*update_positions=True*, *update_structure=True*)

submit()

transport(*temperature=[300], epsilon=True, run_bands=True*)

Wrapper method to call MTF.scfuj.prep_transport and MTF.scfuj.run_transport in the high level user interface. Adds a new step to the workflow.

Parameters **self** – the _calcs_container object

Keyword Arguments

- **epsilon** (*bool*) – if True episilon tensor will be computed
- **temperature** (*list*) – list of temperature(s) at which to calculate transport properties

Returns None

values()

vcrelax()

prep.**changeCalcs**(*calcs*, *keyword='calculation'*, *value='scf'*)

A Wrapper function that writes the function MTF.prep.__changeCalcs to the __ID.py

Parameters **calcs** (*dict*) – a dictionary of dicionaries representing the set of calculations

Keyword Arguments

- **keyword** (*str*) – a string which signifies the type of change that is to be made
- **value** – the value of the choice.

Returns The identical set of calculations as the input to this function

prep.**cleanCalcs**(*calcs*, *runlocal=False*)
Wrapper function for MTF.prep.__cleanCalcs

Parameters **calcs** (*dict*) – a dictionary of dicionaries representing the set of calculations

Keyword Arguments **runlocal** (*bool*) – a flag to choose whether or not to run the wrapped function now or write it to the _ID.py to run during the workflow

Returns None

prep.**construct_and_run**(*__submitNodeName__*, *oneCalc*, *ID*, *build_command=''*, *subset_tasks=[]*, *fault_tolerant=False*, *mult_jobs=True*, *subset_name='SUBSET'*, *keep_file_names=False*)
this is a check to see if we're restarting when mult_jobs==True

prep.**doss**(*calcs*, *kpFactor=1.5*)
Wrapper function to write the functio n MTF.prep.__oneDoss to the _ID.py

Parameters **calcs** (*dict*) – a dictionary of dicionaries representing the set of calculations

Keyword Arguments **kpFactor** (*float*) – the factor to which we make each direction in the kpoint grid denser

Returns The identical "calcs" input variable

prep.**extractvars**(*refFile*)
Read refFile and return an empty dictionary with all the keys and None values

Parameters **refFile** (*str*) – filename of the reference input file for the frame

Returns A dictionary containing the keyword extracted from the reference input file as keys with None for values..

prep.**generateAnotherCalc**(*old*, *new*, *calcs*)
Modify the calculation in each subdir and update the master dictionary

Parameters

- **old** (*str*) – string to replace
- **new** (*str*) – replacement string
- **calcs** (*dict*) – dictionary of dictionaries of calculations

Returns A new set of calculations with a new ID of the hash of the new input strings

prep.**getMPGrid**(*primLatVec*, *offset=True*, *string=True*)

class prep.**init**(*PROJECT*, *SET=''*, *AUTHOR=''*, *CORRESPONDING=''*, *SPONSOR=''*, *config=''*, *workdir=None*)

__delitem__(*index*)

__getitem__(*index*)

__init__(*PROJECT*, *SET=''*, *AUTHOR=''*, *CORRESPONDING=''*, *SPONSOR=''*, *config=''*, *workdir=None*)

__len__()

__module__ = 'prep'

__repr__()

__str__()

from_file(*fileList*, *reffile=None*, *pseudodir=None*, *build=True*, *workdir=None*)

Reads in a string of an QE input file path, a string of an QE input, a file object of a QE input or a list of them and attempts to fill create a calculation from them. If they are missing things such as k_points card, they are automtically generated.

Parameters

- **aflowkeys** (*dict*) – a dictionary generated by MTF.prep.init
- **fileList** (*str*) – a string of an QE input file path, a string of an QE input, a file object of a QE input or a list of them

Keyword Arguments

- **reffile** (*str*) – a partially filled QE input file used in case the input(s) in fileList are missing. i.e. wfc cutoff. If the names of the Pseudopotential files are not included in the input(s) in fileList, they are chosen depending on the pseudodir chosen and included when the calculation set is formed.
- **workdir** (*str*) – a string of the workdir path that be used to override what is in the config file used when initating the MTFrame session
- **pseudodir** (*str*) – a string of the pseudodir path that be used to override what is in the config file used when initating the MTFrame session
- **build** (*bool*) – <DEFUNCT OPTION. NEEDS REMOVAL>

items()

iteritems()

keys()

load(*step=1*)

Loads the calc logs from a given step

Parameters **step** (*int*) – the step of the calculation for whose calclogs are to be loaded

Returns **calcs** – the loaded calc logs

Return type dict

scfs(*allMTFVars*, *refFile*, *name='first'*, *pseudodir=None*, *build_type='product'*, *build=True*, *run=True*)

A wrapper method to call MTF.prep.scfs to form the calculation set. This will also create directory within the set directory for every calculation in the set.

Parameters

- **allMTFVars** (*dict*) – a dictionary whose keys correspond to the keywords in the reference input file and whose values will be used to construct the set of calculations
- **refFile** (*str*) – a filename as a string, a file object, or a string of the file that contains keywords to construct the inputs to the different calculations in the set

Keyword Arguments

- **pseudodir** (*str*) – path of the directory that contains your Pseudopotential files The value in the MTFrame config file used will override this.
- **build_type** (*str*) – how to construct the calculation set from allMTFVars dictionary:

  zip | The first calculation takes the first entry from the list of

each of the keywords. The second calculation takes the second and so on. The keywords for all lists in allMTFVars must be the same length for this method.

product | Calculation set is formed via a "cartesian product" with

the values the list of each keyword combined. (i.e if allMTFVars has one keyword with a list of 5 entires and another with 4 and a third with 10, there would be 2000 calculations in the set formed from them via product mode.

- **build** (*bool*) – <DEFUNCT OPTION. NEEDS REMOVAL>
- **run** (*bool*) – <DEFUNCT OPTION. NEEDS REMOVAL>

Returns A dictionary of dictionaries containing the set of calculations.

status (*status={}*, *step=0*, *negate_status=False*)
Loads the calc logs from a given step

Parameters **step** (*int*) – The step of the calculation for whose calclogs are to be loaded. If no step is specified then it will default to load calculations from all steps with the chosen status.

Keyword Arguments

- **status** (*dict*) – key,value pairs for status type and their value to filter on. i.e. status={'Finished':False}
- **negate_status** (*bool*) – filter on the opposite of the status filters

Returns **calcs** – the loaded calcs for one or more steps with the given status

Return type dict

values()

prep.**init__**(*PROJECT*, *SET=''*, *AUTHOR=''*, *CORRESPONDING=''*, *SPONSOR=''*, *config=''*, *workdir=None*)
Initializes the frame

Parameters

- **PROJECT** (*str*) – Name of project
- **SET** (*str*) – Name of set
- **author** (*str*) – Name of author
- **CORRESPONDING** (*str*) – Name of corresponding
- **SPONSOR** (*str*) – Name of sponsor

e.g. initFrame('LNTYPE','', 'MF', 'marco.fornari@cmich.edu','DOD-MURI'). Return the AFLOKEYS dictionary.

prep.**line_prepender**(*filename*, *new_text*)
prepends a file with a new line containing the contents of the string new_text.

Parameters

- **filename** (*str*) – string of the filename that is to be prepended
- **new_text** (*str*) – a string that is one line long to be prepended to the file

Returns None

prep.**loadlogs**(*PROJECT=''*, *SET=''*, *logname=''*, *config=None*)

prep.**lockAtomMovement**(*calcs*)
A Wrapper function that writes the function MTF.prep.__freezeAtoms to the __ID.py

Parameters **calcs** (*dict*) – a dictionary of dicionaries representing the set of calculations

Returns None

prep.**maketree**(*calcs*, *pseudodir=None*, *build=True*, *workdir=None*)
Make the directoy tree and place in the input file there

Parameters **calcs** (*dict*) –

- Dictionary of dictionaries of calculations

Keyword Arguments

- **pseudodir** (*str*) – path of pseudopotential files directory
- **workdir** (*str*) – a string of the workdir path that be used to override what is in the config file used when initating the MTFrame session
- **build** (*bool*) – <DEFUNCT OPTION. NEEDS REMOVAL>

Returns None

prep.**modifyCalcs**(*old*, *new*, *calcs*)
Modify the calculation in each subdir and update the master dictionary

Parameters

- **old** (*str*) – string to replace
- **new** (*str*) – replacement string
- **calcs** (*dict*) – dictionary of dictionaries of calculations

prep.**modifyInputPrefixPW**(*calcs*, *pre*)
A Wrapper function that is used to write a function to the __ID.py

Parameters **calcs** (*dict*) – a dictionary of dicionaries representing the set of calculations

Returns None

prep.**modifyNamelistPW**(*calcs*, *namelist*, *parameter*, *value*, *runlocal=False*)
A Wrapper function that is used to write the function MTF.prep.__modifyNameListPW to the _ID.py. If the value is intended to be a string in the QE input file, it must be dually quoted i.e. value="'scf"' will become 'scf' in the input file.

Parameters

- **calcs** (*dict*) – a dictionary of dicionaries representing the set of calculations
- **namelist** (*str*) – a string of the fortran namelist that the parameter is in
- **parameter** (*str*) – a string of the parameter name
- **value** – the value of that parameter

Keyword Arguments **runlocal** (*bool*) – a flag to choose whether or not to run the wrapped function now or write it to the _ID.py to run during the workflow.

Returns Either the identical set of calculations if runlocal == False or the set of calculations with the parameter's value changed in their oneCalc['_MTF_INPUT_'] if runlocal==True

prep.**newstepWrapper**(*pre*)

A function that wraps another function that is to be run before the a certain function runs. Its use must be in the form:

@newstepwrapper(func) def being_wrapped(oneCalc,ID,*args,**kwargs)

where func is the function that is to be run before and being_wrapped is the function being wrapped. oneCalc and ID must be the first two arguments in the function being wrapped. additional arguments and keyword arguments can follow.

Parameters **pre** (*func*) – function object that is to be wrapped before another function runs

Returns the returned values of function being wrapped or the execution of the function is skipped entirely.

class prep.**plotter**(*calcs*)

Class for adding common plotting functions from MTF.plot module to the high level user interface.

__init__(*calcs*)

__module__ = 'prep'

bands(*yLim=[-10, 10], DOSPlot='', LSDA=False, runlocal=False, postfix=''*)

Wrapper method to call MTF.plot.bands in the high level user interface.

Parameters **self** – the plotter object

Keyword Arguments

- **yLim** (*list*) – a tuple or list of the range of energy around the fermi/Highest occupied level energy that is to be included in the plot.
- **DOSPlot** (*str*) – a string that flags for the option to have either a DOS plot share the Y-axis of the band structure plot.

  Options include: "" | A blank string (default) will cause No Density of

  States plotted alongside the Band Structure

  "APDOS" | Atom Projected Density of States "DOS" | Normal Density of States
- **LSDA** (*bool*) – Plot the up and down of a spin polarized orbital projected DOS calculation.
- **runlocal** (*bool*) – a flag to choose whether or not to run the wrapped function now or write it to the _ID.py to run during the workflow
- **postfix** (*str*) – a string of an optional postfix to the plot filename for every calculation.

Returns None

dos(*yLim=[-10, 10], LSDA=False, runlocal=False, postfix=''*)

Wrapper method to call MTF.plot.dos in the high level user interface.

Parameters **self** – the plotter object

Keyword Arguments

- **yLim** (*list*) – a tuple or list of the range of energy around the fermi/Highest occupied level energy that is to be included in the plot.
- **LSDA** (*bool*) – Plot the up and down of a spin polarized DOS calculation.
- **runlocal** (*bool*) – a flag to choose whether or not to run the wrapped function now or write it to the _ID.py to run during the workflow
- **postfix** (*str*) – a string of an optional postfix to the plot filename for every calculation.

Returns None

opdos (*yLim=[-10, 10], LSDA=False, runlocal=False, postfix=''*)
Wrapper method to call MTF.plot.opdos in the high level user interface.

Parameters **self** – the plotter object

Keyword Arguments

- **yLim** (*list*) – a tuple or list of the range of energy around the fermi/Highest occupied level energy that is to be included in the plot.
- **LSDA** (*bool*) – Plot the up and down of a spin polarized orbital projected DOS calculation.
- **runlocal** (*bool*) – a flag to choose whether or not to run the wrapped function now or write it to the _ID.py to run during the workflow
- **postfix** (*string*) – a string of an optional postfix to the plot filename for every calculation.

Returns None

phonon (*runlocal=False*, *postfix=''*, *THz=True*)

transport (*runlocal=False*, *postfix=''*)
Wrapper method to call MTF.plot.epsilon in the high level user interface.

Parameters **self** – the plotter object

Keyword Arguments

- **nm** (*bool*) – whether to plot in nanometers for spectrum or eV for energy
- **runlocal** (*bool*) – a flag to choose whether or not to run the wrapped function now or write it to the _ID.py to run during the workflow

Returns None

prep.**prep_split_step** (*calcs*, *subset_creator*, *subset_tasks=[]*, *mult_jobs=False*, *substep_name='SUBSET'*, *keep_file_names=False*)

prep.**remove_blank_lines** (*inp_str*)
Removes whitespace lines of text

Parameters **inp_str** (*str*) – input string of text

Returns the same string with blank lines removed

prep.**runAfterAllDone** (*calcs*, *command*, *faultTolerant=True*)
Adds a command to the BATCH command block at the end of each calculation's _ID.py for all calculations in the set. Used to execute a command over all calculations in particular step have completed.

Parameters **calcs** (*dict*) – a dictionary of dicionaries representing the set of calculations

Keyword Arguments

- **command** (*str*) – the text to be added to the BATCH block
- **faultTolerant** (*bool*) – a flag to choose if we return True if some of the calculations ran but did not complete successful

Returns None

prep.**scfs** (*aflowkeys*, *allMTFVars*, *refFile*, *pseudodir=None*, *build_type='product'*, *build=True*)
Read a reference input file, and construct a set of calculations from the allMTFVars dictionary defining values

for the keywords in the reference input file. This will also create directory within the set directory for every calculation in the set.

Parameters

- **allMTFVars** (*dict*) – a dictionary whose keys correspond to the keywords in the reference input file and whose values will be used to construct the set of calculations
- **refFile** (*str*) – a filename as a string, a file object, or a string of the file that contains keywords to construct the inputs to the different calculations in the set

Keyword Arguments

- **pseudodir** (*str*) – path of the directory that contains your Pseudopotential files The value in the MTFrame config file used will override this.
- **build_type** (*str*) – how to construct the calculation set from allMTFVars dictionary:

  zip | The first calculation takes the first entry from the list of

  each of the keywords. The second calculation takes the second and so on. The keywords for all lists in allMTFVars must be the same length for this method.

  product | Calculation set is formed via a "cartesian product" with

  the values the list of each keyword combined. (i.e if allMTFVars has one keyword with a list of 5 entires and another with 4 and a third with 10, there would be 2000 calculations in the set formed from them via product mode.
- **build** (*bool*) – <DEFUNCT OPTION. NEEDS REMOVAL>

Returns A dictionary of dictionaries containing the set of calculations.

prep.**totree**(*tobecopied*, *calcs*, *rename=None*, *symlink=False*)

Populate all the subdirectories for the calculation with the file in input

Parameters

- **tobecopied** (*str*) – filepath to be copied to the MTF directory tree
- **calcs** (*dict*) – Dictionary of dictionaries of calculations

Keyword Arguments

- **rename** (*bool*) – option to rename the file/directory being moves into the MTFrame directory tree
- **symlink** (*bool*) – whether to copy the data to the MTFrame directory tree or to use symbolic links

Returns None

prep.**unlockAtomMovement**(*calcs*)

A Wrapper function that writes the function MTF.prep.__unfreezeAtoms to the __ID.py

Parameters **calcs** (*dict*) – a dictionary of dicionaries representing the set of calculations

Returns None

prep.**updateStructs**(*calcs*, *update_structure=True*, *update_positions=True*)

A Wrapper function that writes the function MTF.prep.__oneUpdateStructs to the __ID.py

Parameters **calcs** (*dict*) – a dictionary of dicionaries representing the set of calculations

Keyword Arguments

- **update_structure** (*bool*) – if True update the cell parameter if possible from the output of previous calculations in the workflow.
- **update_positions** (*bool*) – if True update the atomic positions if possible from the output of previous calculations in the workflow.

Returns The identical set of calculations as the input to this function

prep.**updatelogs**(*calcs*, *logname*, *runlocal=False*)

prep.**varyCellParams**(*oneCalc*, *ID*, *param=()*, *amount=0.15*, *steps=8*, *constraint=None*)

Forms and returns a set of calcs with varied cell params must be in A,B,C, and, in degrees,alpha,beta,gamma and then returns it.

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments

- **param** (*tuple*) – the params assoc. with the amount and step. i.e. ('celldm(1)','celldm(3))
- **amount** (*float*) – percentage amount to be varied up and down. i.e (0.04,0.02,0.01)
- **steps** (*int*) – how many steps to within each range. i.e (4,5,7)
- **constraint** (*list*) – a list or tuple containing two entry long list or tuples with the first being the constraint type and the second the free parameter in params that its constraining for example in a orthorhombic cell: constraint=(["volume",'c'],) allows for A and B to move freely but C is such that it keeps the cell volume the same in all calculations generated by the input oneCalc calculation.

Returns A dictionary of dictionaries representing a new calculation set

prep.**writeToScript**(*executable*, *calcs*, *from_step=0*)

Generates calls on several functions to set up everything that is needed for a new step in the workflow. The mechanics of the _ID.py are written to it here.

Parameters

- **calcs** (*dict*) – dictionary of dictionaries of calculations
- **executable** (*str*) – <DEFUNCT OPTION: HERE FOR LEGACY SUPPORT>
- ***args** – <DEFUNCT OPTION: HERE FOR LEGACY SUPPORT>

Keyword Arguments ****kwargs** – <DEFUNCT OPTION: HERE FOR LEGACY SUPPORT>

Returns A set of calculations for a new step in the workflow

pseudo module

pseudo.**__check_test_conv**(*oneCalc*, *ID*, *thresh*)

pseudo.**__crawlingMinimization**(*oneCalc*, *ID*, *fitVars=None*, *options=None*, *faultTolerant=True*, *initial_variance=0.15*, *steps=10*, *constraint=None*, *thresh=0.001*, *mult_jobs=True*)

this is a check to see if we're restarting when mult_jobs==True

pseudo.**__cubicInterpolation**(*variable_array*, *solution_array*)

try to convert a numpy array into a list if this fails assume that the inputs are lists

pseudo.**__cubicInterpolation_onePoint**(*point*, *variable_array*, *solution_array*)

try to convert a numpy array into a list if this fails assume that the inputs are lists

pseudo.**__genetic**(*calcs*)

pseudo.**__getCenter**(*origCalcs*, *fitVars=None*, *options=None*, *return_energy=False*)

pseudo.**__getMatrices**(*calcs*, *fitVars=None*, *options=None*)

pseudo.**__getMin**(*calcs*, *fitVars=None*, *options=None*, *bulk_modulus=False*, *minimize_var='Energy'*)

calculates the minimum energy value from the fit variables you supply. Only picks out the energy values that satisfy the cutoff/kpoint values for one combination of them that you are testing (i.e. would calculate the minimum energy from calculations that satisfy {_MTF_KPOINTS_ = '4 4 4 1 1 1', _MTF_ECUTW_ = 20}

Parameters **calcs** (*dict*) – Dictionary of Dictionaries of the calculations.

Keyword Arguments

- **fitVars** (*list*) – a tuple of the independent variables you want to make the fit and find the min energy for
- **options** (*dict*) – additional options passed to L-BFGS-B minimization (see scipy documentation for scipy.optimize.minimize for more details)

Returns **minEnergy** – the minimum energy of the fit outDict (dict): dictionary containing fit information

Return type float

pseudo.**__getMinimization**(*origCalcs*, *fitVars=None*, *options=None*, *return_energy=False*, *minimize_var='Energy'*)

Looks through the dictionary of dictionaries that you supply it and finds the minimum energy for the different cutoff/kpoint choices. outputs a list of dictionaries with the information about the cutoffs and the value of the variables that gives you a minimum energy.

Parameters **origCalcs** (*list*) – a list of Dictionary of Dictionaries of the calculations.

Keyword Arguments

- **fitVars** (*list*) – tuple of variables that you want to minimize energy with respect to.
- **options** (*dict*) – additional options passed to L-BFGS-B minimization (see scipy documentation for scipy.optimize.minimize for more details)
- **return_energy** (*bool*) – If true add the min energy to the resultList list for each
- **minimize_var** (*str*) – which key to minimize on in oneCalc

Returns

resultList – list if minumum energies and the parameters of the set that they

correspond to.

Return type list

pseudo.**__get_crawl_params**(*oneCalc*, *ID*)

pseudo.**__go_plot**(*oneCalc*, *ID*, *plot=False*)

pseudo.**__grabEnergyOut**(*calcs*)

<CONSIDER MOVING TO retr.py> Goes in every subdirectory of the calculation and searches for the final energy of the calculation and returns a new copy of the input dictionary that includes the final energy.

Parameters **calcs** (*dict*) – Dictionary of Dictionaries of the calculations.

Keyword Arguments **None**

MASTER_ENERGY (dict): Dictionary for the calc set with the energy added to each calc in the set

pseudo.**__medE**(*calcs*)

pseudo.**__passGlobalVar**(*varname*, *value*)

pseudo.**__plotOne**(*plots*, *labs*, *fig*, *entry*, *key*, *xaxis*, *pltTitle=None*, *rename=None*, *entryNum=0*, *maxs=None*, *mins=None*)

takes in a list of dictionaries of dictionaries of calculations and generates a plot with the x axis being some value in the list 'key' and splits the calculations and plots them with each plot being a unique combination of the items in key that are not

'xaxis'

Arguments: entry (list): list of dictionaries of dictionaries of calculations key (list): a list of cutoff variables used in the calculations xaxis (str): the cutoff that you choose to be the x axis in your plots

Keyword Arguments: plotTitle – title of the plots (default: None)

pseudo.**__record__thresh**(*oneCalc*, *ID*)

pseudo.**__set_cutoffs**(*oneCalc*, *ID*, *ecutwfc=[]*, *dual=[]*, *sampling=[]*)

pseudo.**__shiftGrid**(*calcs*, *outFile*, *fitVars=None*, *options=None*, *constraint=None*, *thresh=0.001*, *mult_jobs=True*)

sleep for anywhere from 0 to 10 seconds

pseudo.**__shift_cutoffs**(*oneCalc*, *ID*, *shift_type='wfc'*)

pseudo.**__splitCalcs**(*calcs*, *splitVars=''*)

splits a set of calculations by keyword and its respective values and returns the split set of calculations as a list of the split set.

Parameters **calcs** (*dict*) – dictionary of dictionary of calculations that is to be split

Keyword Arguments **splitVars** (*list*) – a list or tuple of strings of MTF variable(s) in the set of calculations that you want to split the set on

Returns **splitCalcList** – a list of dictionaries of dictionaries

Return type list

pseudo.**brute_test**(*calcs*, *ecutwfc*, *dual=None*, *sampling=None*, *constraint=None*, *thresh=0.001*, *initial_variance=0.05*, *grid_density=10*, *mult_jobs=False*, *conv_thresh=0.01*, *calc_type='relax'*)

pseudo.**crawlingMinimization**(*calcs*, *options=None*, *faultTolerant=True*, *constraint=None*, *thresh=0.001*, *initial_variance=0.05*, *grid_density=10*, *mult_jobs=False*, *final_minimization='relax'*)

pseudo.**getMinimization**(*origCalcs*, *fitVars=None*, *options=None*, *runlocal=False*, *faultTolerant=True*, *minimize_var='Energy'*)

pseudo.**plot**(*resultList*, *xaxis=''*, *xtitle=None*, *ytitle=None*, *title=None*, *rename=None*, *file_name=''*)

takes in a list of dictionaries of dictionaries of calculations and generates a plot with the x axis being some value in the list 'key' and splits the calculations and plots them with each plot being a unique combination of the items in key that are not 'xaxis'

Parameters

- **resultList** (*list*) – list of dictionaries of dictionaries of calculations
- **xaxis** (*str*) – the keyword in oneCalc that you choose to be the x axis in your plots

Keyword Arguments

- **xtitle** (*str*) – title of the x axis of the plot (default: None)
- **ytitle** (*str*) – title of the y axis of the plot (default: None)
- **plotTitle** (*str*) – title of the plots (default: None)
- **rename** (*dict*) – a mapping of the names of the keywords of whose values used to generate the plot to some other name. ex. {' _MTF_ECUTW_':'wavefunction cutoff'}
- **file_name** (*str*) – use this instead of "PT_RESULTS.pdf" as filename of plot

Returns None

CHAPTER 4

retr module

retr.**__cellMatrixToString**(*cellMatrix*)
Arguments:

Keyword Arguments:

Returns:

retr.**__cellStringToMatrix**(*cellParamString*)
Arguments:

Keyword Arguments:

Returns:

retr.**__checkEqualPoint**(*oldMatrix*, *newMatrix*)
Arguments:

Keyword Arguments:

Returns:

retr.**__constructSupercell**(*inputString*, *numX=1*, *numY=1*, *numZ=1*, *stringOrMatrix='String'*, *newVectors=True*)

retr.**__conv2PrimVec**(*cellParamMatrix*, *ibrav=0*)
Arguments:

Keyword Arguments:

Returns:

retr.**__conv2primPositions**(*symMatrix*, *cellParamMatrix*)
Arguments:

Keyword Arguments:

Returns:

retr.**__convertCartesian**(*symMatrix*, *cellMatrix*, *scaleFactor=1*)
Converts atomic positions from crystal to cartesian coordinates

Parameters

- **symMatrix** (*numpy.matrix*) – atomic positions in matrix form
- **cellMatrix** (*numpy.matrix*) – primitive lattice vectors in matrix form

Keyword Arguments **scaleFactor** (*int*) – scaling factor for output matrix

Returns in_cart (postions in cartiesian coordinates)

retr.**__convertCellBC**(*cellVectors*, *toPrimOrConv='conv'*)
Arguments:

Keyword Arguments:

Returns:

retr.**__convertFractional**(*symMatrix*, *cellMatrix*, *scaleFactor=1*)
Converts atomic positions from cartesian to crystal coordinates

Parameters

- **symMatrix** (*numpy.matrix*) – atomic positions in matrix form
- **cellMatrix** (*numpy.matrix*) – primitive lattice vectors in matrix form

Keyword Arguments **scaleFactor** (*int*) – scaling factor for output matrix

Returns **in_cart** – postions in crystal coordinates

Return type numpy.matrix

retr.**__duplicateEdgeAtoms**(*symMatrix*)
Arguments:

Keyword Arguments:

Returns:

retr.**__expandBoundaries**(*labels*, *symMatrix*, *numX*, *numY*, *numZ*, *beginX=0*, *beginY=0*, *beginZ=0*)

retr.**__expandBoundariesNoScale**(*labels*, *symMatrix*, *numX*, *numY*, *numZ*, *inList=False*, *expand=True*)

Parameters **symMatrix** (*numpy.matrix*) – atomic positions in matrix form

Keyword Arguments:

Returns:

retr.**__find_numkpoints**(*outputFile*)
DEFUNCT <TAGGED FOR REMOVAL>

Arguments:

Keyword Arguments:

Returns:

retr.**__free2celldm**(*cellparamatrix*, *ibrav=0*, *primitive=True*)
Convert lattice vectors to celldm

Parameters **cellparamatrix** (*numpy.matrix*) – matrix of cell vectors

Keyword Arguments

- **ibrav** (*int*) – Overrides the bravais lattice automatically detected (must be in QE convention if primitive)
- **primitive** (*bool*) – If True it will treat cellparamatrix as primitive lattice vectors

Returns **paramDict** – dictionary of the celldm generated by the input matrix

Return type dict

retr.**__getAlatFromInput**(*inputString*)
Grabs the value of alat from the QE pwscf input

Parameters **inputString** –

Keyword Arguments None

Returns **alat** – the alat scale for the primitive lattice vectors

Return type float

retr.**__getAtomNum**(*inputString*, *strip=False*)
Gets the number of atoms from a QE pwscf input file

Parameters **inputString** (*str*) – String of a QE pwscf input file

Keyword Arguments **strip** (*bool*) – If True then Cr2,Cr45,Cr would be all considered the same species

Returns **numOfEach** – dictionary with keys for the species labels and values the number

Return type collections.OrderedDict

retr.**__getCartConvMatrix**(*symMatrix*, *cellMatrix*)
Arguments:

Keyword Arguments:

Returns:

retr.**__getCellInput**(*oneCalc*, *ID*, *scaled=True*)
Gets the primitive cell vectors from the input file

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments **scaled** (*bool*) – scale vectors by alat

Returns **symMatrix** – matrix representing primive lattice vectors

Return type numpy.matrix

retr.**__getCellOutDim**(*oneCalc*, *ID*)
Returns the cell parameters a,b,c,alpha,beta,gamma of the primitive lattice from the output. If the primitive lattice does not change during the calculation it takes it from the input

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments None

Returns **out_params** – dictionary of a,b,c,alpha,beta,gamma of the primitive lattice from the output

Return type dict

retr.**__getCellParams**(*oneCalc*, *ID*)
Reads the output from the SCF or relax calculation to get the primitive cell parameters produced by the calculation. If it fails it defaults to the input lattice vectors

Parameters

- **oneCalc** (*dict*) – a dictionary of a single calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments None

Returns **alat** – the scale for the lattice vectors cell_matrix (numpy.array): the unscaled primitive lattice vectors

Return type float

retr.**__getCelldm2freeDict**(*free2celldm_output_dict*)
Cleans fortran array format into a format for python ex. celldm(1) to celldm1

Parameters **free2celldm_output_dict** (*dict*) – output from MTF.retr.free2celldm

Keyword Arguments **None**

Returns output_dict (dict) dictionary with cleaned keywords

retr.**__getConventionalCellFromInput**(*oneCalc*, *ID*)

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments:

Returns:

retr.**__getConventionalCellFromOutput**(*oneCalc*, *ID*)

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments:

Returns:

retr.**__getDist**(*oneCalc*, *ID*, *inputOrOutput='input'*, *distance=20.0*, *filterSpecies=None*, *atomNum=None*)

retr.**__getDistArray**(*labels*, *symMatrix*, *cellParamMatrix*, *dist_cutoff=10.0*, *atom_filter=None*, *atom_num=None*)

retr.**__getDistInput**(*inputString*, *distance=20.0*, *atom_filter=None*, *atom_num=None*)

retr.**__getDistOutput**(*inputFile*, *distance=20.0*)

retr.**__getEfermi**(*oneCalc*, *ID*)
Grabs fermi level from <ID>.efermi file. if there is none returns 0.0

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments **None**

Returns **efermi** – fermi level

Return type float

retr.**__getHighSymPoints**(*oneCalc*, *ID=None*)
Searching for the ibrav number in the input file for the calculation to determine the path for the band structure calculation

Parameters **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation

Keyword Arguments **ID** (*str*) – ID string for the particular calculation and step

Returns **special_points** – list of the HSP names band_path (str): path in string form

Return type list

retr.**__getInitialInputString**(*oneCalc*)
Gets initial input to the workflow

Parameters **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation

Keyword Arguments **None**

Returns **inFileString** – initial input to the workflow

Return type str

retr.**__getInitialOutputString**(*oneCalc*)
Gets initial output to the workflow

Parameters **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation

Keyword Arguments **None**

Returns **inFileString** – initial output to the workflow

Return type str

retr.**__getInputFileString**(*oneCalc*, *ID*)
Gets the string of the input that step of the calculation

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments **None**

Returns **inFileString** – input string to that calculation step

Return type str

retr.**__getInputParams**(*oneCalc*, *ID*)
Returns the cell parameters a,b,c,alpha,beta,gamma of the primitive lattice from the input.

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments **None**

Returns **out_params** – dictionary of a,b,c,alpha,beta,gamma of the primitive lattice from the input

Return type dict

retr.**__getOutputString**(*oneCalc*, *ID*)
Gets string of output for that particular step in the workflow for a single calcualtion

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments **None**

Returns **outFileString** – output of that particular step in the workflow

Return type str

retr.**__getPath**(*dk*, *oneCalc*, *ID=None*, *points=False*)
Get path between HSP

Parameters

- **dk** (*float*) – distance between points
- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation

Keyword Arguments

- **ID** (*str*) – ID string for the particular calculation and step
- **points** (*bool*) – Give the path as points or in aflow convention

Returns **numPointStr** – path between HSP

Return type str

retr.**__getPathFromFile**(*oneCalc*)
Reads the input file for the band structure calculation and retrieves the k point path from the file

Parameters **oneCalc** (*dict*) – one calculation that is a dictionary of values associated with that calculation

Keyword Arguments **None**

Returns **path** – k point path for bands in the bands pwscf input file

Return type str

retr.**__getPosLabels**(*inputString*)
Arguments:

Keyword Arguments:

Returns:

retr.**__getPositions**(*inputString*, *matrix=True*)
Arguments:

Keyword Arguments:

Returns:

retr.**__getStoicName**(*oneCalc*, *strip=False*, *latex=False*)
Determines the name of the compound by looking at the input and finds the stoichiometric number of species in the compound

Parameters **oneCalc** (*dict*) – one calculation that is a dictionary of values associated with that calculation

Keyword Arguments

- **strip** (*bool*) – strip species number when it equals 1
- **latex** (*bool*) – output string in latex format

Returns **name** – chemical namex

Return type str

retr.**__getSymList**(*ID*, *oneCalc*)
Gets symmetry operations from output if available

Parameters

- **ID** (*str*) – ID string for the particular calculation and step

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation

Keyword Arguments None

Returns **joinedList** – a string of the sym ops pulled from the engine output

Return type str

retr.**__getSymmInput**(*oneCalc*, *ID*)

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments:

Returns:

retr.**__get_pool_num**(*oneCalc*, *ID*)

Gets number of pools requested for this particular execution for engine.

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments None

Returns **npool** – number of pools requested for this particular execution for engine

Return type int

retr.**__invertX**(*coordVec*, *xScale=1*)

Arguments:

Keyword Arguments:

Returns:

retr.**__invertXYZ**(*coordVec*, *xScale*, *yScale*, *zScale*)

Arguments:

Keyword Arguments:

Returns:

retr.**__invertY**(*coordVec*, *yScale=1*)

Arguments:

Keyword Arguments:

Returns:

retr.**__invertZ**(*coordVec*, *zScale=1*)

Arguments:

Keyword Arguments:

Returns:

retr.**__joinInput**(*inputDict*)

Joins input tokenized by MTF.retr.__splitInput and returns a string of a QE pwscf input file

Parameters **inputDict** (*dict*) – tokenized input file

Keyword Arguments None

Returns **newInputString** – a string of a QE pwscf input file

Return type str

retr.**__joinMatrixLabels**(*labels*, *matrix*)
Joins a list of the atomic position species labels with a list or array of their atomic positions

Parameters

- **labels** (*list*) – list or array of the atomic positions species labels
- **matrix** (*numpy.matrix*) – the positions in matrix, array, or list form

Keyword Arguments **None**

Returns **outString** – a string of the atomic positions joined with their species labels

Return type str

retr.**__moveToSavedir**(*filePath*)
Move a file to the savedir specified in the config file

Parameters **filePath** (*str*) – path of the file to be copied

Keyword Arguments **None**

Returns None

retr.**__orderSplitInput**(*inputCalc*)
Order tokenized input as QE needs it

Parameters **inputCalc** (*calc*) – tokenized QE pwscf input

Keyword Arguments **None**

Returns **newOrderedDict** – the ordered tokenized QE pwscf input

Return type dict

retr.**__prefixFromInput**(*inputString*)
DEFUNCT <CONSIDER FOR REMOVAL>

Arguments:

Keyword Arguments:

Returns:

retr.**__prim2ConvMatrix**(*cellParamMatrix*, *ibrav=0*)
Arguments:

Keyword Arguments:

Returns:

retr.**__prim2ConvVec**(*cellParamMatrix*, *ibrav=0*)
Arguments:

Keyword Arguments:

Returns:

retr.**__prim2convPositions**(*labels*, *symMatrix*, *cellParamMatrix*)
Arguments:

Keyword Arguments:

Returns:

retr.**__pw2aflowConvention**(*cellParamMatrix*, *symMatrix*)
DEFUNCT <CONSIDER FOR REMOVAL>

Arguments:

Keyword Arguments:

Returns:

retr.**__pw2aflowConventionVec**(*cellParamMatrix*)
DEFUNCT <CONSIDER FOR REMOVAL>

Arguments:

Keyword Arguments:

Returns:

retr.**__pw2aflowPositions**(*cellParamMatrix*, *symMatrix*)
DEFUNCT <CONSIDER FOR REMOVAL>

Arguments:

Keyword Arguments:

Returns:

retr.**__pw2cif**(*oneCalc*, *ID*, *inOrOut='input'*, *outputFolder=None*, *filePrefix=''*)
Generates the CIF from input or output of single calculation at that step

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments

- **inOrOut** (*str*) – which structs to do CIF for.. 'input' 'output' or 'both'
- **outputFolder** (*str*) – Output directory for the CIF
- **filePrefix** (*str*) – Optional prefix to the CIF filenames

Returns None

retr.**__reduceDuplicates**(*symMatrix*)

Parameters **symMatrix** (*numpy.matrix*) – atomic positions in matrix form

Keyword Arguments:

Returns:

retr.**__rho2hex**(*cellParamMatrix*, *symMatrix*, *labels*)
Arguments:

Keyword Arguments:

Returns:

retr.**__rotateAlpha**(*coordVec*, *angle*)
Arguments:

Keyword Arguments:

Returns:

retr.**__rotateBeta**(*coordVec*, *angle*)
Arguments:

Keyword Arguments:

Returns:

retr.**__rotateGamma**(*coordVec*, *angle*)
Arguments:

Keyword Arguments:

Returns:

retr.**__shiftAfter**(*matrix*)

Parameters **matrix** (*numpy.matrix*) – sorts the matrix

Keyword Arguments:

Returns:

retr.**__shiftAfterRotation**(*coordVec*)

Parameters **coordVec** (*numpy.matrix*) – atomic positions in matrix form

Keyword Arguments:

Returns:

retr.**__shiftBeforeRotation**(*coordVec*)
Arguments:

Keyword Arguments:

Returns:

retr.**__shiftCell**(*symMatrix*)

Parameters **symMatrix** (*numpy.matrix*) – atomic positions in matrix form

Keyword Arguments:

Returns:

retr.**__shiftX**(*coordVec*, *shift*)

Parameters **coordVec** (*numpy.matrix*) – atomic positions in matrix form

Keyword Arguments:

Returns:

retr.**__shiftY**(*coordVec*, *shift*)

Parameters **coordVec** (*numpy.matrix*) – atomic positions in matrix form

Keyword Arguments:

Returns:

retr.**__shiftZ**(*coordVec*, *shift*)

Parameters **coordVec** (*numpy.matrix*) – atomic positions in matrix form

Keyword Arguments:

Returns:

retr.**__sortMatrix**(*matrix*)

Parameters **matrix** (*numpy.matrix*) – sorts the matrix

Keyword Arguments:

Returns:

retr.**__splitInput**(*inFileString*)
Tokenizes the QE pwscf input file

Parameters **inFileString** (*str*) – string of a QE pwscf input file

Keyword Arguments **None**

Returns **inputDict** – the tokenized input

Return type dict

retr.**__writeEfermi**(*oneCalc*, *ID*)
Grabs the fermi enery or HOMO energy from the output files of the dos calculations and converts into rydberg then writes it to file <ID>.efermi

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments **None**

Returns None

retr.**__writeInputFromOutput**(*oneCalc*, *ID*, *replace=False*)
Writes an input file of that step updated with the positions and lattice vectors of its output

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments **replace** (*bool*) – if True then replace the input with the updated one

Returns None

retr.**__writeInputFromOutputString**(*oneCalc*, *ID*)
Generates an input of that step updated with the positions and lattice vectors of its output

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments **None**

Returns **newInput** – string of that step's input updated with the positions and lattice vectors of its output

Return type str

retr.**abc2celldm**(*a=None*, *b=None*, *c=None*, *alpha=None*, *beta=None*, *gamma=None*, *ibrav=None*)
Convert a,b,c,alpha,beta,gamma into celldm for QE

Parameters **None** –

Keyword Arguments

- **a** (*float*) – length of a

- **b** (*float*) – length of b
- **c** (*float*) – length of c
- **alpha** (*float*) – Angle between axis b and c
- **beta** (*float*) – Angle between axis a and c
- **gamma** (*float*) – Angle between axis a and b
- **ibrav** (*int*) – ibrav to be used to convert for defaults

Returns **celldm_array** – an array of (ibrav,celldm(1),celldm(2),celldm(3),celldm(4),celldm(5),celldm(6))

Return type array

retr.**abc2free**(*a=None*, *b=None*, *c=None*, *alpha=None*, *beta=None*, *gamma=None*, *ibrav=None*, *returnString=True*)

Converts a,b,c,alpha,beta,gamma into QE convention primitive lattice vectors

Parameters **None** –

Keyword Arguments

- **a** (*float*) – length of a
- **b** (*float*) – length of b
- **c** (*float*) – length of c
- **alpha** (*float*) – Angle between axis b and c
- **beta** (*float*) – Angle between axis a and c
- **gamma** (*float*) – Angle between axis a and b
- **ibrav** (*int*) – bravais lattice type by QE convention
- **returnString** (*bool*) – return vectors as a string or as a numpy matrix

Returns Either a numpy.matrix or a string of the primitive lattice vectors generated from the celldm input

retr.**abcVol**(*a=None*, *b=None*, *c=None*, *alpha=None*, *beta=None*, *gamma=None*, *ibrav=None*)

Get volume from a,b,c,alpha,beta,gamma

Parameters **None** –

Keyword Arguments

- **a** (*float*) – length of a
- **b** (*float*) – length of b
- **c** (*float*) – length of c
- **alpha** (*float*) – Angle between axis b and c
- **beta** (*float*) – Angle between axis a and c
- **gamma** (*float*) – Angle between axis a and b
- **ibrav** (*int*) – ibrav to be used to convert for defaults

Returns **cell_vol** – volume of the cell

Return type float

retr.**atomicDistances**(*calcs*, *runlocal=False*, *inputOrOutput='input'*, *distance=20.0*)

retr.**attachPosFlags**(*positionString*, *flagString*)
Reattaches the flags to the end of the atomic position

Parameters

- **positionString** (*str*) – string of the atomic positions
- **flagString** (*str*) – string of the flags

Keyword Arguments None

Returns **positionString** – Positions with flags attached

Return type str

retr.**celldm2abc**(*ibrav=None*, *celldm1=None*, *celldm2=None*, *celldm3=None*, *celldm4=None*, *celldm5=None*, *celldm6=None*, *cosine=True*, *degrees=False*)
Convert celldm for espresso into A,B,C,and angles alpha,beta,gamma

Parameters **cellparamatrix** (*numpy.matrix*) – matrix of cell vectors

Keyword Arguments

- **cosine** (*bool*) – If True alpha,beta,gamma are cos(alpha),cos(beta),cos(gamma),
- **degrees** (*bool*) – If True return alpha,beta,gamma in degrees; radians if False

Returns **paramArray** – a list of the parameters a,b,c,alpha,beta,gamma generated from the input matrix

Return type list

retr.**celldm2free**(*ibrav=None*, *celldm1=None*, *celldm2=None*, *celldm3=None*, *celldm4=None*, *celldm5=None*, *celldm6=None*, *returnString=True*)
Converts QE's celldm format into QE convention primitive lattice vectors

Parameters **None** –

Keyword Arguments

- **ibrav** (*int*) – bravais lattice type by QE convention
- **celldm1** (*float*) – a
- **celldm2** (*float*) – b/a
- **celldm3** (*float*) – c/a
- **celldm4** (*float*) – Cosine of the angle between axis b and c
- **celldm5** (*float*) – Cosine of the angle between axis a and c
- **celldm6** (*float*) – Cosine of the angle between axis a and b
- **returnString** (*bool*) – return vectors as a string or as a numpy matrix

Returns Either a numpy.matrix or a string of the primitive lattice vectors generated from the celldm input

retr.**celldm2params**(*a*, *b*, *c*, *alpha*, *beta*, *gamma*, *k*, *v*)
DEFUNCT <Marked for removal>

Arguments:

Keyword Arguments:

Returns:

retr.**checkStatus**(*PROJECT*, *SET=''*, *config=''*, *step=0*, *status={}*, *negate_status=False*)
function that loads the calclogs of each of the calculations in your run and displays status information about them.

Parameters **PROJECT** (*str*) – Project name

Keyword Arguments

- **SET** (*str*) – Set name
- **config** (*str*) – Config file used
- **step** (*int*) – Which number step to seee (default all of them)
- **status** (*dict*) – dictionary containing the status and the value you want to see (ex. {'Error':True})
- **negate_status** (*bool*) – Whether to take the calculations with that status or without it

Returns calcsList (list) list of each step of the calculations you selected that satisfay the status criteria

retr.**chemAsKeys**(*calcs*)
For sets of calcs that only differ by chemistry you can replace the hash by the chemical name. May not always work. Especially if there are two compounds with swapped positions of atoms of different elements

Parameters **calcs** (*dict*) – dictionary of dictionaries of calculations

Keyword Arguments **None**

Returns **calcsCopy** – returns new dictionay with keys as the chemical stoiciometry

Return type dict

retr.**compMatrices**(*matrix1*, *matrix2*)
Compare two numpy matrices to see if they are equal or not

Parameters

- **matrix1** (*numpy.matrix*) – first matrix
- **matrix2** (*numpy.matrix*) – second matrix

Keyword Arguments **None**

Returns **equalBool** – True if they're equal False if they're not

Return type bool

retr.**convertFCC**(*oneCalc*, *ID*)

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments:

Returns:

retr.**detachPosFlags**(*positionString*)
Detach the position flags from the string of the atomic positions

Parameters **positionString** (*str*) – string of the atomic positions

Keyword Arguments **None**

Returns **positions** – atomic positions as a list flags (list): a list of the flags for by each position

Return type list

retr.**free2abc**(*cellparamatrix*, *cosine=True*, *degrees=True*, *string=True*, *bohr=False*)
Convert lattice vectors to a,b,c,alpha,beta,gamma of the primitive lattice

Parameters **cellparamatrix** (*numpy.matrix*) – matrix of cell vectors

Keyword Arguments

- **cosine** (*bool*) – If True alpha,beta,gamma are cos(alpha),cos(beta),cos(gamma),
- **degrees** (*bool*) – If True return alpha,beta,gamma in degrees; radians if False
- **string** (*bool*) – If True return a,b,c,alpha,beta,gamma as a string; if False return as a list
- **bohr** (*bool*) – If True return a,b,c in bohr radii; if False return in angstrom

Returns **paramArray** – a list of the parameters a,b,c,alpha,beta,gamma generated from the input matrix

Return type list

retr.**free2ibrav**(*cellparamatrix*, *ibrav=0*, *primitive=True*)
Convert lattice vectors to celldm

Parameters **cellparamatrix** (*numpy.matrix*) – matrix of cell vectors

Keyword Arguments

- **ibrav** (*int*) – Overrides the bravais lattice automatically detected (must be in QE convention if primitive)
- **primitive** (*bool*) – If True it will treat cellparamatrix as primitive lattice vectors

Returns **ibravStr** – string of the celldm used for QE pwscf input

Return type str

retr.**getBravaisLatticeName**(*bravaisLatticeNumber*)
Returns the name of the crystal system from the bravias lattice number

Parameters **braviasLatticeNumber** (*int*) – the number of the bravais lattice in QE convention

Keyword Arguments **None**

Returns A string of the name of the crystal system

retr.**getCellMatrixFromInput**(*inputString*, *string=False*, *scale=True*)
Get the primitive cell vectors from the input.

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments

- **string** (*bool*) – Return a string or matrix
- **scale** (*bool*) – scale the vectors by alat

Returns **cellParamMatrix** – primitive lattice params

Return type numpy.matrix

retr.**getCellOutput**(*oneCalc*, *ID*)
retreives information about the structure and its chemistry and prints it into a file in the MTF folder inside the project/set directories

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments None

Returns **outputStr** – Output of the report for the calculation

Return type str

retr.**getCellVolume**(*oneCalc*, *ID*, *conventional=True*, *string=True*)
Gets the cell volume from output if avaiable and if not gets it from the input

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments

- **conventional** (*bool*) – return the volume of the primitive or conventional lattice
- **string** (*bool*) – to return as a string or as a float

Returns **vol** – Volume of the cell

Return type str

retr.**getCellVolumeFromVectors**(*cellInput*)
Calculates cell volume from basis vectors

Parameters **cellInput** (*numpy.matrix*) – basis set defining the cell

Keyword Arguments None

Returns **vol** – volume of the cell (may be not scaled. it depends on your input matrix)

Return type float

retr.**getForce**(*oneCalc*, *ID*)
Gets last entry of the total force in the calculation from the engine output.

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments None

Returns **force_string** – Total force from engine output

Return type str

retr.**getIbravFromVectors**(*cellVectors*)
Arguments:

Keyword Arguments:

Returns:

retr.**getPointGroup**(*oneCalc*, *ID*, *source='input'*)
NOT COMPLETED.

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments **source** (*string*) – either 'input' or 'output' to get point group from input or output atomic positions

Returns None

retr.**getPositionsFromInput**(*oneCalc*, *ID*)

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments:

Returns:

retr.**getPositionsFromOutput**(*oneCalc*, *ID*)
Get the atomic positions from the output. If atomic positons can not be read from output then the the positions from the input are returned.

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments **None**

Returns **pos** – atomic positions

Return type numpy.matrix

retr.**getRecipParams**(*oneCalc*)
reads the output from the SCF or relax calculation to get the reciprocal cell parameters produced by the calculation.

Parameters **oneCalc** (*dict*) – a dictionary of a single calculation

Keyword Arguments **None**

Returns **alat** – alat multiplier paramMatrix (numpy.matrix): matrix of reciprocal lattice vectors

Return type float

retr.**get_parameters**(*oneCalc*, *ID*, *conventional=True*)

retr.**glideXshiftY**(*symMatrix*, *cellMatrix*, *shift*)

retr.**glideXshiftZ**(*symMatrix*, *cellMatrix*, *shift*)

retr.**glideYshiftX**(*symMatrix*, *cellMatrix*, *shift*)

retr.**glideYshiftZ**(*symMatrix*, *cellMatrix*, *shift*)

retr.**glideZshiftX**(*symMatrix*, *cellMatrix*, *shift*)

retr.**glideZshiftY**(*symMatrix*, *cellMatrix*, *shift*)

retr.**grabEnergy**(*oneCalc*, *ID*)

Grabs energy for oneCalc and adds the keyword 'Energy' with the value of the energy grabbed from the output

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments None

Returns

oneCalc – a dictionary containing properties about the MTFrame calculation

with the 'Energy' keyword added

Return type dict

retr.**grabEnergyOut**(*calcs*)

Goes in every subdirectory of the calculation and searches for the final energy of the calculation and returns a new copy of the input dictionary that includes the final energy.

Parameters **calcs** (*dict*) – dictionary of dictionaries of calculations

Keyword Arguments None

Returns **calcs** – dictionary of dictionaries of calculations with energy added

Return type dict

retr.**ibrav2String**(*ibrav=None*, *celldm1=None*, *celldm2=None*, *celldm3=None*, *celldm4=None*, *celldm5=None*, *celldm6=None*)

DEFUNCT <CONSIDER FOR REMOVAL>

Arguments:

Keyword Arguments:

Returns:

retr.**inputDict2params**(*inputDict*)

Arguments:

Keyword Arguments:

Returns:

retr.**invertX**(*symMatrix*, *cellMatrix*)

Parameters

- **symMatrix** (*numpy.matrix*) – atomic positions in matrix form
- **cellMatrix** (*numpy.matrix*) – primitive lattice vectors in matrix form

Keyword Arguments:

Returns:

retr.**invertXYZ**(*symMatrix*, *cellMatrix*)

Parameters

- **symMatrix** (*numpy.matrix*) – atomic positions in matrix form
- **cellMatrix** (*numpy.matrix*) – primitive lattice vectors in matrix form

Keyword Arguments:

Returns:

retr.**invertY**(*symMatrix*, *cellMatrix*)

Parameters

- **symMatrix** (*numpy.matrix*) – atomic positions in matrix form
- **cellMatrix** (*numpy.matrix*) – primitive lattice vectors in matrix form

Keyword Arguments:

Returns:

retr.**invertZ**(*symMatrix*, *cellMatrix*)

Parameters

- **symMatrix** (*numpy.matrix*) – atomic positions in matrix form
- **cellMatrix** (*numpy.matrix*) – primitive lattice vectors in matrix form

Keyword Arguments:

Returns:

retr.**pw2cif**(*calcs*, *inpt=True*, *outp=True*, *runlocal=False*, *outputFolder=None*, *filePrefix=''*)

Writes a simple CIF file for engine input and outputs for viewing the structure

Parameters **calcs** (*dict*) – dictionary of dictionaries of calculations

Keyword Arguments

- **inpt** (*bool*) – Do CIF for input
- **outp** (*bool*) – Do CIF for output
- **outputFolder** (*str*) – Output directory for the CIF
- **filePrefix** (*str*) – Optional prefix to the CIF filenames

Returns None

retr.**rotateAlpha**(*symMatrix*, *cellMatrix*, *angle*)

Parameters

- **symMatrix** (*numpy.matrix*) – atomic positions in matrix form
- **cellMatrix** (*numpy.matrix*) – primitive lattice vectors in matrix form

Keyword Arguments:

Returns:

retr.**rotateBeta**(*symMatrix*, *cellMatrix*, *angle*)

Parameters

- **symMatrix** (*numpy.matrix*) – atomic positions in matrix form
- **cellMatrix** (*numpy.matrix*) – primitive lattice vectors in matrix form

Keyword Arguments:

Returns:

retr.**rotateGamma**(*symMatrix*, *cellMatrix*, *angle*)

Parameters

- **symMatrix** (*numpy.matrix*) – atomic positions in matrix form
- **cellMatrix** (*numpy.matrix*) – primitive lattice vectors in matrix form

Keyword Arguments:

Returns:

retr.**shiftCell**(*symMatrix*)

Parameters **symMatrix** (*numpy.matrix*) – atomic positions in matrix form

Keyword Arguments:

Returns:

retr.**shiftX**(*symMatrix*, *cellMatrix*, *shift*)

Parameters

- **symMatrix** (*numpy.matrix*) – atomic positions in matrix form
- **cellMatrix** (*numpy.matrix*) – primitive lattice vectors in matrix form

Keyword Arguments:

Returns:

retr.**shiftY**(*symMatrix*, *cellMatrix*, *shift*)

Parameters

- **symMatrix** (*numpy.matrix*) – atomic positions in matrix form
- **cellMatrix** (*numpy.matrix*) – primitive lattice vectors in matrix form

Keyword Arguments:

Returns:

retr.**shiftZ**(*symMatrix*, *cellMatrix*, *shift*)

Parameters

- **symMatrix** (*numpy.matrix*) – atomic positions in matrix form
- **cellMatrix** (*numpy.matrix*) – primitive lattice vectors in matrix form

Keyword Arguments:

Returns:

retr.**supercell**(*inputString*, *numX=1*, *numY=1*, *numZ=1*)

retr.**transform_input_conv**(*oneCalc*, *ID*)

Parameters

- **oneCalc** (*dict*) – a dictionary containing properties about the MTFrame calculation
- **ID** (*str*) – ID string for the particular calculation and step

Keyword Arguments:

Returns:

retr.**writeInputFromOutput**(*calcs*, *replace=False*, *runlocal=False*)

Parameters **calcs** (*dict*) – dictionary of dictionaries of calculations

Keyword Arguments

- **replace** (*bool*) – If true replace the input with updated atomic positions and cell parameters
- **runlocal** (*bool*) – run local or write to <ID>.py

Returns None

CHAPTER 5

run module

run.**__PW_bands_fix**(*oneCalc*, *ID*)

Accounts for the occational problem of the bands.x output being improperly formatted for MTFrame to parse later.

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments None

Returns None

run.**__collect_fd_field_forces**(*oneCalc*, *ID*, *for_type='raman'*)

Runs at the end of the scf calculations for the Finite Difference phonon calculations. It uses regex to pull the forces to be saved to files for parsing by fd_ifc.x

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments **for_type** (*str*) – type of file to pull from in FD_PHONON dir (choices are 'raman', 'born', 'epol')

Returns force_out_str (str) a string of the contents of the file

run.**__colorize_message**(*string*, *level='ERROR'*, *show_level=True*)

Colorizes text. Used for colorizing the logging

Parameters **string** (*str*) – A string of text

Keyword Arguments **level** (*str*) – Specific colors are chosen for logging message type

Returns **levelname_color** – Colorized version of the string input

Return type str

run.**__convert_fortran_double**(*fort_double_string*, *string_output=False*)

find the start time of current step in chain and record that into the walltime file

Parameters **fort_double_string** (*str*) – Fortran double in string form

Keyword Arguments **string_output** (*bool*) – Output a string or a float

Returns **fort_double_string** – the fotran double in float form or string

Return type float

run.**__exitClean**(*signal*, *frame*)
If the terminate 15 signal is received exit with 0 exit code

Parameters

- **signal** (*signal.SIGNAL*) – *nix signal
- **frame** (*frame*) – Inspect frame

Keyword Arguments None

Returns None

run.**__field_factory**(*input_str*, *field_strength=0.003*, *for_type='raman'*, *nberrycyc=3*, *dir_index=0*)
Modifies the QE pwscf input for the finite field calc of a given index

Parameters **input_str** (*str*) – string of QE pwscf input file

Keyword Arguments

- **feild_strength** (*float*) – applied electric field strength
- **nberrycyc (int) number of berry phase cycles**
- **for_type** (*str*) – which type of FD input do we want to generate files for. 'raman','born','eps'
- **dir_index** (*int*) – index of the finite field direction to choose

Returns **new_input** – the modified input string

Return type str

run.**__gen_fd_input**(*oneCalc*, *ID*, *for_type='raman'*, *de=0.003*)
Generates the input files for the finite field calcs from the input file string in oneCalc

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments

- **for_type** (*str*) – which type of FD input do we want to generate files for. 'raman','born','eps'
- **de** (*float*) – applied electric field strength

Returns None

run.**__generic_restart_check**(*oneCalc*, *ID*, *__submitNodeName__*)
Checks to see if walltime limit is almost up. If it is within the buffer of time The job is resubmitted.

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation
- **__submitNodeName__** (*str*) – String of hostname that cluster jobs should be submitted from

Keyword Arguments None

Returns None

run.**__getEnginePath**(*engine*, *calcType*)
Gives the name of the executable file for the ab initio engine for a given type of calculation.

Parameters

- **engine** (*str*) – Ab initio engine being used
- **calcType** (*str*) – Type of calculation to be done

Keyword Arguments **None**

Returns **execPath** – the name of the executable for that engine for that calcType

Return type str

run.**__getExecutable**(*engine*, *calcType*)
Gives the name of the executable file for the ab initio engine for a given type of calculation.

OBSOLETE. NEEDS REMOVAL. MTF.run.__getEnginePath is almost identical

Parameters

- **engine** (*str*) – Ab initio engine being used
- **calcType** (*str*) – Type of calculation to be done

Keyword Arguments **None**

Returns **executable** – the name of the executable for that engine for that calcType

Return type str

run.**__getWalltime**(*oneCalc*, *ID*)
Get the walltime requested from the cluster submission script

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments **None**

Returns walltime (int) amount of time requested

run.**__get_index_from_pp_step**(*oneCalc*, *ID*)
DEFUNCT. CANDIDATE FOR REMOVAL

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments **None**

Returns **ID** – Returns the input ID

Return type str

run.**__get_qsub_name**(*path*)
Takes path of the cluster submission file and forms a name for the submission

Parameters **path** (*str*) – path of the cluster job submission file

Keyword Arguments **None**

Returns **calcName** – name of the calculation used with the submission

Return type str

run.**__grabWalltime**(*oneCalc*, *ID*)
find the start time of current step in chain and record that into the walltime file

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments None

Returns **walltimeStart** – walltime requested startTime (int): start time of the script

Return type int

run.**__io_error_restart**(*ID*, *oneCalc*, *__submitNodeName__*)
Restarts if I/O error encountered

BROKEN/NOT NEEDED POSSIBLY DELETE

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation
- **__submitNodeName__** (*str*) – String of hostname that cluster jobs should be submitted from

Keyword Arguments None

Returns None

run.**__makeInput**(*oneCalc*, *engine*, *calcType*, *ID=''*)
Writes the input file for postprocessing calculations

Parameters

- **oneCalc** (*dict*) – Dictionary of one of the calculations
- **engine** (*str*) – Particular engine executable being used for the postprocessing step that you are calling to run the calculations (default='pw.x')
- **calcType** (*str*) – Type of PP calculation to be done

Keyword Arguments **ID** (*str*) – ID hash for the calculation. Needed for the filename (<ID>_<calcType>.in)

Returns **stringDict** – Input string of the PP step

Return type str

run.**__onePrep**(*oneCalc*, *ID*, *execPrefix=None*, *execPostfix=None*, *engine='espresso'*, *calcType=''*, *alt_ID=None*)
Prepares one calculation run an engine executable

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments

- **engine** (*str*) – executable that you are calling to run the calculations (default='pw.x')
- **execPrefix** (*str*) – commands to go before the executable when run (ex. mpiexec nice -n 19 <executable>) (default = None)
- **execPostfix** (*str*) – commands to go after the executable when run (ex. <execPrefix> <executable> -npool 12) (default = None)

- **calcType** (*str*) – used to prep for a particular type of calc (i.e. 'scf','pdos'..etc)

Returns None

run.**__oneRun**(*__submitNodeName__*, *oneCalc*, *ID*, *execPrefix=''*, *execPostfix=''*, *engine='espresso'*, *calcType=None*, *executable=None*, *execPath=None*, *nextCalc=None*, *nextConf=None*)

Run a single calculation in the dictionary with a specific engine

Parameters

- **oneCalc** (*dict*) – dictionary of the calculation
- **ID** (*str*) – Identifying hash of the calculation

Keyword Arguments

- **engine** (*str*) – executable that you are calling to run the calculations
- **execPrefix** (*str*) – commands to go before the executable when run (ex. mpiexec nice -n 19 <executable>) (default = None)
- **execPostfix** (*str*) – commands to go after the executable when run (ex. <execPrefix> <executable> -ndiag 12 -nimage 2) (default = None)
- **calcType** (*str*) – used to pull the engine post processing executable to the calc dir and to write the postprocessing input file if needed
- **executable** (*str*) – if the executable has already been copied to the calc's directory then use this to run the input <ID>.in
- **execPath** (*str*) – path of the executable if needed
- **nextCalc** (*str*) – DEFUNCT
- **nextConf** (*str*) – DEFUNCT

Returns None

run.**__phonon_band_path**(*oneCalc*, *ID*)

Gets path for the cell between High Symmetry points in Brillouin Zone and returns for to be part of matdyn.x input for the phonon dispersion

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments **None**

Returns **path** – Path between High Symmetry points in Brillouin Zone

Return type str

run.**__pp_phonon**(*__submitNodeName__*, *oneCalc*, *ID*, *LOTO=True*, *de=0.01*, *raman=True*)

Calls the executables for the post processing of the force data generated by the scf calculations created by fd.x

Parameters

- **__submitNodeName__** (*str*) – String of hostname that cluster jobs should be submitted from
- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments **None**

Returns None

run.**__pull_born_out**(*oneCalc*, *ID*)
Runs at the end of the scf calculations for the Finite Difference phonon calculations. It uses regex to pull the born effective charges to be saved to files for use by matdyn.x

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments **None**

Returns born_string (str) string of the born eff. charges

run.**__pull_eps_out**(*oneCalc*, *ID*)
Runs at the end of the scf calculations for the Finite Difference phonon calculations. It uses regex to pull the eps_0 to be saved to files for use by matdyn.x

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments **None**

Returns **eps_string** – string of the eps

Return type str

run.**__pull_forces**(*oneCalc*, *ID*)
Runs at the end of the scf calculations for the Finite Difference phonon calculations. It uses regex to pull the forces to be saved to files for parsing by fd_ifc.x

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments **None**

Returns None

run.**__pull_polarization**(*oneCalc*, *ID*)
Runs at the end of the scf calculations for the Finite Difference phonon calculations. It uses regex to pull the forces to be saved to files for parsing by fd_ifc.x

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments **None**

Returns None

run.**__qe__pre_run**(*oneCalc*, *ID*, *calcType*, *__submitNodeName__*, *engine*)
Performs the pre-run checks and restarts the cluster job if needed. On local mode this gets skipped

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation
- **calcType** (*str*) – type of calculation

- **__submitNodeName__** (*str*) – String of hostname that cluster jobs should be submitted from

Keyword Arguments None

Returns **oneCalc** – dictionary of one of the calculations ID (str): ID of calculation

Return type dict

run.**__qsubGen**(*oneCalc*, *ID*)
Generates the cluster job submission file

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments None

Returns **qsubFileString** – string of cluster submission file

Return type str

run.**__readWalltimeLog**(*oneCalc*, *ID*)
reads walltime log from oneCalc

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments None

Returns

walltimeLog – dictionary containing start time for MTFrame as well

as the requested walltime for the cluster job

Return type dict

run.**__recordDeath**(*signal*, *frame*)
If the 15 signal to terminate the process is sent. Record it in oneCalc['__status__']['Error']

Parameters

- **signal** (*signal.SIGNAL*) – *nix signal
- **frame** (*frame*) – Inspect frame

Keyword Arguments None

Returns None

run.**__restartGIPAW**(*oneCalc*, *ID*, *a*, *__submitNodeName__*)
If pw.x ends becuause it hit max_seconds. set the input to restart the calculation and resubmit the job

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation
- **a** (*float*) – Time that has passed since calculation has started
- **__submitNodeName__** (*str*) – String of hostname that cluster jobs should be submitted from

Keyword Arguments None

Returns None

run.**__restartPW**(*oneCalc*, *ID*, *a*, *__submitNodeName__*)
If pw.x ends becuause it hit max_seconds. set the input to restart the calculation and resubmit the job

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation
- **a** (*float*) – Time that has passed since calculation has started
- **__submitNodeName__** (*str*) – String of hostname that cluster jobs should be submitted from

Keyword Arguments None

Returns None

run.**__restartScript**(*oneCalc*, *ID*, *PID*)
special script started when the _<ID>.py script starts. reads the walltime in the qsub file and resubmits and then exits if the time ran gets within 1 minute of the walltime requested. Used for when the script is on the post processing stages and may happen to get killed by the cluster daemon.

run.**__setStartTime**(*oneCalc*, *ID*)
If this python script is the first to run in for MTFrame in a cluster job then the start time will be recorded and stored in the oneCalc object with the key "__walltime_dict__" and the values for the start time of the script and the requested walltime.

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments None

Returns **walltimeStart** – walltime requested startTime (int): start time of the script

Return type int

run.**__setupRestartGIPAW**(*oneCalc*, *ID*)
Sets up GIPAW input with max_seconds to it will cleanly exit if it doesn't finish within the time limit of the cluster submission walltime requested.

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments None

Returns **oneCalc** – dictionary of one of the calculations ID (str): ID of calculation

Return type dict

run.**__setupRestartPW**(*oneCalc*, *ID*, *__submitNodeName__*)
Sets up pw.x input with max_seconds to it will cleanly exit if it doesn't finish within the time limit of the cluster submission walltime requested.

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations

- **ID** (*str*) – ID of calculation
- **__submitNodeName__** (*str*) – String of hostname that cluster jobs should be submitted from

Keyword Arguments None

Returns **oneCalc** – dictionary of one of the calculations ID (str): ID of calculation

Return type dict

run.**__setup_raman**(*calc_subset*, *orig_oneCalc*, *field_strength=0.003*, *nberrycyc=3*, *for_type='raman'*)
Sets up the input files and the command to run the finite field calculations.

Parameters

- **calc_subset** (*dict*) – dictionary of dictionaries of the FD_PHONON calculations
- **orig_oneCalc** (*dict*) – oneCalc of one calculation the main calc set

Keyword Arguments

- **feild_strength** (*float*) – applied electric field strength
- **nberrycyc (int) number of berry phase cycles**
- **for_type** (*str*) – which type of FD input do we want to generate files for. 'raman','born','eps'

Returns None

run.**__skeletonRun**(*calcs*, *engine=''*, *execPrefix=None*, *execPostfix=None*, *holdFlag=True*, *config=None*)
Wrapper to set up a custom calculation. Inputs and oneCalc calculation dictionary must be created before calling this.

Parameters **calcs** (*dict*) – Dictionary of dictionaries of calculations

Keyword Arguments

- **engine** (*str*) – executable that you are calling to run the calculations
- **execPrefix** (*str*) – commands to go before the executable when run (ex. mpiexec nice -n 19 <executable>) (default = None)
- **execPostfix** (*str*) – commands to go after the executable when run (ex. <execPrefix> <executable> -ndiag 12 -nimage 2) (default = None)
- **holdFlag** (*bool*) – DEFUNCT. NEEDS REMOVAL
- **config** (*str*) – DEFUNCT. NEEDS REMOVAL

Returns None

run.**__submitJob**(*ID*, *oneCalc*, *__submitNodeName__*, *sajOverride=False*, *forceOneJob=False*)
Submits a step of a calculation's pipeline to be run

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation
- **__submitNodeName__** (*str*) – String of hostname that cluster jobs should be submitted from

Keyword Arguments

- **sajOverride** (*bool*) – Overrides stepsasjobs=False in the config file used

- **forceOneJob** (*bool*) – Overrides stepsasjobs=True in the config file used

Returns None

run.**__swap_scf_inputs**(*oneCalc*, *ID*)
Swaps the value of oneCalc['_MTF_INPUT_'] in oneCalc to the finite field input

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments **None**

Returns **oneCalc** – dictionary of one of the calculations

Return type dict

run.**__testOne**(*ID*, *oneCalc*, *engine=''*, *calcType=''*, *execPrefix=None*, *execPostfix=None*)
Stages the first the first step in a calculation workflow of a calc set to be submitted

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments

- **engine** (*str*) – executable that you are calling to run the calculations
- **execPrefix** (*str*) – commands to go before the executable when run (ex. mpiexec nice -n 19 <executable>) (default = None)
- **execPostfix** (*str*) – commands to go after the executable when run (ex. <execPrefix> <executable> -ndiag 12 -nimage 2) (default = None)
- **calcType** (*str*) – used to pull the engine post processing executable to the calc dir and to write the postprocessing input file if needed

Returns None

run.**__vcrelax_error_restart**(*ID*, *oneCalc*, *__submitNodeName__*)
Restarts if vc-relax calculation fails

BROKEN/NOT NEEDED POSSIBLY DELETE

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation
- **__submitNodeName__** (*str*) – String of hostname that cluster jobs should be submitted from

Keyword Arguments **None**

Returns None

run.**__writeWalltimeLog**(*oneCalc*, *ID*, *walltimeDict*)
Writes the walltimeDict to disk (_<ID>.oneCalc)

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

- **walltimeDict** (*dict*) – saves the walltime log to oneCalc and then to disk

Keyword Arguments None

Returns None

run.**_fancy_error_log**(*e*)
Logs an error and prints it out on the stdout if logging=debug in the config file used

Parameters **e** (*str*) – string of the error

Keyword Arguments None

Returns None

run.**addatexit__**(*command*, **args*, ***kwargs*)
Wrapper to add function to be run at exit

Parameters

- **command** (*func*) – function to be run
- ***args** – arguments for command

Keyword Arguments ****kwargs** – Keyword arguments for command

Returns None

run.**bands**(*calcs*, *engine=''*, *execPrefix=None*, *execPostfix=' '*, *holdFlag=True*, *config=None*)
Wrapper to set up Electronic Band Structure calculation

Parameters **calcs** (*dict*) – Dictionary of dictionaries of calculations

Keyword Arguments

- **engine** (*str*) – executable that you are calling to run the calculations
- **execPrefix** (*str*) – commands to go before the executable when run (ex. mpiexec nice -n 19 <executable>) (default = None)
- **execPostfix** (*str*) – commands to go after the executable when run (ex. <execPrefix> <executable> -ndiag 12 -nimage 2) (default = None)
- **holdFlag** (*bool*) – DEFUNCT. NEEDS REMOVAL
- **config** (*str*) – DEFUNCT. NEEDS REMOVAL

Returns None

run.**clean_cell_params**(*output*)
Parses the atomic shifts in a supercell from fd.x outputted pw.x input files to correct for formatting issues when they are imported to MTFrame

Parameters **output** (*str*) – pw.x input files generated by fd.x

Keyword Arguments None

Returns **output** – pw.x input files generated by fd.x (cleaned by MTFrame)

Return type str

run.**cleanup**(*calcs*)
Deletes all files a calculation set's directory tree that are prepended with '_'

Parameters **calcs** (*dict*) – Dictionary of dictionaries of calculations

Keyword Arguments None

Returns None

run.**dos**(*calcs*, *engine=''*, *execPrefix=None*, *execPostfix=None*, *holdFlag=True*, *config=None*)
Wrapper to set up DOS nscf calculation

Parameters **calcs** (*dict*) – Dictionary of dictionaries of calculations

Keyword Arguments

- **engine** (*str*) – executable that you are calling to run the calculations
- **execPrefix** (*str*) – commands to go before the executable when run (ex. mpiexec nice -n 19 <executable>) (default = None)
- **execPostfix** (*str*) – commands to go after the executable when run (ex. <execPrefix> <executable> -ndiag 12 -nimage 2) (default = None)
- **holdFlag** (*bool*) – DEFUNCT. NEEDS REMOVAL
- **config** (*str*) – DEFUNCT. NEEDS REMOVAL

Returns None

run.**emr**(*calcs*, *engine=''*, *execPrefix=None*, *execPostfix=None*, *holdFlag=True*, *config=None*)
Wrapper to set up GIPAW EMR calculation

Parameters **calcs** (*dict*) – Dictionary of dictionaries of calculations

Keyword Arguments

- **engine** (*str*) – executable that you are calling to run the calculations
- **execPrefix** (*str*) – commands to go before the executable when run (ex. mpiexec nice -n 19 <executable>) (default = None)
- **execPostfix** (*str*) – commands to go after the executable when run (ex. <execPrefix> <executable> -ndiag 12 -nimage 2) (default = None)
- **holdFlag** (*bool*) – DEFUNCT. NEEDS REMOVAL
- **config** (*str*) – DEFUNCT. NEEDS REMOVAL

Returns None

run.**generateSubRef**(*qsubRefFileString*, *oneCalc*, *ID*)
Reads in the reference cluster submission file specified in "jobreffile" in the config used. Tries to insert a few parameters.

OBSOLETE PLANNED FOR REMOVAL

Parameters

- **qsubRefFileString** (*str*) – string of the "reference" cluster submission file
- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments **None**

Returns **clusterTypeDict** – string cluster submission file

Return type dict

run.**gvectors**(*calcs*, *engine=''*, *execPrefix=None*, *execPostfix=None*, *holdFlag=True*, *config=None*)
Wrapper to set up GIPAW gvectors calculation

Parameters **calcs** (*dict*) – Dictionary of dictionaries of calculations

Keyword Arguments

- **engine** (*str*) – executable that you are calling to run the calculations
- **execPrefix** (*str*) – commands to go before the executable when run (ex. mpiexec nice -n 19 <executable>) (default = None)
- **execPostfix** (*str*) – commands to go after the executable when run (ex. <execPrefix> <executable> -ndiag 12 -nimage 2) (default = None)
- **holdFlag** (*bool*) – DEFUNCT. NEEDS REMOVAL
- **config** (*str*) – DEFUNCT. NEEDS REMOVAL

Returns None

run.**hyperfine**(*calcs*, *engine=''*, *execPrefix=None*, *execPostfix=None*, *holdFlag=True*, *config=None*, *isotope=()*)

Wrapper to set up GIPAW hyperfine calculation

Parameters **calcs** (*dict*) – Dictionary of dictionaries of calculations

Keyword Arguments

- **engine** (*str*) – executable that you are calling to run the calculations
- **execPrefix** (*str*) – commands to go before the executable when run (ex. mpiexec nice -n 19 <executable>) (default = None)
- **execPostfix** (*str*) – commands to go after the executable when run (ex. <execPrefix> <executable> -ndiag 12 -nimage 2) (default = None)
- **holdFlag** (*bool*) – DEFUNCT. NEEDS REMOVAL
- **config** (*str*) – DEFUNCT. NEEDS REMOVAL

Returns None

run.**nmr**(*calcs*, *engine=''*, *execPrefix=None*, *execPostfix=None*, *holdFlag=True*, *config=None*)

Wrapper to set up GIPAW NMR calculation

Parameters **calcs** (*dict*) – Dictionary of dictionaries of calculations

Keyword Arguments

- **engine** (*str*) – executable that you are calling to run the calculations
- **execPrefix** (*str*) – commands to go before the executable when run (ex. mpiexec nice -n 19 <executable>) (default = None)
- **execPostfix** (*str*) – commands to go after the executable when run (ex. <execPrefix> <executable> -ndiag 12 -nimage 2) (default = None)
- **holdFlag** (*bool*) – DEFUNCT. NEEDS REMOVAL
- **config** (*str*) – DEFUNCT. NEEDS REMOVAL

Returns None

run.**pdos**(*calcs*, *engine=''*, *execPrefix=None*, *execPostfix=''*, *holdFlag=True*, *config=None*)

Wrapper to set up DOS projection calculation

Parameters **calcs** (*dict*) – Dictionary of dictionaries of calculations

Keyword Arguments

- **engine** (*str*) – executable that you are calling to run the calculations
- **execPrefix** (*str*) – commands to go before the executable when run (ex. mpiexec nice -n 19 <executable>) (default = None)

- **execPostfix** (*str*) – commands to go after the executable when run (ex. <execPrefix> <executable> -ndiag 12 -nimage 2) (default = None)
- **holdFlag** (*bool*) – DEFUNCT. NEEDS REMOVAL
- **config** (*str*) – DEFUNCT. NEEDS REMOVAL

Returns None

run.**prep_fd**(*__submitNodeName__*, *oneCalc*, *ID*, *nrx1=2*, *nrx2=2*, *nrx3=2*, *innx=2*, *de=0.01*, *atom_sym=True*, *disp_sym=True*)

Generates input files for fd.x, fd_ifc.x, and matdyn.x for the finite difference phonon calculations.

Parameters

- **__submitNodeName__** (*str*) – String of hostname that cluster jobs should be submitted from
- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments

- **nrx1** (*int*) – supercell size for first primitive lattice vector
- **nrx2** (*int*) – supercell size for second primitive lattice vector
- **nrx3** (*int*) – supercell size for third primitive lattice vector
- **innx** (*int*) – how many differernt shifts in each direction for finite difference phonon calculation
- **de** (*float*) – amount to shift the atoms for finite differences

Returns None

run.**reset_logs**(*calcs*)

Removes log files from MTF directory

Parameters **calcs** (*dict*) – Dictionary of dictionaries of calculations

Keyword Arguments **None**

Returns None

run.**resubmit**(*calcs*)

Stages loaded calculation set to be resubmitted on a cluster

Parameters **calcs** (*dict*) – Dictionary of dictionaries of calculations

Keyword Arguments **None**

Returns None

run.**scf**(*calcs*, *engine=''*, *execPrefix=None*, *execPostfix=None*, *holdFlag=True*, *config=None*)

Wrapper to set up self-consitent calculation

Parameters **calcs** (*dict*) – Dictionary of dictionaries of calculations

Keyword Arguments

- **engine** (*str*) – executable that you are calling to run the calculations
- **execPrefix** (*str*) – commands to go before the executable when run (ex. mpiexec nice -n 19 <executable>) (default = None)
- **execPostfix** (*str*) – commands to go after the executable when run (ex. <execPrefix> <executable> -ndiag 12 -nimage 2) (default = None)

- **holdFlag** (*bool*) – DEFUNCT. NEEDS REMOVAL
- **config** (*str*) – DEFUNCT. NEEDS REMOVAL

Returns:

run.**submit**()
sets global __submit__flag__ so calculations will start when user script completes

Parameters **None** –

Keyword Arguments **None**

Returns None

run.**submitFirstCalcs__**(*calcs*)
Submits the first step of a calculation's pipeline

Parameters **calcs** (*dict*) – Dictionary of dictionaries of calculations

Keyword Arguments **None**

Returns None

run.**testOne**(*calcs*, *calcType='scf'*, *engine=''*, *execPrefix=None*, *execPostfix=None*, *holdFlag=True*, *config=None*)
Run all the calculation in the dictionary with a specific engine

Parameters **calcs** (*dict*) – Dictionary of dictionaries of calculations

Keyword Arguments

- **engine** (*str*) – executable that you are calling to run the calculations
- **execPrefix** (*str*) – commands to go before the executable when run (ex. mpiexec nice -n 19 <executable>) (default = None)
- **execPostfix** (*str*) – commands to go after the executable when run (ex. <execPrefix> <executable> -ndiag 12 -nimage 2) (default = None)

Returns None

run.**write_fdx_template**(*oneCalc*, *ID*, *nrx1=2*, *nrx2=2*, *nrx3=2*, *innx=2*, *de=0.01*, *atom_sym=True*, *disp_sym=True*)
Generates input files for fd.x, fd_ifc.x, and matdyn.x for the finite difference phonon calculations.

Parameters

- **oneCalc** (*dict*) – dictionary of one of the calculations
- **ID** (*str*) – ID of calculation

Keyword Arguments

- **nrx1** (*int*) – supercell size for first primitive lattice vector
- **nrx2** (*int*) – supercell size for second primitive lattice vector
- **nrx3** (*int*) – supercell size for third primitive lattice vector
- **innx** (*int*) – how many differernt shifts in each direction for finite difference phonon calculation
- **de** (*float*) – amount to shift the atoms for finite differences

Returns None

CHAPTER 6

scfuj module

scfuj.**WanT_bands**(*oneCalc*, *ID=None*, *eShift=10.0*, *nbnd=None*)
Make input files for WanT bands calculation

Parameters **calc_copy -- dictionary of dictionaries of calculations** (-) –

scfuj.**WanT_dos**(*oneCalc, ID=None, eShift=10.0, temperature=300.0, energy_range=[-10.0, 10.0]*)
Make input files for WanT bands calculation

Parameters **calc_copy -- dictionary of dictionaries of calculations** (-) –

scfuj.**WanT_epsilon**(*oneCalc, ID=None, eShift=10.0, temperature=300.0, energy_range=[0.1, 12.5]*)
Make input files for WanT bands calculation

Parameters **calc_copy -- dictionary of dictionaries of calculations** (-) –

scfuj.**__getPAOfilename**(*atom*, *PAOdir=None*)
Get the pseudopotential filename for the specific atomic species in input

Parameters **atom -- a string designating the atomic species you want the pseudofile name for** (-) –

Keyword Arguments **- pseudodir – the path of the directory containing pseudofiles**

scfuj.**__get_ham_xml**(*oneCalc*, *ID*)

scfuj.**__oneMinimizeCalcs**(*oneCalc*, *ID*, *config=None*, *pThresh=10.0*)
Get equilibrium volume using evfit to Murnaghan's EOS with 5 volumes in +/-20% of input volume

scfuj.**__oneScfprep**(*oneCalc*, *ID*, *paodir=None*)
Read a ref file (str), the dictionary defining the calculations, and the dictionary with the aflowkeys Create the dir tree for the run and return a dictionary dictAllcalcs with all the calculations. Store the dictionary in a log file

Parameters

- **allMTFVars -- all the variables that you want to make a list of combinations of calculations from** (-) –
- **refFile -- string that contains the input ref file file path** (-) –

Keyword Arguments

- **- pseudodir – path of the directory that contains your Pseudopotential files**

- **- paodir – path of the directory that contains pseudo atomic orbital files for the respective pseudo-potentials**
- **- calcs – Dictionary of dictionaries of calculations - this can be empty if it is the initial acbn0 run.**

scfuj.**__run**(*__submitNodeName__*, *oneCalc*, *ID*, *config=None*, *mixing=0.0*)

scfuj.**acbn0**(*oneCalc*, *projCalcID*, *byAtom=False*)

scfuj.**checkOscillation**(*ID*, *oneCalc*, *uThresh=0.001*)

scfuj.**chkSpinCalc**(*oneCalc*, *ID=None*)
Check whether an calculation is spin polarized or not.

Arguments:

–oneCalc : dictionary of a single calculation.

scfuj.**chk_species**(*elm*)

scfuj.**evCurveMinimize**(*calcs*, *config=None*, *pThresh=10.0*, *final_minimization='vc-relax'*)

scfuj.**getU_frmACBN0out**(*oneCalc*, *ID*, *byAtom=False*)

scfuj.**maketree**(*oneCalc*, *ID*, *paodir=None*)
Make the directoy tree and place in the input file there

Parameters **calcs -- Dictionary of dictionaries of calculations** (-) –

Keyword Arguments

- **- pseudodir – path of pseudopotential files directory**
- **- paodir - path of pseudoatomic orbital basis set**

scfuj.**nscf_nosym_noinv**(*oneCalc*, *ID=None*, *kpFactor=1.5*)
Add the ncsf input to each subdir and update the master dictionary

Parameters

- **calc_copy -- dictionary of one calculation** (-) –
- **kpFactor -- multiplicative factor for kpoints from SCF to DOS (default** (-) –

  2.

scfuj.**projwfc**(*oneCalc*, *ID=None*)
Run projwfc on each calculation

Parameters **oneCalc -- dictionary of a single calculation** (-) –

scfuj.**run**(*calcs*, *uThresh=0.001*, *nIters=20*, *mixing=0.0*)

scfuj.**run_transport**(*__submitNodeName__*, *oneCalc*, *ID*, *run_scf=True*, *run_transport_prep=True*, *run_bands=True*, *epsilon=True*, *temperature=300*)

scfuj.**scfprep**(*calcs*, *paodir=None*)

scfuj.**transport_prep**(*oneCalc*, *ID*)
sets up the environment do do scf->nscf->projwfc to get overlap for transport calcs

scfuj.**updateUvals**(*oneCalc*, *Uvals*, *ID=None*)
Modify scf input file to do a lda+u calculation.

Parameters

- **oneCalc -- Dictionary of one calculation** (-) –

- **Uvals -- Dictionary of Uvals** (-) –

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

p

plot, **??**
prep, **??**
pseudo, **??**

r

retr, **??**
run, **??**

s

scfuj, **??**