

---

# **easyInterface**

***Release 0.0.7***

**Simon Ward**

**Mar 12, 2020**



## EXAMPLE GALLERIES

<b>1</b>	<b>Features of easyInterface</b>	<b>3</b>
<b>2</b>	<b>Projects using easyInterface</b>	<b>5</b>
<b>3</b>	<b>Installation</b>	<b>7</b>
3.1	Install via <code>pip</code> . . . . .	7
3.2	Install as an easyInterface developer . . . . .	7
3.3	Main Contents . . . . .	7
<b>4</b>	<b>Indices and tables</b>	<b>25</b>
	<b>Python Module Index</b>	<b>27</b>
	<b>Index</b>	<b>29</b>





easyInterface is a library to interface crystallographic calculators to front end applications, jupyter notebooks and scripting interfaces.

The code of the project is on Github: [easyInterface](#)



## FEATURES OF EASYINTERFACE

easyInterface is a way of storing information about crystal structures, providing commonly used functions in an easy to use package. The data structure interfaces to crystallographic libraries, making a common way to calculate observable phenomena regardless of your choice of backend calculator. Currently we support:

- [Cryspy](#) - a crystallographic library for neutron data analysis.

With more interfaces coming.





## PROJECTS USING EASYINTERFACE

easyInterface is currently being used in the following projects:

- [easyDiffraction](#) - Scientific software for modelling and analysis of neutron diffraction data



## INSTALLATION

### 3.1 Install via pip

You can do a direct install via pip by using:

```
$ pip install easyInterface
```

### 3.2 Install as an easyInterface developer

You can get the latest development source from our [Github repository](#). You need `setuptools` installed in your system to install `easyInterface`. For example, you can do:

```
$ git clone https://github.com/easyDiffraction/easyInterface
$ cd easyInterface
$ pip install -r requirements.txt
$ pip install -e .
```

### 3.3 Main Contents

#### 3.3.1 Introduction to easyInterface

Here we can see some examples of `easyInterface` in action

---

**Note:** Click [here](#) to download the full example code or to run this example in your browser via Binder

---

#### Creating a QT interface

This demonstrates an example of how to load an example and create a QT interface to a `crispy` calculator. Information about the project is then displayed.

```
# import os
#
# from easyInterface.Utils.Helpers import getExamplesDir
# from easyInterface.Diffraction.Calculators import CrispyCalculator
# from easyInterface.Diffraction.QtInterface import QtCalculatorInterface
```

(continues on next page)

(continued from previous page)

```
#
# data_dir = getExamplesDir()
# main_rcif = os.path.join(data_dir, 'Fe3O4_powder-1d_neutrons-pol_5C1(LLB)', 'main.
↪cif')
# calculator = CryspyCalculator(main_rcif)
#
# interface = QtCalculatorInterface(calculator, None)
#
# print(interface.project_dict)
#
# print(interface.phasesIds())
#
# print(interface.getPhase(interface.phasesIds()[0]))
```

**Total running time of the script:** ( 0 minutes 0.000 seconds)

### 3.3.2 Scripted Examples

This section gathers examples which don't produce any figures. These examples show the basic features of easyInterface.

---

**Note:** Click [here](#) to download the full example code or to run this example in your browser via Binder

---

#### Creating a interface

This demonstrates an example of how to load an example and create a interface to a cryspy calculator. Information about the project is then displayed.

```
import os
from easyInterface.Uutils.Helpers import getExamplesDir
from easyInterface.Diffraction.Calculators import CryspyCalculator
from easyInterface.Diffraction.Interface import CalculatorInterface

data_dir = getExamplesDir()
main_rcif = os.path.join(data_dir, 'Fe3O4_powder-1d_neutrons-pol_5C1(LLB)', 'main.cif
↪')
calculator = CryspyCalculator(main_rcif)

interface = CalculatorInterface(calculator)

print(interface.project_dict)

print(interface.phasesIds())

print(interface.getPhase(interface.phasesIds()[0]))
```

**Total running time of the script:** ( 0 minutes 0.000 seconds)

---

**Note:** Click [here](#) to download the full example code or to run this example in your browser via Binder

---

## Performing a fit

This demonstrates an example of how to load an example and create an interface to a crspsy calculator and then fit a value.

```
import os
from easyInterface.Utils.Helpers import getExamplesDir
from easyInterface.Diffraction.Calculators import CrspsyCalculator
from easyInterface.Diffraction.Interface import CalculatorInterface

data_dir = getExamplesDir()
main_rcif = os.path.join(data_dir, 'PbSO4_powder-1d_neutrons-unpol_D1A(ILL)', 'main.
↪cif')

calculator = CrspsyCalculator(main_rcif)

interface = CalculatorInterface(calculator)

print(interface.project_dict)

phase_ids = interface.phasesIds()

print(phase_ids)

phase = interface.getPhase(phase_ids[0])
phase['phasename'] = 'PbSO5'
interface.addPhase(phase)
interface.removePhase('PbSO5')

phase = interface.getPhase(phase_ids[0])
interface.setPhaseValue(phase_ids[0], ['atoms', 'Pb', 'fract_x'], 0.18)
interface.setPhases(phase)
print(phase)

interface.setPhaseRefine(phase_ids[0], ['atoms', 'Pb', 'fract_x'], True)

calc = interface.getCalculations()
print(calc)

res = interface.refine()

print(res)
```

Total running time of the script: ( 0 minutes 0.000 seconds)

## 3.3.3 Interface and Calculators

### easyInterface Interface

**class** easyInterface.Diffraction.Interface.CalculatorInterface (*calculator*)

Interface to calculators in the *easyInterface.Diffraction.Calculator* class.

**addExperiment** (*experiment*)

Add an experiment to the list of experiments in both the project dict and the calculator.

**Parameters** **experiment** (*Experiment*) – Experiment object to be added to the system.

**Return type** NoReturn

**addExperimentDefinition** (*exp\_path*)

Add an experiment to be simulated from a cif file. Note that this will not have any crystallographic phases associated with it.

**Parameters** **exp\_path** (*str*) – Path to a experiment file (*.cif*)

**Return type** NoReturn

**addPhase** (*phase*)

Add a new phase from an easyInterface phase object to the list of existing crystal phases in the calculator.

**Parameters** **phase** (*Phase*) – New phase to be added to the phase list.

**Return type** NoReturn

**addPhaseDefinition** (*phase\_path*)

Add new phases from a cif file to the list of existing crystal phases in the calculator.

**Parameters** **phase\_path** (*str*) – Path to a phase definition file (*.cif*)

Example:

```
interface = CalculatorInterface(calculator)
phase_path = '~/Experiments/new_phase.cif'
interface.addPhaseDefinition(phase_path)
```

**Return type** NoReturn

**addPhaseToExp** (*exp\_name, phase\_name, scale=0.0*)

Link a phase in the project dictionary to an experiment in the project dictionary. Links in the calculator will also be made.

**Parameters**

- **exp\_name** (*str*) – The name of the experiment
- **phase\_name** (*str*) – The name of the phase to be associated with the experiment
- **scale** (*float*) – The scale of the crystallographic phase in the experimental system.

**Raises** **KeyError** – If the exp\_name or phase\_name are unknown

**Return type** NoReturn

**asCifDict** ()

Converts the project dictionary into a *cif* structure.

**Return type** *str*

**Returns** Project dictionary as a string encoded to the cif specification.

**asDict** ()

Converts the project dictionary info a standard python dictionary. If there is an error then an empty dictionary is returned.

**Return type** *dict*

**Returns** Python dictionary of the project dictionary.

**canRedo** ()

Informs on if the project dictionary can have redo() called. Typically called after an undo function call.

**Return type** *bool*

**Returns** Can or Can't redo the project dictionary.

**canUndo()**

Inform on if the project dictionary can have undo() called.

**Return type** bool

**Returns** Can or Can't undo the project dictionary.

**clearUndoStack()**

Resets the Undo/Redo stack of the project dictionary.

ALL PREVIOUS UNDO/REDO EDITS WILL BE LOST

**Return type** NoReturn

**experimentsCount()**

Get the number of experiments in the project dictionary.

**Return type** int

**Returns** number of experiments in the project dictionary.

**experimentsIds()**

Returns labels of the experiments in the project dictionary.

**Return type** List[str]

**property final\_chi\_square**

Calculates the final chi squared of the simulation. Where the final chi squared is the chi squared divided by the number of data points.

**Return type** float

**Returns** Final chi squared

**getCalculation(calculation\_name)**

Returns a specified calculation from the project dictionary.

**Parameters** **calculation\_name** (str) – Name of the calculation to be returned.

**Raises** **KeyError** – If the calculation\_name is not known.

**Return type** Calculation

**Returns** Calculation requested.

**getCalculations()**

Returns all calculations in the project dictionary. Calculations will be updated if members of the phases or experiments section of the project dictionary has been modified.

**Return type** Calculations

**Returns** Calculations object containing all calculations.

**getDictByPath(keys)**

Returns an object in the project dictionary by the path to the object.

**Parameters** **keys** (List[str]) – Path to the object in the project dictionary

**Return type** Any

**Returns** Object from the project dictionary.

**Raises** **KeyError** – The supplied keys do not return an object in the project dictionary

**getExperiment** (*experiment\_name*)

Returns a experiment from the project dictionary by name if one is supplied. If the experiment name is None then all experiments are returned. If the experiment name does not exist KeyError is thrown.

**Parameters** **experiment\_name** (Optional[str]) – Name of the experiment to be returned or None for all experiments

**Return type** *Experiment*

**Returns** Copy of the project dictionaries phase object with name experiment\_name

**Raises** **KeyError** – The supplied key is not a valid experiment name

**getPhase** (*phase\_name*)

Returns a phase from the project dictionary by name if one is supplied. If the phase name is None then all phases are returned. If the phase name does not exist KeyError is thrown.

**Parameters** **phase\_name** (Optional[str]) – Name of the phase to be returned or None for all phases

**Return type** *Phase*

**Returns** Copy of the project dictionaries phase object with name phase\_name

**Raises** **KeyError** – The supplied key is not a valid phase name

**name** ()

Returns the name of the current project.

**Return type** str

**Returns** Name of the current project

**phasesCount** ()

Get the number of phases in the project dictionary.

**Return type** int

**Returns** number of phases in the project dictionary.

**phasesIds** ()

Get the labels of the phases in the project dictionary.

**Return type** List[str]

**Returns** labels of the phases in the project dictionary.

**redo** ()

Perform an redo operation on the project dictionary.

**Return type** NoReturn

**refine** ()

Perform a refinement on parameters which are marked in the project dictionary. If the refinement fails then only the “refinement\_message” will be returned in the results dictionary with an explanation of the error.

**Return type** dict

**Returns** Refinement results of the following fields: “num\_refined\_parameters”, “refinement\_message”, “nfev”, “nit”, “njev”, “final\_chi\_sq”

**removeExperiment** (*experiment\_name*)

Remove a experiment from both the project dictionary and the calculator.

**Parameters** **experiment\_name** (str) – Name of the experiment to be removed.

**Return type** NoReturn



**removePhase** (*phase\_name*)

Remove a phase of a given name from the dictionary and the calculator object.

**Parameters** **phase\_name** (*str*) – name of the phase to be removed.

**Return type** NoReturn

**removePhaseFromExp** (*exp\_name, phase\_name*)

Remove the link between an experiment and a crystallographic phase. Links in the calculator will also be removed.

**Parameters**

- **exp\_name** (*str*) – The name of the experiment.
- **phase\_name** (*str*) – The name of the phase to be removed.

**Raises** **KeyError** – If the *exp\_name* or *phase\_name* are unknown

**Return type** NoReturn

**saveCifs** (*save\_dir*)

Write project cif files (*main.cif*, *experiments.cif* and *phases.cif*) to a user supplied directory. This contains all information needed to recreate the project dictionary.

**Parameters** **save\_dir** (*str*) – Directory to where the project cif files should be saved.

**Return type** NoReturn

**setCalculatorFromProject** ()

Resets the project phases and experiments fields of the project dictionary from the calculator.

**Return type** NoReturn

**setDictByPath** (*keys, value*)

Set an object in the project dictionary by a key path.

**Parameters**

- **keys** (*List[str]*) – Path to the object to be modified/created
- **value** (*Any*) – Value to be set at the key path

**Return type** NoReturn

**setExperiment** (*experiment*)

Set an experiment to the project dictionary. If an experiment by the same name exists, the necessary changes will be propagated. If it does not exist, then it will be added to the project dictionary.

**Parameters** **experiment** (*Experiment*) – Experiment container with experimental information

**Raises** **TypeError** – If the input isn't an *Experiment*.

**Return type** NoReturn

**setExperimentDefinition** (*exp\_path*)

Set an experiment/s to be simulated from a cif file. Note that this will not have any crystallographic phases associated with it.

**Parameters** **exp\_path** (*str*) – Path to a experiment file (*.cif*)

**Return type** NoReturn

**setExperimentDefinitionFromString** (*exp\_cif\_string*)

Set an experiment/s to be simulated from a string. Note that this will not have any crystallographic phases associated with it.

**Parameters** `exp_cif_string` (`str`) – String containing the contents of an experiment file (.cif)

**Return type** `NoReturn`

**setExperimentRefine** (`experiment`, `key`, `value=True`)

Shortcut for setting the refinement key for items in the experiment list.

**Parameters**

- **experiment** (`str`) – Name of experiment to be modified
- **key** (`List[str]`) – Location of element to be modified in the named experiment.
- **value** (`bool`) – Should the parameter specified by above be refined

**Raises** **KeyError** – If experiment is unknown

**Return type** `NoReturn`

**setExperimentValue** (`experiment`, `key`, `value`)

Shortcut for setting the value key for items in the experiment list.

**Parameters**

- **experiment** (`str`) – Name of experiment to be modified
- **key** (`List[str]`) – Location of element to be modified in the named experiment.
- **value** – New value of the parameter specified by above

**Raises** **KeyError** – If experiment is unknown

**setExperiments** (`experiments`)

Overwrite all experiments in the project dictionary with supplied experiments.

**Parameters** **experiments** (`Union[Experiment, Experiments]`) – Experiments container with experimental information

**Raises** **TypeError** – If the input isn't an *Experiments* or 'Experiment'.

**setPhase** (`phase`)

Modify a phase in the calculator. The phase will be added if it does not currently exist.

**Parameters** **phase** (*Phase*) – easyInterface phase object to be added.

**Raises** **TypeError** – If the phase object is not a easyInterface phase object.

**Return type** `NoReturn`

**setPhaseDefinition** (`phase_path`)

Parse a phases cif file and replace existing crystal phases

**Parameters** **phase\_path** (`str`) – Path to new phase definition file (.cif)

Example:

```
interface = CalculatorInterface(calculator)
phase_path = '~/Experiments/phases.cif'
interface.setPhaseDefinition(phase_path)
```

**Return type** `NoReturn`

**setPhaseRefine** (`phase`, `key`, `value=True`)

Shortcut for setting the refinement key for items in the phase list.

**Parameters**

- **phase** (`str`) – Name of phase to be modified
- **key** (`List[str]`) – Location of element to be modified in the named phase.
- **value** (`bool`) – Should the parameter specified by above be refined

**Raises** **KeyError** – If phase is unknown

**Return type** `NoReturn`

**setPhaseValue** (*phase, key, value*)

Shortcut for setting the value key for items in the phase list.

**Parameters**

- **phase** (`str`) – Name of phase to be modified
- **key** (`List[str]`) – Location of element to be modified in the named phase.
- **value** – New value of the parameter specified by above

**Raises** **KeyError** – If phase is unknown

**Return type** `NoReturn`

**setPhases** (*phases*)

Set the phases in the calculator to an easyInterface phases object. If a phase in the supplied phases exists then the phase will be modified, if not, it will be added.

**Parameters** **phases** (`Union[Phase, Phases]`) – phases to be added to the calculator.

**Raises** **TypeError** – If the phase object is not a easyInterface phase/phases object or dictionary object.

**Return type** `NoReturn`

**setProjectFromCalculator** ()

Sets the project dictionary from the calculator given on initialisation. Calling this function will regenerate the project dictionary and changes may be lost.

**Return type** `NoReturn`

**undo** ()

Perform an undo operation on the project dictionary.

**Return type** `NoReturn`

**updateCalculations** ()

Calculate all experiments and populate the calculations field in the project dictionary. Note that this will only occur if a member of the phases or experiments section of the project dictionary has been modified since the last call to *updateCalculations*.

**Return type** `NoReturn`

**updateExperiments** ()

Synchronise the experiments portion of the project dictionary from the calculator.

**Return type** `NoReturn`

**updatePhases** ()

Synchronise the phases in project dictionary by queering the calculator object.

**Return type** `NoReturn`

**writeExpCif** (*save\_dir*)

Write the *experiments.cif* where all experiments in the project dictionary are saved to file. This includes the instrumental parameters and which phases are in the experiment/s

**Parameters** **save\_dir** (*str*) – Directory to where the experiment cif file should be saved.

**Return type** NoReturn

**writeMainCif** (*save\_dir*)

Write the *main.cif* where links to the experiments and phases are stored and other generalised project information.

**Parameters** **save\_dir** (*str*) – Directory to where the main cif file should be saved.

**Return type** NoReturn

**writePhaseCif** (*save\_dir*)

Write the *phases.cif* where all phases in the project dictionary are saved to file. This cif file should be compatible with other crystallographic software.

**Parameters** **save\_dir** (*str*) – Directory to where the phases cif file should be saved.

**Return type** NoReturn

## easyInterfaces Project Dictionary

```
class easyInterface.Diffraction.Interface.ProjectDict (interface, app, calculator,  
                                                    info, phases, experiments,  
                                                    calculations)
```

This class deals with the creation and modification of the main project dictionary.

**classmethod** **default** ()

Create a default and empty project dictionary

**Return type** *ProjectDict*

**Returns** Default project dictionary with undo/redo functionality

**classmethod** **fromPars** (*experiments, phases, calculations={}*)

Create a main project dictionary from phases and experiments.

**Parameters**

- **calculations** (Union[*Calculations, Calculation, List[Calculation], None*]) –
- **experiments** (Union[*Experiments, Experiment, List[Experiment]*]) – A collection of experiments to be compared to calculations
- **phases** (Union[*Phases, Phase, List[Phase]*]) – A Collection of crystallographic phases to be calculated

**Return type** *ProjectDict*

**Returns** Project dictionary with undo/redo

## easyInterface Crispy Calculator

```
class easyInterface.Diffraction.Calculators.CrispyCalculator (main_rcif_path=None)
```

**addExpDefinitionFromString** (*exp\_rcif\_content*)

Set an experiment/s to be simulated from a string. Note that this will not have any crystallographic phases associated with it.

**Parameters** **exp\_rcif\_content** (*str*) – String containing the contents of an experiment file (*.cif*)

**Return type** NoReturn

**addExpsDefinition** (*exp\_path*)

Add an experiment to be simulated from a cif file. Note that this will not have any crystallographic phases associated with it.

**Parameters** **exp\_path** (*str*) – Path to a experiment file (*.cif*)

**Return type** NoReturn

**addPhaseDefinition** (*phases\_path*)

Add new phases from a cif file to the list of existing crystal phases in the calculator.

**Parameters** **phases\_path** (*str*) – Path to a phase definition file (*.cif*)

**Return type** NoReturn

**asCifDict** ()

...

**Return type** dict

**getCalculations** ()

Returns all calculations from the calculator object.

**Return type** *Calculations*

**getExperiments** ()

Returns all experiments from the calculator object.

**Return type** *Experiments*

**getPhases** ()

Returns all phases from the calculator object.

**Return type** *Phases*

**readPhaseDefinition** (*phases\_path*)

Parse the relevant phases file and update the corresponding model

**Return type** Optional[Tuple[*Phase*, Phase]]

**refine** ()

refinement ...

**Return type** Tuple[dict, dict]

**removeExpsDefinition** (*experiment\_name*)

Remove a experiment from both the project dictionary and the calculator.

**Parameters** **experiment\_name** (*str*) – Name of the experiment to be removed.

**Return type** NoReturn

**removePhaseDefinition** (*phase\_name*)

Remove a phase of a given name from the calculator object.

**Parameters** **phase\_name** (*str*) – name of the phase to be removed.

**Return type** NoReturn

**saveCifs** (*save\_dir*, *filename*='main.cif', *exp\_name*='experiments.cif', *phase\_name*='phases.cif')

Write project cif files (*main.cif*, *experiments.cif* and *phases.cif*) to a user supplied directory. This contains all information needed to recreate the calculator object.

**Parameters**

- **phase\_name** (*str*) – What to call the phases file.
- **exp\_name** (*str*) – What to call the experiments file.
- **filename** (*str*) – What to call the main file.
- **save\_dir** (*str*) – Directory to where the main cif file should be saved.

**Return type** NoReturn

**setExperiments** (*experiments*)

Set experiments (Experimental data tab in GUI)

**Return type** NoReturn

**setExpsDefinition** (*exp\_path*)

Set an experiment/s to be simulated from a cif file. Note that this will not have any crystallographic phases associated with it.

**Parameters** **exp\_path** (*str*) – Path to a experiment file (.cif)

**Return type** NoReturn

**setObjFromProjectDicts** (*phases*, *experiments*)

Set all the cryspy parameters from project dictionary

**Return type** NoReturn

**setPhaseDefinition** (*phases\_path*)

Parse a phases cif file and replace existing crystal phases

**Parameters** **phases\_path** (*str*) – Path to new phase definition file (.cif)

**Return type** NoReturn

**setPhases** (*phases*)

Set phases (sample model tab in GUI)

**Return type** NoReturn

**writeExpCif** (*save\_dir*, *exp\_name*='experiments.cif')

Write the *experiments.cif* where all experiments in the calculator are saved to file. This includes the instrumental parameters and which phases are in the experiment/s

**Parameters**

- **exp\_name** (*str*) – What to call the experiments file.
- **save\_dir** (*str*) – Directory to where the experiment cif file should be saved.

**Return type** NoReturn

**writeMainCif** (*save\_dir*, *filename*='main.cif', *exp\_filename*='experiments.cif',  
*phase\_filename*='phases.cif')

Write the *main.cif* where links to the experiments and phases are stored and other generalised project information.

**Parameters**

- **phase\_filename** (*str*) – What to call the phases file.
- **exp\_filename** (*str*) – What to call the experiments file.

- **filename** (str) – What to call the main file.
- **save\_dir** (str) – Directory to where the main cif file should be saved.

**Return type** NoReturn

**writePhaseCif** (save\_dir, phase\_name='phases.cif')

Write the *phases.cif* where all phases in the calculator are saved to file. This cif file should be compatible with other crystallographic software.

**Parameters**

- **phase\_name** (str) – What to call the phases file.
- **save\_dir** (str) – Directory to where the phases cif file should be saved.

**Return type** NoReturn

### 3.3.4 Container Classes

#### Phase Classes

```
class easyInterface.Diffraction.DataClasses.PhaseObj.Atom.ADP (u_11,      u_22,
                                                             u_33, u_12, u_13,
                                                             u_23)
```

Data store for Atom site anisotropic displacement parameters

```
class easyInterface.Diffraction.DataClasses.PhaseObj.Atom.Atom (atom_site_label,
                                                                type_symbol,
                                                                scat_length_neutron,
                                                                fract_x, fract_y,
                                                                fract_z,      oc-
                                                                cupancy,
                                                                adp_type,
                                                                U_iso_or_equiv,
                                                                ADp, MSp)
```

Storage for details about an atom

```
classmethod default (atom_site_label)
```

Default constructor for an atom given a unique name in the phase

**Parameters** **atom\_site\_label** (str) – The atoms unique name in the phase

**Return type** *Atom*

**Returns** Default atom with a given name

```
classmethod fromPars (atom_site_label, type_symbol, scat_length_neutron, fract_x, fract_y,
                                                                fract_z, occupancy, adp_type, U_iso_or_equiv, ADp=None, MSp=None)
```

Atom constructor from parameters

**Parameters**

- **atom\_site\_label** (str) – The unique name of the atom in the phase
- **type\_symbol** (str) – The type of atom
- **scat\_length\_neutron** (float) – Neutron scattering length
- **fract\_x** (float) – X position
- **fract\_y** (float) – Y position

- **fract\_z** (float) – Z position
- **occupancy** (float) – Site occupancy
- **adp\_type** (str) – ADP type code
- **U\_iso\_or\_equiv** (float) – Isotropic atomic displacement parameter

**Return type** *Atom*

**Returns** Fully formed atom data store

**classmethod** **fromXYZ** (*atom\_site\_label*, *type\_symbol*, *x*, *y*, *z*)

Construct an atom from name, type and position

**Parameters**

- **atom\_site\_label** (str) – The atoms unique name in the phase
- **type\_symbol** (str) – The type of atom
- **x** (float) – X position
- **y** (float) – Y position
- **z** (float) – Z position

**Return type** *Atom*

**Returns** Atom with name type and position filled in

**class** `easyInterface.Diffraction.DataClasses.PhaseObj.Atom.Atoms` (*atoms*)

Container for multiple atoms

**class** `easyInterface.Diffraction.DataClasses.PhaseObj.Atom.MSP` (*MSPtype*, *chi\_11*,  
*chi\_22*, *chi\_33*,  
*chi\_12*, *chi\_13*,  
*chi\_23*)

Data store for Atom site magnetic susceptibility parameters

**class** `easyInterface.Diffraction.DataClasses.PhaseObj.Cell.Cell` (*length\_a*,  
*length\_b*,  
*length\_c*, *angle\_alpha*,  
*angle\_beta*,  
*angle\_gamma*)

Container for crystallographic unit cell parameters

**classmethod** **default** ()

Default constructor for a crystallographic unit cell

**Return type** *Cell*

**Returns** Default crystallographic unit cell container

**classmethod** **fromPars** (*length\_a*, *length\_b*, *length\_c*, *angle\_alpha*, *angle\_beta*, *angle\_gamma*)

Constructor of a crystallographic unit cell when parameters are known

**Parameters**

- **length\_a** (float) – Unit cell length a
- **length\_b** (float) – Unit cell length b
- **length\_c** (float) – Unit cell length c
- **angle\_alpha** (float) – Unit cell angle alpha



- **angle\_beta** (float) – Unit cell angle beta
- **angle\_gamma** (float) – Unit cell angle gamma

**Return type** *Cell*

**Returns**

**class** easyInterface.Diffraction.DataClasses.PhaseObj.Phase.**Phase** (*name, space-group, cell, atoms, sites*)

Container for crystallographic phase information

**classmethod default** (*name*)

Default constructor for a crystallographic phase with a given name

**Return type** *Phase*

**Returns** Default empty phase with a name

**class** easyInterface.Diffraction.DataClasses.PhaseObj.Phase.**Phases** (*phases*)

Container for multiple phases

**renamePhase** (*old\_phase\_name, new\_phase\_name*)

Easy method of renaming a phase

**Parameters**

- **old\_phase\_name** (str) – phase name to be changed
- **new\_phase\_name** (str) – new phase name

**Return type** NoReturn

**class** easyInterface.Diffraction.DataClasses.PhaseObj.SpaceGroup.**SpaceGroup** (*crystal\_system, space\_group\_name\_Hi, space\_group\_IT\_number, origin\_choice*)

## Data Classes

**class** easyInterface.Diffraction.DataClasses.DataObj.Calculation.**BraggPeaks** (*bragg\_peaks*)

Container for multiple calculations

**class** easyInterface.Diffraction.DataClasses.DataObj.Calculation.**CalculatedPattern** (*x, y\_calc, y\_diff\_lower, y\_diff\_upper*)

Storage container for a calculated pattern

**class** easyInterface.Diffraction.DataClasses.DataObj.Calculation.**Calculation** (*name, bragg\_peaks, calculated\_pattern, limits*)

Storage container for calculations

**class** easyInterface.Diffraction.DataClasses.DataObj.Calculation.**Calculations** (*calculations*)

Container for multiple calculations

```
class easyInterface.Diffraction.DataClasses.DataObj.Calculation.CrystalBraggPeaks (name,  
                                                                    h,  
                                                                    k,  
                                                                    l,  
                                                                    ttheta)
```

Generator for HKL reflections and corresponding two theta.

```
class easyInterface.Diffraction.DataClasses.DataObj.Calculation.Limits (y_obs_lower=-  
                                                                    inf,  
                                                                    y_obs_upper=inf,  
                                                                    y_diff_upper=inf,  
                                                                    y_diff_lower=-  
                                                                    inf,  
                                                                    x_calc=None,  
                                                                    y_calc=None)
```

Generator for limits of a dataset

```
class easyInterface.Diffraction.DataClasses.DataObj.Experiment.Background (ttheta,  
                                                                    in-  
                                                                    ten-  
                                                                    sity)
```

Data store for the background data parameters

**classmethod default** ()

Default constructor for a background point

**Return type** *Background*

**Returns** Default background data object

**classmethod fromPars** (*ttheta, intensity*)

Constructor for background when two theta and intensity are known

**Parameters**

- **ttheta** (float) – Two Theta angle in degrees
- **intensity** (float) – Value for intensity

**Return type** *Background*

**Returns** Background data dict

```
class easyInterface.Diffraction.DataClasses.DataObj.Experiment.Backgrounds (backgrounds)  
Store for a collection of background points
```

```
class easyInterface.Diffraction.DataClasses.DataObj.Experiment.Experiment (name,  
                                                                    wave-  
                                                                    length,  
                                                                    off-  
                                                                    set,  
                                                                    phase,  
                                                                    back-  
                                                                    ground,  
                                                                    res-  
                                                                    o-  
                                                                    lu-  
                                                                    tion,  
                                                                    mea-  
                                                                    sured_pattern)
```

Experimental details data container

**classmethod default** (*name*)

Default constructor for an Experiment

**Parameters** *name* (*str*) – What the experiment should be called

**Return type** *Experiment*

**Returns** Default empty experiment

**classmethod fromPars** (*name*, *wavelength*, *offset*, *scale*, *background*, *resolution*, *measured\_pattern*)

Constructor of experiment from parameters

**Parameters**

- **name** (*str*) – What the experiment should be called
- **wavelength** (*float*) – Experimental wavelength
- **offset** (*float*) – Experimental offset
- **scale** (*float*) – Scale parameter
- **background** (*Backgrounds*) – Background model
- **resolution** (*Resolution*) – Resolution model
- **measured\_pattern** (*MeasuredPattern*) – The Measured data

**Return type** *Experiment*

**Returns** Experiment from parameters

**class** easyInterface.Diffraction.DataClasses.DataObj.Experiment.**ExperimentPhase** (*name*, *scale*)

Storage container for the Experimental Phase details

**classmethod default** (*name*)

Default experimental phase data container

**Return type** *ExperimentPhase*

**Returns** Default experimental phase data container

**classmethod fromPars** (*name*, *scale*)

Parameter initialised experimental phase data container

**Return type** *ExperimentPhase*

**Returns** Set experimental phase data container

**class** easyInterface.Diffraction.DataClasses.DataObj.Experiment.**ExperimentPhases** (*experiment\_phases*)

Storage of multiple phase markers associated with experiments

**class** easyInterface.Diffraction.DataClasses.DataObj.Experiment.**Experiments** (*experiments*)

Container for multiple experiments

**class** easyInterface.Diffraction.DataClasses.DataObj.Experiment.**MeasuredPattern** (*x*, *y\_obs*, *sy\_obs*, *y\_obs\_up=None*, *sy\_obs\_up=None*, *y\_obs\_down=None*, *sy\_obs\_down=None*)

Storage container for measured patterns

**classmethod default** (*polarised=False*)

Default constructor for measured data container.

**Parameters** **polarised** (bool) – Should the container be initialised as a polarised data container?

**Returns** Empty data container

**property isPolarised**

Is the measured data of a polarised type?

**Return type** bool

**Returns** True if it is from a polarised measurement, false otherwise

**property y\_obs\_lower**

Lower data confidence bound.

**Return type** list

**Returns** value of lower confidence bound

**property y\_obs\_upper**

Upper data confidence bound.

**Return type** list

**Returns** value of upper confidence bound

**class** easyInterface.Diffraction.DataClasses.DataObj.Experiment.**Resolution** (*u*,  
*v*,  
*w*,  
*x*,  
*y*)

Data store for the resolution parameters

**classmethod default** ()

Default constructor for the resolution dict

**Return type** *Resolution*

**Returns** Default resolution dict

**classmethod fromPars** (*u, v, w, x, y*)

Constructor when resolution parameters are known

**Parameters**

- **u** (float) – resolution parameter u
- **v** (float) – resolution parameter v
- **w** (float) – resolution parameter w
- **x** (float) – resolution parameter x
- **y** (float) – resolution parameter y

**Return type** *Resolution*

**Returns** Resolution dictionary with values set

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### e

`easyInterface.Diffraction.DataClasses.DataObj.Calculation,`  
21  
`easyInterface.Diffraction.DataClasses.DataObj.Experiment,`  
22  
`easyInterface.Diffraction.DataClasses.PhaseObj.Atom,`  
19  
`easyInterface.Diffraction.DataClasses.PhaseObj.Cell,`  
20  
`easyInterface.Diffraction.DataClasses.PhaseObj.Phase,`  
21  
`easyInterface.Diffraction.DataClasses.PhaseObj.SpaceGroup,`  
21





## A

addExpDefinitionFromString() (*easyInterface.Diffraction.Calculators.CryspyCalculator method*), 16  
 addExperiment() (*easyInterface.Diffraction.Interface.CalculatorInterface method*), 9  
 addExperimentDefinition() (*easyInterface.Diffraction.Interface.CalculatorInterface method*), 10  
 addExpsDefinition() (*easyInterface.Diffraction.Calculators.CryspyCalculator method*), 17  
 addPhase() (*easyInterface.Diffraction.Interface.CalculatorInterface method*), 10  
 addPhaseDefinition() (*easyInterface.Diffraction.Calculators.CryspyCalculator method*), 17  
 addPhaseDefinition() (*easyInterface.Diffraction.Interface.CalculatorInterface method*), 10  
 addPhaseToExp() (*easyInterface.Diffraction.Interface.CalculatorInterface method*), 10  
 ADP (class in *easyInterface.Diffraction.DataClasses.PhaseObj.Atom*), 19  
 asCifDict() (*easyInterface.Diffraction.Calculators.CryspyCalculator method*), 17  
 asCifDict() (*easyInterface.Diffraction.Interface.CalculatorInterface method*), 10  
 asDict() (*easyInterface.Diffraction.Interface.CalculatorInterface method*), 10  
 Atom (class in *easyInterface.Diffraction.DataClasses.PhaseObj.Atom*), 19  
 Atoms (class in *easyInterface.Diffraction.DataClasses.PhaseObj.Atom*),

20

## B

Background (class in *easyInterface.Diffraction.DataClasses.DataObj.Experiment*), 22  
 Backgrounds (class in *easyInterface.Diffraction.DataClasses.DataObj.Experiment*), 22  
 BraggPeaks (class in *easyInterface.Diffraction.DataClasses.DataObj.Calculation*), 21

## C

CalculatedPattern (class in *easyInterface.Diffraction.DataClasses.DataObj.Calculation*), 21  
 Calculation (class in *easyInterface.Diffraction.DataClasses.DataObj.Calculation*), 21  
 Calculations (class in *easyInterface.Diffraction.DataClasses.DataObj.Calculation*), 21  
 CalculatorInterface (class in *easyInterface.Diffraction.Interface*), 9  
 canRedo() (*easyInterface.Diffraction.Interface.CalculatorInterface method*), 10  
 canUndo() (*easyInterface.Diffraction.Interface.CalculatorInterface method*), 11  
 Cell (class in *easyInterface.Diffraction.DataClasses.PhaseObj.Cell*), 20  
 clearUndoStack() (*easyInterface.Diffraction.Interface.CalculatorInterface method*), 11  
 CryspyCalculator (class in *easyInterface.Diffraction.Calculators*), 16  
 CrystalBraggPeaks (class in *easyInterface.Diffraction.DataClasses.DataObj.Calculation*), 21

## D

default () (easyInterface.Diffraction.DataClasses.DataObj.Experiment.Experiment class method), 22

experimentsCount () (easyInterface.Diffraction.Interface.CalculatorInterface method), 11

experimentsIds () (easyInterface.Diffraction.Interface.CalculatorInterface method), 11

default () (easyInterface.Diffraction.DataClasses.DataObj.Experiment.Experiment class method), 22

default () (easyInterface.Diffraction.DataClasses.DataObj.Experiment.Experiment class method), 23

face.Diffraction.Interface.CalculatorInterface property), 11

default () (easyInterface.Diffraction.DataClasses.DataObj.Experiment.Experiment class method), 23

face.Diffraction.DataClasses.DataObj.Experiment.Experiment class method), 22

default () (easyInterface.Diffraction.DataClasses.DataObj.Experiment.Experiment class method), 24

face.Diffraction.DataClasses.DataObj.Experiment.Experiment class method), 23

default () (easyInterface.Diffraction.DataClasses.PhaseObj.Atom.Atom class method), 19

face.Diffraction.DataClasses.DataObj.Experiment.Experiment class method), 23

default () (easyInterface.Diffraction.DataClasses.PhaseObj.Cell.Cell class method), 20

face.Diffraction.DataClasses.DataObj.Experiment.Resolution class method), 24

default () (easyInterface.Diffraction.DataClasses.PhaseObj.Phase.Phase class method), 21

face.Diffraction.DataClasses.PhaseObj.Atom.Atom class method), 19

default () (easyInterface.Diffraction.Interface.ProjectDict class method), 16

face.Diffraction.DataClasses.PhaseObj.Cell.Cell class method), 20

## E

easyInterface.Diffraction.DataClasses.DataObj.Calculation (module), 21

easyInterface.Diffraction.DataClasses.DataObj.Experiment (module), 22

easyInterface.Diffraction.DataClasses.PhaseObj.Atom (module), 19

easyInterface.Diffraction.DataClasses.PhaseObj.Cell (module), 20

easyInterface.Diffraction.DataClasses.PhaseObj.Phase (module), 21

easyInterface.Diffraction.DataClasses.PhaseObj.SpaceGroup (module), 21

Experiment (class in easyInterface.Diffraction.DataClasses.DataObj.Experiment), 22

ExperimentPhase (class in easyInterface.Diffraction.DataClasses.DataObj.Experiment), 23

ExperimentPhases (class in easyInterface.Diffraction.DataClasses.DataObj.Experiment), 23

Experiments (class in easyInterface.Diffraction.DataClasses.DataObj.Experiment), 23

## G

getCalculation () (easyInterface.Diffraction.Interface.CalculatorInterface method), 11

getCalculations () (easyInterface.Diffraction.Interface.CalculatorInterface method), 11

getDictByPath () (easyInterface.Diffraction.Interface.CalculatorInterface method), 11

getExperiment () (easyInterface.Diffraction.Interface.CalculatorInterface method), 11

getExperiments () (easyInterface.Diffraction.Calculators.CrspyCalculator method), 17

method), 17

getPhase() (easyInterface.Diffraction.Interface.CalculatorInterface method), 12

getPhases() (easyInterface.Diffraction.Calculators.CrspyCalculator method), 17

## I

isPolarised() (easyInterface.Diffraction.DataClasses.DataObj.Experiment.MeasuredPattern property), 24

## L

Limits (class in easyInterface.Diffraction.DataClasses.DataObj.Calculation), 22

## M

MeasuredPattern (class in easyInterface.Diffraction.DataClasses.DataObj.Experiment), 23

MSP (class in easyInterface.Diffraction.DataClasses.PhaseObj.Atom), 20

## N

name() (easyInterface.Diffraction.Interface.CalculatorInterface method), 12

## P

Phase (class in easyInterface.Diffraction.DataClasses.PhaseObj.Phase), 21

Phases (class in easyInterface.Diffraction.DataClasses.PhaseObj.Phase), 21

phasesCount() (easyInterface.Diffraction.Interface.CalculatorInterface method), 12

phasesIds() (easyInterface.Diffraction.Interface.CalculatorInterface method), 12

ProjectDict (class in easyInterface.Diffraction.Interface), 16

## R

readPhaseDefinition() (easyInterface.Diffraction.Calculators.CrspyCalculator method), 17

redo() (easyInterface.Diffraction.Interface.CalculatorInterface method), 12

refine() (easyInterface.Diffraction.Calculators.CrspyCalculator method), 17

refine() (easyInterface.Diffraction.Interface.CalculatorInterface method), 12

removeExperiment() (easyInterface.Diffraction.Interface.CalculatorInterface method), 12

removeExpsDefinition() (easyInterface.Diffraction.Calculators.CrspyCalculator method), 17

removePhase() (easyInterface.Diffraction.Interface.CalculatorInterface method), 12

removePhaseDefinition() (easyInterface.Diffraction.Calculators.CrspyCalculator method), 17

removePhaseFromExp() (easyInterface.Diffraction.Interface.CalculatorInterface method), 13

renamePhase() (easyInterface.Diffraction.DataClasses.PhaseObj.Phase.Phases method), 21

Resolution (class in easyInterface.Diffraction.DataClasses.DataObj.Experiment), 24

saveCifs() (easyInterface.Diffraction.Calculators.CrspyCalculator method), 17

saveCifs() (easyInterface.Diffraction.Interface.CalculatorInterface method), 13

setCalculatorFromProject() (easyInterface.Diffraction.Interface.CalculatorInterface method), 13

setDictByPath() (easyInterface.Diffraction.Interface.CalculatorInterface method), 13

setExperiment() (easyInterface.Diffraction.Interface.CalculatorInterface method), 13

setExperimentDefinition() (easyInterface.Diffraction.Interface.CalculatorInterface method), 13

setExperimentDefinitionFromString() (easyInterface.Diffraction.Interface.CalculatorInterface method), 13

setExperimentRefine() (easyInterface.Diffraction.Interface.CalculatorInterface method), 14

setExperiments() (easyInterface

*face.Diffraction.Calculators.CryspyCalculator*  
method), 18

setExperiments() (*easyInter-*  
*face.Diffraction.Interface.CalculatorInterface*  
method), 14

setExperimentValue() (*easyInter-*  
*face.Diffraction.Interface.CalculatorInterface*  
method), 14

setExpsDefinition() (*easyInter-*  
*face.Diffraction.Calculators.CryspyCalculator*  
method), 18

setObjFromProjectDicts() (*easyInter-*  
*face.Diffraction.Calculators.CryspyCalculator*  
method), 18

setPhase() (*easyInter-*  
*face.Diffraction.Interface.CalculatorInterface*  
method), 14

setPhaseDefinition() (*easyInter-*  
*face.Diffraction.Calculators.CryspyCalculator*  
method), 18

setPhaseDefinition() (*easyInter-*  
*face.Diffraction.Interface.CalculatorInterface*  
method), 14

setPhaseRefine() (*easyInter-*  
*face.Diffraction.Interface.CalculatorInterface*  
method), 14

setPhases() (*easyInter-*  
*face.Diffraction.Calculators.CryspyCalculator*  
method), 18

setPhases() (*easyInter-*  
*face.Diffraction.Interface.CalculatorInterface*  
method), 15

setPhaseValue() (*easyInter-*  
*face.Diffraction.Interface.CalculatorInterface*  
method), 15

setProjectFromCalculator() (*easyInter-*  
*face.Diffraction.Interface.CalculatorInterface*  
method), 15

SpaceGroup (class in *easyInter-*  
*face.Diffraction.DataClasses.PhaseObj.SpaceGroup*),  
21

## U

undo() (*easyInterface.Diffraction.Interface.CalculatorInterface*  
method), 15

updateCalculations() (*easyInter-*  
*face.Diffraction.Interface.CalculatorInterface*  
method), 15

updateExperiments() (*easyInter-*  
*face.Diffraction.Interface.CalculatorInterface*  
method), 15

updatePhases() (*easyInter-*  
*face.Diffraction.Interface.CalculatorInterface*  
method), 15

## W

writeExpCif() (*easyInter-*  
*face.Diffraction.Calculators.CryspyCalculator*  
method), 18

writeExpCif() (*easyInter-*  
*face.Diffraction.Interface.CalculatorInterface*  
method), 15

writeMainCif() (*easyInter-*  
*face.Diffraction.Calculators.CryspyCalculator*  
method), 18

writeMainCif() (*easyInter-*  
*face.Diffraction.Interface.CalculatorInterface*  
method), 16

writePhaseCif() (*easyInter-*  
*face.Diffraction.Calculators.CryspyCalculator*  
method), 19

writePhaseCif() (*easyInter-*  
*face.Diffraction.Interface.CalculatorInterface*  
method), 16

## Y

y\_obs\_lower() (*easyInter-*  
*face.Diffraction.DataClasses.DataObj.Experiment.MeasuredPatt*  
property), 24

y\_obs\_upper() (*easyInter-*  
*face.Diffraction.DataClasses.DataObj.Experiment.MeasuredPatt*  
property), 24