# easyInterface

***Release 0.0.5***

**Simon Ward**

**Feb 19, 2020**

# EXAMPLE GALLERIES


`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

CHAPTER

ONE

# FEATURES OF EASYINTERFACE

Boom!

CHAPTER

TWO

# PROJECTS USING EASYINTERFACE

easyInterface is currently being used in the following projects:

- easyDiffraction - A scientific software for modelling and analysis of the neutron diffraction data

CHAPTER

THREE

# 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

### 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.Utils.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

### Creating a QT interface

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

```
import os

from easyInterface.Utils.Helpers import getExamplesDir
from easyInterface.Diffraction.Calculators import CryspyCalculator
from easyInterface.Diffraction.QtInterface import QtCalculatorInterface

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)

**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 a interface to a cryspy calculator and then fit a value.

```
import os
from easyInterface.Utils.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, 'PbSO4_powder-1d_neutrons-unpol_D1A(ILL)', 'main.
↪cif')

calculator = CryspyCalculator(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.

**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*)

**addPhase**(*phase*)
Add a new phases from a cif file to the list of existing crystal phases.

**Parameters** **phase** (*Phase*) –

**addPhaseDefinition**(*phase_path*)
Add a new phases from a cif file to the list of existing crystal phases.

**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)
```

**asCifDict**()
...

**Return type** dict

**asDict**()
Return data dict.

**Return type** dict

**experimentsCount**()
Returns number of experiments in the project.

**Return type** int

**experimentsIds**()
Returns labels of the experiments in the project.

**Return type** list

**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

**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

**phasesCount**()
Returns number of phases in the project.

**Return type** int

**phasesIds**()
Returns labels of the phases in the project.

**Return type** list

**refine**()
refinement ...

**Return type** dict

**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.

**Returns** None

**setExperiment**(*experiment*)
Set phases (sample model tab in GUI)

**setExperimentDefinition**(*exp_path*)
Parse the relevant phases file and update the corresponding model

**setExperiments**(*experiments=None*)
Set experiments (Experimental data tab in GUI)

**setPhase**(*phase*)
Set phases (sample model tab in GUI)

**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)
```

**setPhases**(*phases=None*)
Set phases (sample model tab in GUI)

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

**updatePhases**()
Synchronise the phases in project dictionary by queering the calculator object.

**Returns** None

**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.

**Returns** None

**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.

**Returns** None

**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.

**Returns** None

### 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**

- **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 Cryspy Calculator

**class** easyInterface.Diffraction.Calculators.**CryspyCalculator**(*main_rcif_path=None*)

**asCifDict**()
...

**Return type** dict

**getPhases**()
Set phases (sample model tab in GUI)

**Return type** *Phases*

**refine**()
refinement ...

**setExperiments**(*experiments*)
Set experiments (Experimental data tab in GUI)

**setObjFromProjectDicts**(*phases*, *experiments*)
Set all the cryspy parameters from project dictionary

**setPhases**(*phases*)
Set phases (sample model tab in GUI)

### 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*, *occupancy*, *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 crysolagraphic unit cell parameters

**classmethod default**()
Default constructor for a crystolographic unit cell

**Return type** *Cell*

**Returns** Default crystolographic unit cell container

**classmethod fromPars**(*length_a*, *length_b*, *length_c*, *angle_alpha*, *angle_beta*, *angle_gamma*)
Constructor of a crystolographic 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*, *spacegroup*, *cell*, *atoms*, *sites*)
Container for crysolographic 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_H space_group_IT_numb ori-gin_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*, *cal-cu-lated_pattern*, *lim-its*)
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*, *intensity*)

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*, *wavelength*, *offset*, *phase*, *background*, *resolution*, *measured_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_pha*
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=No*, *sy_obs_down=N*
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

CHAPTER

FOUR

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## e

# INDEX

## R



## S



## U



## W



## Y