pyglotaran Documentation

Release v0.5.0

Joern Weissenborn, Joris Snellenburg, Ivo van Stokkum

2021-12-02

CONTENTS:

CHAPTER

ONE

INTRODUCTION

Pyglotaran is a python library for global analysis of time-resolved spectroscopy data. It is designed to provide a state of the art modeling toolbox to researchers, in a user-friendly manner.

Its features are:

- user-friendly modeling with a custom YAML (`*.yml`) based modeling language
- parameter optimization using variable projection and non-negative least-squares algorithms
- easy to extend modeling framework
- battle-hardened model and algorithms for fluorescence dynamics
- build upon and fully integrated in the standard Python science stack (NumPy, SciPy, Jupyter)

1.1 A Note To Glotaran Users

Although closely related and developed in the same lab, pyglotaran is not a replacement for Glotaran - A GUI For TIMP. Pyglotaran only aims to provide the modeling and optimization framework and algorithms. It is of course possible to develop a new GUI which leverages the power of pyglotaran (contributions welcome).

The current 'user-interface' for pyglotaran is Jupyter Notebook. It is designed to seamlessly integrate in this environment and be compatible with all major visualization and data analysis tools in the scientific python environment.

If you are a non-technical user, you should give these tools a try, there are numerous tutorials how to use them. You don't need to really learn to program. If you can use e.g. Matlab or Mathematica, you can use Jupyter and Python.

CHAPTER

TWO

INSTALLATION

2.1 Prerequisites

- Python 3.6 or later

2.1.1 Windows

The easiest way of getting Python (and some basic tools to work with it) in Windows is to use Anaconda, which provides python.

You will need a terminal for the installation. One is provided by *Anaconda* and is called *Anaconda Console*. You can find it in the start menu.

Note: If you use a Windows Shell like cmd.exe or PowerShell, you might have to prefix '$PATH_TO_ANACONDA/' to all commands (e.g. *C:/Anaconda/pip.exe* instead of *pip*)

2.2 Stable release

Warning: pyglotaran is early development, so for the moment stable releases are sparse and outdated. We try to keep the master code stable, so please install from source for now.

This is the preferred method to install pyglotaran, as it will always install the most recent stable release.

To install pyglotaran, run this command in your terminal:

```
$ pip install pyglotaran
```

If you don't have pip installed, this Python installation guide can guide you through the process.

If you want to install it via conda, you can run the following command:

```
$ conda install -c conda-forge pyglotaran
```

2.3 From sources

First you have to install or update some dependencies.

Within a terminal:

```
$ pip install -U numpy scipy Cython
```

Alternatively, for Anaconda users:

```
$ conda install numpy scipy Cython
```

Afterwards you can simply use pip to install it directly from Github.

```
$ pip install git+https://github.com/glotaran/pyglotaran.git
```

For updating pyglotaran, just re-run the command above.

If you prefer to manually download the source files, you can find them on Github. Alternatively you can clone them with git (preferred):

```
$ git clone https://github.com/glotaran/pyglotaran.git
```

Within a terminal, navigate to directory where you have unpacked or cloned the code and enter

```
$ pip install -e .
```

For updating, simply download and unpack the newest version (or run `$ git pull` in pyglotaran directory if you used git) and and re-run the command above.

The following section was generated from `docs/source/notebooks/quickstart/quickstart.ipynb`

CHAPTER

THREE

QUICKSTART/CHEAT-SHEET

Since this documentation is written in a jupyter-notebook we will import a little ipython helper function to display file with syntax highlighting.

[1]:
```
from glotaran.utils.ipython import display_file
```

To start using `pyglotaran` in your project, you have to import it first. In addition we need to import some extra components for later use.

[2]:
```
from glotaran.analysis.optimize import optimize
from glotaran.io import load_model
from glotaran.io import load_parameters
from glotaran.io import save_dataset
from glotaran.io.prepare_dataset import prepare_time_trace_dataset
from glotaran.project.scheme import Scheme
```

Let us get some example data to analyze:

[3]:
```
from glotaran.examples.sequential import dataset

dataset
```

[3]:
```
<xarray.Dataset>
Dimensions:   (time: 2100, spectral: 72)
Coordinates:
  * time      (time) float64 -1.0 -0.99 -0.98 -0.97 ... 19.96 19.97 19.98 19.99
  * spectral  (spectral) float64 600.0 601.4 602.8 604.2 ... 696.6 698.0 699.4
Data variables:
    data      (time, spectral) float64 -0.000586 0.00234 ... 1.719 1.528
Attributes:
    source_path:  dataset_1.nc
```

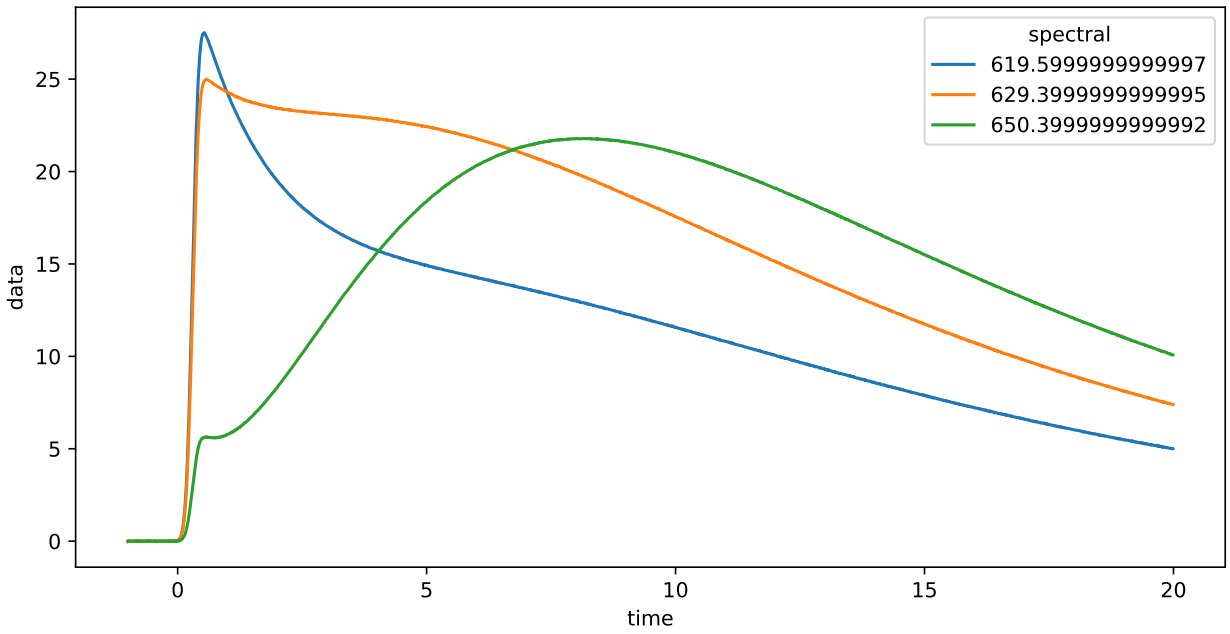
Like all data in `pyglotaran`, the dataset is a xarray.Dataset. You can find more information about the `xarray` library the xarray hompage.

The loaded dataset is a simulated sequential model.

3.1 Plotting raw data

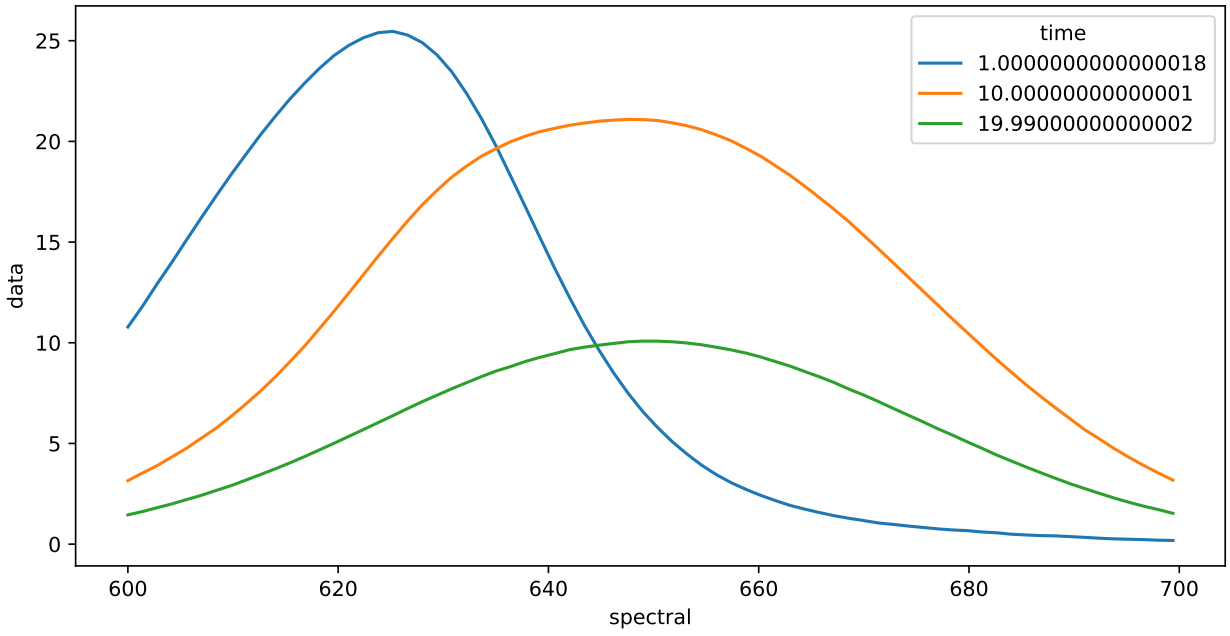
Now we lets plot some time traces.

[4]:
```
plot_data = dataset.data.sel(spectral=[620, 630, 650], method="nearest")
plot_data.plot.line(x="time", aspect=2, size=5);
```



We can also plot spectra at different times.

[5]:
```
plot_data = dataset.data.sel(time=[1, 10, 20], method="nearest")
plot_data.plot.line(x="spectral", aspect=2, size=5);
```



3.2 Preparing data

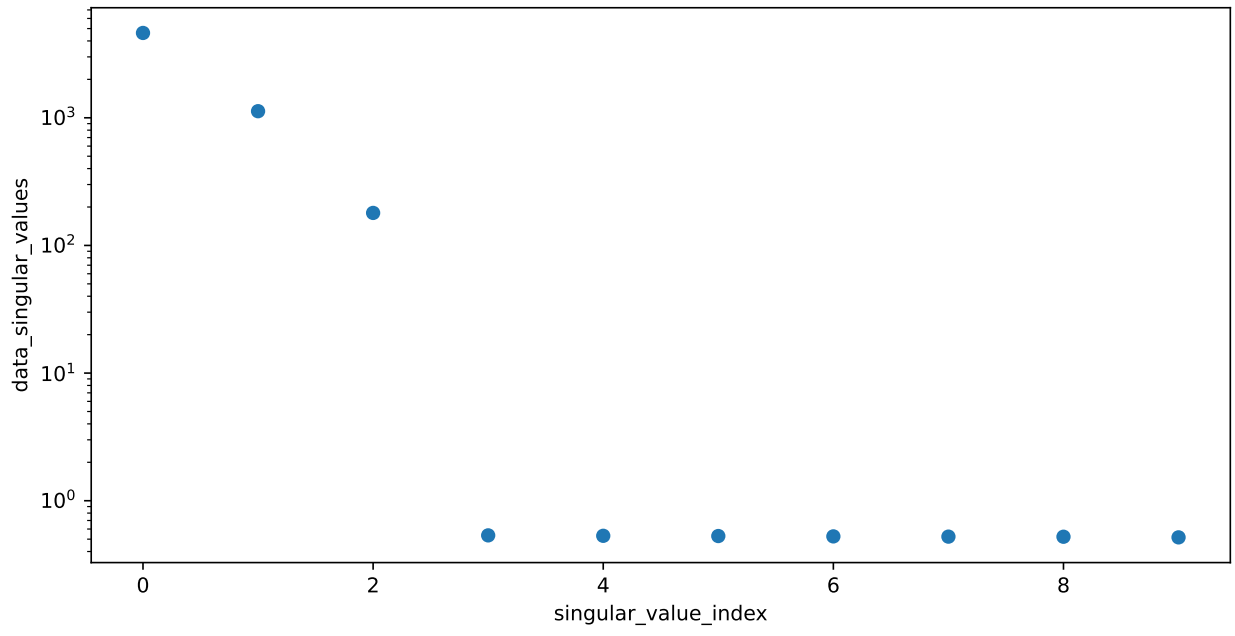
To get an idea about how to model your data, you should inspect the singular value decomposition. Pyglotaran has a function to calculate it (among other things).

[6]:
```
dataset = prepare_time_trace_dataset(dataset)
dataset
```

[6]:
```
<xarray.Dataset>
Dimensions:                         (time: 2100, spectral: 72, left_singular_value_index:␣
↪72, singular_value_index: 72, right_singular_value_index: 72)
Coordinates:
  * time                            (time) float64 -1.0 -0.99 -0.98 ... 19.98 19.99
  * spectral                        (spectral) float64 600.0 601.4 ... 698.0 699.4
Dimensions without coordinates: left_singular_value_index, singular_value_index, right_
↪singular_value_index
Data variables:
    data                            (time, spectral) float64 -0.000586 ... 1.528
    data_left_singular_vectors      (time, left_singular_value_index) float64 2...
    data_singular_values            (singular_value_index) float64 4.62e+03 ... ...
    data_right_singular_vectors     (right_singular_value_index, spectral) float64 ...
Attributes:
    source_path:  dataset_1.nc
```

First, take a look at the first 10 singular values:

[7]:
```
plot_data = dataset.data_singular_values.sel(singular_value_index=range(0, 10))
plot_data.plot(yscale="log", marker="o", linewidth=0, aspect=2, size=5);
```



3.3 Working with models

To analyze our data, we need to create a model.

Create a file called `model.yaml` in your working directory and fill it with the following:

[8]:
```
display_file("model.yaml", syntax="yaml")
```

[8]:
```yaml
default_megacomplex: decay

initial_concentration:
  input:
    compartments: [s1, s2, s3]
    parameters: [input.1, input.0, input.0]

k_matrix:
  k1:
    matrix:
      (s2, s1): kinetic.1
      (s3, s2): kinetic.2
      (s3, s3): kinetic.3

megacomplex:
  m1:
    k_matrix: [k1]

irf:
  irf1:
    type: gaussian
    center: irf.center
    width: irf.width

dataset:
  dataset1:
    initial_concentration: input
    megacomplex: [m1]
    irf: irf1
```

Now you can load the model file.

[9]:
```
model = load_model("model.yaml")
```

You can check your model for problems with `model.validate`.

[10]:
```
model.validate()
```

[10]:
```
'Your model is valid.'
```

3.4 Working with parameters

Now define some starting parameters. Create a file called `parameters.yaml` with the following content.

[11]:
```
display_file("parameters.yaml", syntax="yaml")
```

[11]:
```yaml
input:
 - ['1', 1, {'vary': False, 'non-negative': False}]
 - ['0', 0, {'vary': False, 'non-negative': False}]

kinetic: [
    0.5,
    0.3,
    0.1,
]

irf:
 - ['center', 0.3]
 - ['width', 0.1]
```

[12]:
```
parameters = load_parameters("parameters.yaml")
```

You can `model.validate` also to check for missing parameters.

[13]:
```
model.validate(parameters=parameters)
```

[13]:
```
'Your model is valid.'
```

Since not all problems in the model can be detected automatically it is wise to visually inspect the model. For this purpose, you can just print the model.

[14]:
```
model
```

[14]:

3.4.1 Model

Megacomplex Types: decay

Dataset Groups

- **default**:
- *Label*: default
- *residual_function*: variable_projection
- *link_clp*: None

K Matrix

- **k1**:
  - *Label*: k1
  - *Matrix*:

(continues on next page)

(continued from previous page)

  - ('s2', 's1'): kinetic.1(nan)
  - ('s3', 's2'): kinetic.2(nan)
  - ('s3', 's3'): kinetic.3(nan)

Initial Concentration

- **input**:
  - *Label*: input
  - *Compartments*:
  - s1
  - s2
  - s3
  - *Parameters*:
  - input.1(nan)
  - input.0(nan)
  - input.0(nan)
  - *Exclude From Normalize*:

Irf

- **irf1** (gaussian):
  - *Label*: irf1
  - *Type*: gaussian
  - *Center*: irf.center(nan)
  - *Width*: irf.width(nan)
  - *Normalize*: True
  - *Backsweep*: False

Megacomplex

- **m1** (None):
  - *Label*: m1
  - *Dimension*: time
  - *K Matrix*:
  - k1

(continues on next page)

(continued from previous page)

Dataset

- **dataset1**:
  - *Label*: dataset1
  - *Group*: default
  - *Megacomplex*:
  - m1
  - *Initial Concentration*: input
  - *Irf*: irf1

The same way you should inspect your parameters.

[15]:
```
parameters
```

[15]:

- **input**:

Label	*Value*	*Standard Error*	*Minimum*	*Maximum*	*Vary*	*Non-Negative*	*Expression*
1	1.000e+00	nan	-inf	inf	False	False	None
0	0.000e+00	nan	-inf	inf	False	False	None

- **irf**:

Label	*Value*	*Standard Error*	*Minimum*	*Maximum*	*Vary*	*Non-Negative*	*Expression*
center	3.000e-01	nan	-inf	inf	True	False	None
width	1.000e-01	nan	-inf	inf	True	False	None

- **kinetic**:

Label	*Value*	*Standard Error*	*Minimum*	*Maximum*	*Vary*	*Non-Negative*	*Expression*
1	5.000e-01	nan	-inf	inf	True	False	None
2	3.000e-01	nan	-inf	inf	True	False	None
3	1.000e-01	nan	-inf	inf	True	False	None

3.5 Optimizing data

Now we have everything together to optimize our parameters. First we import optimize.

[16]:
```
scheme = Scheme(model, parameters, {"dataset1": dataset})
result = optimize(scheme)
result
```

```
   Iteration     Total nfev        Cost      Cost reduction    Step norm     Optimality
       0              1         7.5629e+00                                    6.68e+01
       1              2         7.5624e+00      4.68e-04       8.07e-05       2.26e-01
       2              3         7.5624e+00      4.33e-10       2.33e-08       6.04e-06
`ftol` termination condition is satisfied.
Function evaluations 3, initial cost 7.5629e+00, final cost 7.5624e+00, first-order␣
↪optimality 6.04e-06.
```

[16]:

Optimization Result	
Number of residual evaluation	3
Number of variables	5
Number of datapoints	151200
Degrees of freedom	151195
Chi Square	1.51e+01
Reduced Chi Square	1.00e-04
Root Mean Square Error (RMSE)	1.00e-02
RMSE additional penalty	[array([], dtype=float64)]

3.5.1 Model

Megacomplex Types: decay

Dataset Groups

- **default**:
- *Label*: default
- *residual_function*: variable_projection
- *link_clp*: None

K Matrix

- **k1**:
  - *Label*: k1
  - *Matrix*:
  - ('s2', 's1'): kinetic.1(5.00e-01±7.26e-05, initial: 5.00e-01)
  - ('s3', 's2'): kinetic.2(3.00e-01±4.19e-05, initial: 3.00e-01)
  - ('s3', 's3'): kinetic.3(1.00e-01±4.78e-06, initial: 1.00e-01)

(continues on next page)

(continued from previous page)

Initial Concentration

- **input**:
  - *Label*: input
  - *Compartments*:
  - s1
  - s2
  - s3
  - *Parameters*:
  - input.1(1.00e+00, fixed)
  - input.0(0.00e+00, fixed)
  - input.0(0.00e+00, fixed)
  - *Exclude From Normalize*:

Irf

- **irf1** (gaussian):
  - *Label*: irf1
  - *Type*: gaussian
  - *Center*: irf.center(3.00e-01±5.01e-06, initial: 3.00e-01)
  - *Width*: irf.width(1.00e-01±6.71e-06, initial: 1.00e-01)
  - *Normalize*: True
  - *Backsweep*: False

Megacomplex

- **m1** (None):
  - *Label*: m1
  - *Dimension*: time
  - *K Matrix*:
  - k1

Dataset

- **dataset1**:
  - *Label*: dataset1

(continues on next page)

(continued from previous page)

- *Group*: default
- *Megacomplex*:
- m1
- *Initial Concentration*: input
- *Irf*: irf1

[17]:
```
result.optimized_parameters
```

[17]:

- **input**:

Label	*Value*	*Standard Error*	*Minimum*	*Maximum*	*Vary*	*Non-Negative*	*Expression*
1	1.000e+00	nan	-inf	inf	False	False	None
0	0.000e+00	nan	-inf	inf	False	False	None

- **irf**:

Label	*Value*	*Standard Error*	*Minimum*	*Maximum*	*Vary*	*Non-Negative*	*Expression*
center	3.000e-01	5.012e-06	-inf	inf	True	False	None
width	1.000e-01	6.705e-06	-inf	inf	True	False	None

- **kinetic**:

Label	*Value*	*Standard Error*	*Minimum*	*Maximum*	*Vary*	*Non-Negative*	*Expression*
1	5.000e-01	7.256e-05	-inf	inf	True	False	None
2	2.999e-01	4.191e-05	-inf	inf	True	False	None
3	1.000e-01	4.783e-06	-inf	inf	True	False	None

You can get the resulting data for your dataset with `result.get_dataset`.

[18]:
```
result_dataset = result.data["dataset1"]
result_dataset
```

[18]:
```
<xarray.Dataset>
Dimensions:                                    (clp_label: 3, time: 2100, spectral: 72,␣
↪left_singular_value_index: 72, singular_value_index: 72, right_singular_value_index:␣
↪72, species: 3, component: 3, to_species: 3, from_species: 3)
Coordinates:
  * clp_label                                  (clp_label) object 's1' 's2' 's3'
  * time                                       (time) float64 -1.0 ... 19.99
  * spectral                                   (spectral) float64 600.0 ... 699.4
  * species                                    (species) <U2 's1' 's2' 's3'
```

(continues on next page)

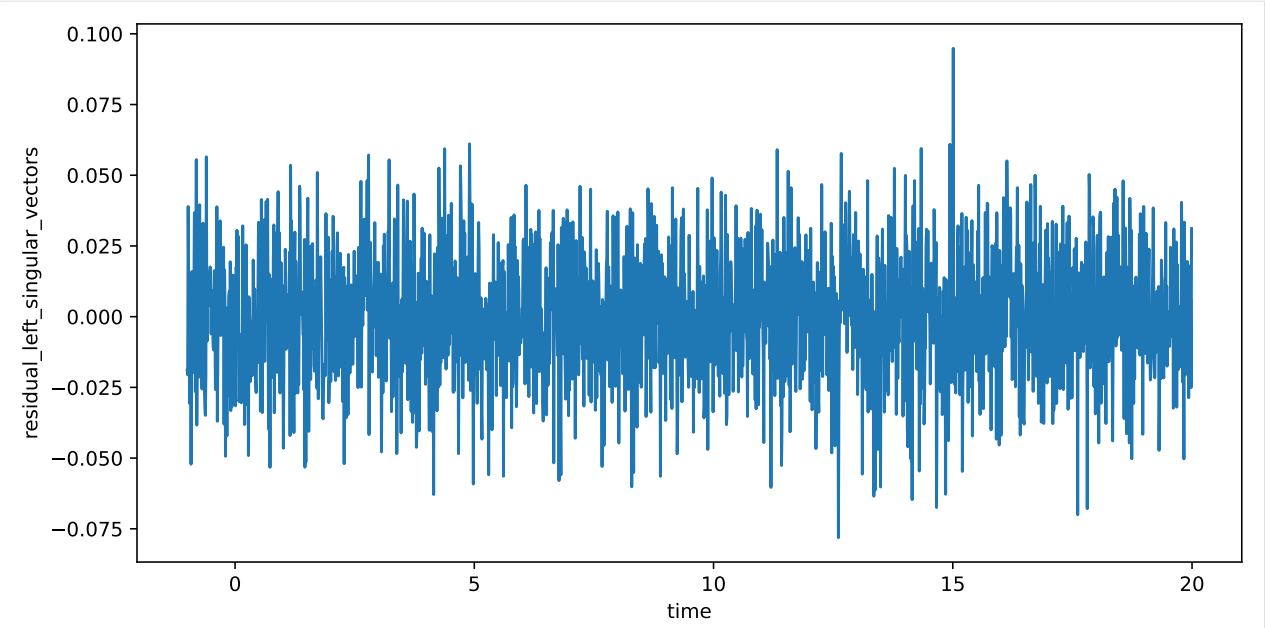
(continued from previous page)

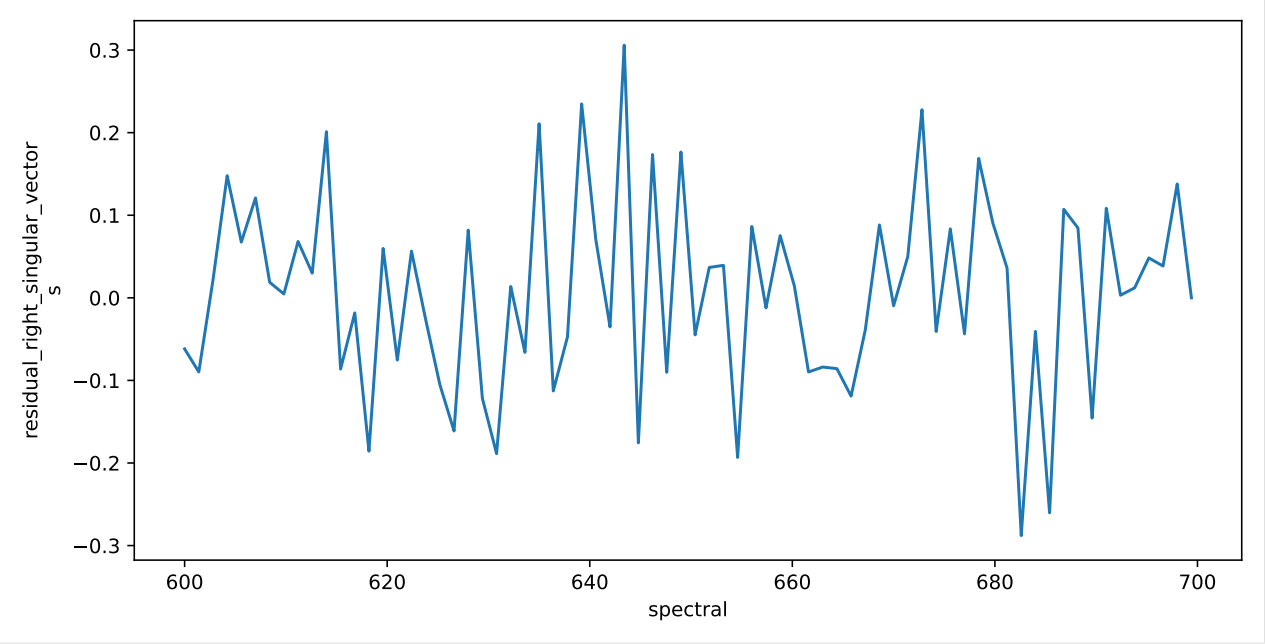
```
  * component                                (component) int64 0 1 2
    rate                                     (component) float64 -0.5 ... -0.1
    lifetime                                 (component) float64 -2.0 ... -9...
  * to_species                               (to_species) <U2 's1' 's2' 's3'
  * from_species                             (from_species) <U2 's1' 's2' 's3'
Dimensions without coordinates: left_singular_value_index, singular_value_index, right_
↪singular_value_index
Data variables: (12/24)
    data                                     (time, spectral) float64 -0.000...
    data_left_singular_vectors               (time, left_singular_value_index) float64␣
↪...
    data_singular_values                     (singular_value_index) float64 ...
    data_right_singular_vectors              (spectral, right_singular_value_index)␣
↪float64 ...
    matrix                                   (time, clp_label) float64 6.101...
    clp                                      (spectral, clp_label) float64 1...
    ...                                       ...
    irf_center                               float64 0.3
    irf_width                                float64 0.1
    decay_associated_spectra                 (spectral, component) float64 2...
    a_matrix                                 (component, species) float64 1...
    k_matrix                                 (to_species, from_species) float64 ...
    k_matrix_reduced                         (to_species, from_species) float64 ...
Attributes:
    source_path:                      dataset_1.nc
    root_mean_square_error:           0.01000158369561628
    weighted_root_mean_square_error:  0.01000158369561628
    dataset_scale:                    1
```

3.6 Visualize the Result

The resulting data can be visualized the same way as the dataset. To judge the quality of the fit, you should look at first left and right singular vectors of the residual.

[19]:
```
residual_left = result_dataset.residual_left_singular_vectors.sel(left_singular_value_
↪index=0)
residual_right = result_dataset.residual_right_singular_vectors.sel(right_singular_value_
↪index=0)
residual_left.plot.line(x="time", aspect=2, size=5)
residual_right.plot.line(x="spectral", aspect=2, size=5);
```





Finally, you can save your result.

[20]:
```
save_dataset(result_dataset, "dataset1.nc")
```

CHAPTER

FOUR

CHANGELOG

4.1 0.5.0 (2021-12-01)

4.1.1 Features

- Feature: Megacomplex Models (#736)
- Feature: Full Models (#747)
- Damped Oscillation Megacomplex (a.k.a. DOAS) (#764)
- Add Dataset Groups (#851)
- Performance improvements (in some cases up to 5x) (#740)

4.1.2 Minor Improvements:

- Add dimensions to megacomplex and dataset_descriptor (#702)
- Improve ordering in k_matrix involved_compartments function (#788)
- Improvements to application of clp_penalties (equal area) (#801)
- Refactor model.from_dict to parse megacomplex_type from dict and add simple_generator for testing (#807)
- Refactor model spec (#836)
- Refactor Result Saving (#841)
- Use ruaml.yaml parser for roundtrip support (#893)
- Refactor Result and Scheme loading/initializing from files (#903)
- Several refactoring in `glotaran.Parameter` (#910)
- Improved Reporting of Parameters (#910, #914, #918)
- Scheme now excepts paths to model, parameter and data file without initializing them first (#912)

4.1.3 Bug fixes

- Fix/cli0.5 (#765)
- Fix compartment ordering randomization due to use of set (#799)
- Fix check_deprecations not showing deprecation warnings (#775)
- Fix and re-enable IRF Dispersion Test (#786)
- Fix coherent artifact crash for index dependent models #808
- False positive model validation fail when combining multiple default megacomplexes (#797)
- Fix ParameterGroup repr when created with 'from_list' (#827)
- Fix for DOAS with reversed oscillations (negative rates) (#839)
- Fix parameter expression parsing (#843)
- Use a context manager when opening a nc dataset (#848)
- Disallow xarray versions breaking plotting in integration tests (#900)
- Fix 'dataset_groups' not shown in model markdown (#906)

4.1.4 Documentation

- Moved API documentation from User to Developer Docs (#776)
- Add docs for the CLI (#784)
- Fix deprecation in model used in quickstart notebook (#834)

4.1.5 Deprecations (due in 0.7.0)

- `glotaran.model.Model.model_dimension` -> `glotaran.project.Scheme.model_dimension`
- `glotaran.model.Model.global_dimension` -> `glotaran.project.Scheme.global_dimension`
- `<model_file>.type.kinetic-spectrum` -> `<model_file>.default_megacomplex.decay`
- `<model_file>.type.spectral-model` -> `<model_file>.default_megacomplex.spectral`
- `<model_file>.spectral_relations` -> `<model_file>.clp_relations`
- `<model_file>.spectral_relations.compartment` -> `<model_file>.clp_relations.source`
- `<model_file>.spectral_constraints` -> `<model_file>.clp_constraints`
- `<model_file>.spectral_constraints.compartment` -> `<model_file>.clp_constraints.target`
- `<model_file>.equal_area_penalties` -> `<model_file>.clp_area_penalties`
- `<model_file>.irf.center_dispersion` -> `<model_file>.irf.center_dispersion_coefficients`
- `<model_file>.irf.width_dispersion` -> `<model_file>.irf.width_dispersion_coefficients`
- `glotaran.project.Scheme(..., non_negative_least_squares=...)` -> `<model_file>dataset_groups.default.residual_function`
- `glotaran.project.Scheme(..., group=...)` -> `<model_file>dataset_groups.default.link_clp`
- `glotaran.project.Scheme(..., group_tolerance=...)` -> `glotaran.project.Scheme(..., clp_link_tolerance=...)`

- `<scheme_file>.maximum-number-function-evaluations -> <scheme_file>.maximum_number_function_evaluations`
- `<model_file>.non-negative-least-squares: true -> <model_file>dataset_groups.default.residual_function: non_negative_least_squares`
- `<model_file>.non-negative-least-squares: false -> <model_file>dataset_groups.default.residual_function: variable_projection`
- `glotaran.parameter.ParameterGroup.to_csv(file_name=parameters.csv) -> glotaran.io.save_parameters(parameters, 'file_name=parameters.csv)`

4.1.6 Maintenance

- Fix Performance Regressions (between version) (#740)
- Add integration test result validation (#754)
- Add more QA tools for parts of glotaran (#739)
- Fix interrogate usage (#781)
- Speedup PR benchmark (#785)
- Use pinned versions of dependencies to run integration CI tests (#892)
- Move megacomplex integration tests from root level to megacomplexes (#894)
- Fix artifact download in pr_benchmark_reaction workflow (#907)

4.2 0.4.0 (2021-06-25)

4.2.1 Features

- Add basic spectral model (#672)
- Add Channel/Wavelength dependent shift parameter to irf. (#673)
- Refactored Problem class into GroupedProblem and UngroupedProblem (#681)
- Plugin system was rewritten (#600, #665)
- Deprecation framework (#631)
- Better notebook integration (#689)

4.2.2 Bug fixes

- Fix excessive memory usage in `_create_svd` (#576)
- Fix several issues with KineticImage model (#612)
- Fix exception in sdt reader index calculation (#647)
- Avoid crash in result markdown printing when optimization fails (#630)
- ParameterNotFoundException doesn't prepend '.' if path is empty (#688)
- Ensure Parameter.label is str or None (#678)
- Properly scale StdError of estimated parameters with RMSE (#704)

- More robust covariance_matrix calculation (#706)
- `ParameterGroup.markdown()` independent parametergroups of order (#592)

4.2.3 Plugins

- `ProjectIo` 'folder'/'legacy' plugin to save results (#620)
- `Model` 'spectral-model' (#672)

4.2.4 Documentation

- User documentation is written in notebooks (#568)
- Documentation on how to write a `DataIo` plugin (#600)

4.2.5 Deprecations (due in 0.6.0)

- `glotaran.ParameterGroup` -> `glotaran.parameterParameterGroup`
- `glotaran.read_model_from_yaml` -> `glotaran.io.load_model(..., format_name="yaml_str")`
- `glotaran.read_model_from_yaml_file` -> `glotaran.io.load_model(..., format_name="yaml")`
- `glotaran.read_parameters_from_csv_file` -> `glotaran.io.load_parameters(..., format_name="csv")`
- `glotaran.read_parameters_from_yaml` -> `glotaran.io.load_parameters(..., format_name="yaml_str")`
- `glotaran.read_parameters_from_yaml_file` -> `glotaran.io.load_parameters(..., format_name="yaml")`
- `glotaran.io.read_data_file` -> `glotaran.io.load_dataset`
- `result.save` -> `glotaran.io.save_result(result, ..., format_name="legacy")`
- `result.get_dataset("<dataset_name>")` -> `result.data["<dataset_name>"]`
- `glotaran.analysis.result` -> `glotaran.project.result`
- `glotaran.analysis.scheme` -> `glotaran.project.scheme`
- `model.simulate` -> `glotaran.analysis.simulation.simulate(model, ...)`

4.3 0.3.3 (2021-03-18)

- Force recalculation of SVD attributes in `scheme._prepare_data` (#597)
- Remove unneeded check in `spectral_penalties._get_area` Fixes (#598)
- Added python 3.9 support (#450)

4.4 0.3.2 (2021-02-28)

- Re-release of version 0.3.1 due to packaging issue

4.5 0.3.1 (2021-02-28)

- Added compatibility for numpy 1.20 and raised minimum required numpy version to 1.20 (#555)
- Fixed excessive memory consumption in result creation due to full SVD computation (#574)
- Added feature parameter history (#557)
- Moved setup logic to `setup.cfg` (#560)

4.6 0.3.0 (2021-02-11)

- Significant code refactor with small API changes to parameter relation specification (see docs)
- Replaced lmfit with scipy.optimize

4.7 0.2.0 (2020-12-02)

- Large refactor with significant improvements but also small API changes (see docs)
- Removed doas plugin

4.8 0.1.0 (2020-07-14)

- Package was renamed to `pyglotaran` on PyPi

4.9 0.0.8 (2018-08-07)

- Changed `nan_policiy` to `omit`

4.10 0.0.7 (2018-08-07)

- Added support for multiple shapes per compartment.

4.11 0.0.6 (2018-08-07)

- First release on PyPI, support for Windows installs added.
- Pre-Alpha Development

CHAPTER

FIVE

AUTHORS

5.1 Development Lead

- Joern Weissenborn <joern.weissenborn@gmail.com>
- Joris Snellenburg <j.snellenburg@gmail.com>

5.2 Contributors

- Sebastian Weigand <s.weigand.phy@gmail.com>

5.3 Special Thanks

- Stefan Schuetz
- Sergey P. Laptenok

5.4 Supervision

- **dr. Ivo H.M. van Stokkum** <i.h.m.van.stokkum@vu.nl> (University profile)

5.5 Original publications

1. Joris J. Snellenburg, Sergey Laptenok, Ralf Seger, Katharine M. Mullen, Ivo H. M. van Stokkum. "Glotaran: A Java-Based Graphical User Interface for the R Package TIMP". Journal of Statistical Software (2012), Volume 49, Number 3, Pages: 1–22. URL https://dx.doi.org/10.18637/jss.v049.i03
2. Katharine M. Mullen, Ivo H. M. van Stokkum. "TIMP: An R Package for Modeling Multi-way Spectroscopic Measurements". Journal of Statistical Software (2007), Volume 18, Number 3, Pages 1-46, ISSN 1548-7660. URL https://dx.doi.org/10.18637/jss.v018.i03
3. Ivo H. M. van Stokkum, Delmar S. Larsen, Rienk van Grondelle, "Global and target analysis of time-resolved spectra". Biochimica et Biophysica Acta (BBA) - Bioenergetics (2004), Volume 1657, Issues 2–3, Pages 82-104, ISSN 0005-2728. URL https://doi.org/10.1016/j.bbabio.2004.04.011

CHAPTER

SIX

OVERVIEW

CHAPTER

SEVEN

DATA IO

CHAPTER

EIGHT

PLOTTING

CHAPTER

NINE

MODELLING

CHAPTER

TEN

PARAMETER

CHAPTER

ELEVEN

OPTIMIZING

CHAPTER

TWELVE

PLUGINS

To be as flexible as possible `pyglotaran` uses a plugin system to handle new `Models`, `DataIo` and `ProjectIo`. Those plugins can be defined by `pyglotaran` itself, the user or a 3rd party plugin package.

12.1 Builtin plugins

12.1.1 Models

- `KineticSpectrumModel`
- `KineticImageModel`

12.1.2 Data Io

Plugins reading and writing data to and from xarray.Dataset or xarray.DataArray.

- `AsciiDataIo`
- `NetCDFDataIo`
- `SdtDataIo`

12.1.3 Project Io

Plugins reading and writing, `Model`,:class:*Schema*,:class:*ParameterGroup* or `Result`.

- `YmlProjectIo`
- `CsvProjectIo`
- `FolderProjectIo`

12.2 Reproducibility and plugins

With a plugin ecosystem there always is the possibility that multiple plugins try register under the same format/name. This is why plugins are registered at least twice. Once under the name the developer intended and secondly under their full name (full import path). This allows to ensure that a specific plugin is used by manually specifying the plugin, so if someone wants to run your analysis the results will be reproducible even if they have conflicting plugins installed. You can gain all information about the installed plugins by calling the corresponding `*_plugin_table` function with both options (`plugin_names` and `full_names`) set to true. To pin a used plugin use the corresponding `set_*_plugin` function with the intended name (`format_name`/`model_name`) and the full name (`full_plugin_name`) of the plugin to use.

If you wanted to ensure that the pyglotaran builtin plugin is used for `sdt` files you could add the following lines to the beginning of your analysis code.

```
from glotaran.io import set_data_plugin
set_data_plugin("sdt", "glotaran.builtin.io.sdt.sdt_file_reader.SdtDataIo_sdt")
```

12.2.1 Models

The functions for model plugins are located in `glotaran.model` and called `model_plugin_table` and `set_model_plugin`.

12.2.2 Data Io

The functions for data io plugins are located in `glotaran.io` and called `data_io_plugin_table` and `set_data_plugin`.

12.2.3 Project Io

The functions for project io plugins are located in `glotaran.io` and called `project_io_plugin_table` and `set_project_plugin`.

12.3 3rd party plugins

Plugins not part of `pyglotaran` itself.

- Not yet, why not be the first? Tell us about your plugin and we will feature it here.

CHAPTER

THIRTEEN

COMMAND-LINE INTERFACE

13.1 glotaran

The glotaran CLI main function.

```
glotaran [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

13.1.1 optimize

Optimizes a model. e.g.: glotaran optimize –

```
glotaran optimize [OPTIONS] [SCHEME_FILE]
```

Options

-dfmt, --dataformat <dataformat>
The input format of the data. Will be inferred from extension if not set.

Options ascii | nc | sdt

-d, --data <data>
Path to a dataset in the form ‘–data DATASET_LABEL PATH_TO_DATA’

-o, --out <out>
Path to an output directory.

-ofmt, --outformat <outformat>
The format of the output.

Default folder

Options folder | legacy | yaml

-n, --nfev <nfev>
Maximum number of function evaluations.

--nnls
Use non-negative least squares.

-y, --yes
Don't ask for confirmation.

-p, --parameters_file <parameters_file>
(optional) Path to parameter file.

-m, --model_file <model_file>
Path to model file.

Arguments

SCHEME_FILE
Optional argument

13.1.2 pluginlist

Prints a list of installed plugins.

```
glotaran pluginlist [OPTIONS]
```

13.1.3 print

Parses scheme, a model or a parameter file and prints the result as a Markdown formatted string.

```
glotaran print [OPTIONS] [SCHEME_FILE]
```

Options

-p, --parameters_file <parameters_file>
(optional) Path to parameter file.

-m, --model_file <model_file>
Path to model file.

Arguments

SCHEME_FILE
Optional argument

13.1.4 validate

Validates a model file and optionally a parameter file.

```
glotaran validate [OPTIONS] [SCHEME_FILE]
```

Options

-p, --parameters_file <parameters_file>
(optional) Path to parameter file.

-m, --model_file <model_file>
Path to model file.

Arguments

SCHEME_FILE
Optional argument

CHAPTER

FOURTEEN

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

14.1 Types of Contributions

14.1.1 Report Bugs

Report bugs at https://github.com/glotaran/pyglotaran/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

14.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

14.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

14.1.4 Write Documentation

pyglotaran could always use more documentation, whether as part of the official pyglotaran docs, in docstrings, or even on the web in blog posts, articles, and such. If you are writing docstrings please use the NumPyDoc style to write them.

14.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/glotaran/pyglotaran/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

14.2 Get Started!

Ready to contribute? Here's how to set up `pyglotaran` for local development.

1. Fork the `pyglotaran` repo on GitHub.
2. Clone your fork locally:

```
$ git clone https://github.com/<your_name_here>/pyglotaran.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pyglotaran
(pyglotaran)$ cd pyglotaran
(pyglotaran)$ python -m pip install -r requirements_dev.txt
(pyglotaran)$ pip install -e . --process-dependency-links
```

4. Install the `pre-commit` hooks, to automatically format and check your code:

```
$ pre-commit install
```

5. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

6. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ pre-commit run -a
$ py.test
```

   Or to run all at once:

```
$ tox
```

7. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request through the GitHub website.

Note: By default pull requests will use the template located at `.github/PULL_REQUEST_TEMPLATE.md`. But we also provide custom tailored templates located inside of `.github/PULL_REQUEST_TEMPLATE`. Sadly the GitHub Web Interface doesn't provide an easy way to select them as it does for issue templates (see this comment for more details).

To use them you need to add the following query parameters to the url when creating the pull request and hit enter:

- Feature PR: `?expand=1&template=feature_PR.md`
- Bug Fix PR: `?expand=1&template=bug_fix_PR`
- Documentation PR: `?expand=1&template=docs_PR.md`

14.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a *docstring*.
3. The pull request should work for Python 3.8 and 3.9 Check your Github Actions `https://github.com/<your_name_here>/pyglotaran/actions` and make sure that the tests pass for all supported Python versions.

14.4 Docstrings

We use numpy style docstrings, which can also be autogenerated from function/method signatures by extensions for your editor.

Some extensions for popular editors are:

- autodocstring (VS-Code)
- vim-python-docstring (Vim)

Note: If your pull request improves the docstring coverage (check `pre-commit run -a interrogate`), please raise the value of the interrogate setting `fail-under` in pyproject.toml. That way the next person will improve the docstring coverage as well and everyone can enjoy a better documentation.

Warning: As soon as all our docstrings are in proper shape we will enforce that it stays that way. If you want to check if your docstrings are fine you can use pydocstyle and darglint.

14.5 Tips

To run a subset of tests:

```
$ py.test tests.test_pyglotaran
```

14.6 Deprecations

Only maintainers are allowed to decide about deprecations, thus you should first open an issue and check back with them if they are ok with deprecating something.

To make deprecations as robust as possible and give users all needed information to adjust their code, we provide helper functions inside the module *glotaran.deprecation*.

The functions you most likely want to use are

- *deprecate()* for functions, methods and classes
- *warn_deprecated()* for call arguments
- *deprecate_module_attribute()* for module attributes
- *deprecate_submodule()* for modules
- *deprecate_dict_entry()* for dict entries
- *raise_deprecation_error()* if the original behavior cannot be maintained

Those functions not only make it easier to deprecate something, but they also check that that deprecations will be removed when they are due and that at least the imports in the warning work. Thus all deprecations need to be tested.

Tests for deprecations should be placed in `glotaran/deprecation/modules/test` which also provides the test helper functions `deprecation_warning_on_call_test_helper` and `changed_import_test_warn`. Since the tests for deprecation are mainly for maintainability and not to test the functionality (those tests should be in the appropriate place) `deprecation_warning_on_call_test_helper` will by default just test that a `GlotaranApiDeprecationWarning` was raised and ignore all raise `Exception` s. An exception to this rule is when adding back removed functionality (which shouldn't happen in the first place but might), which should be implemented in a file under `glotaran/deprecation/modules` and filenames should be like the relative import path from glotaran root, but with _ instead of `.`.

E.g. `glotaran.analysis.scheme` would map to `analysis_scheme.py`

The only exceptions to this rule are the root `__init__.py` which is named `glotaran_root.py` and testing changed imports which should be placed in `test_changed_imports.py`.

14.6.1 Deprecating a Function, method or class

Deprecating a function, method or class is as easy as adding the `deprecate` decorator to it. Other decorators (e.g. `@staticmethod` or `@classmethod`) should be placed both `deprecate` in order to work.

Listing 1: glotaran/some_module.py

```
from glotaran.deprecation import deprecate

@deprecate(
    deprecated_qual_name_usage="glotaran.some_module.function_to_deprecate(filename)",
```

(continues on next page)

(continued from previous page)

```
    new_qual_name_usage='glotaran.some_module.new_function(filename, format_name="legacy
↪")',
    to_be_removed_in_version="0.6.0",
)
def function_to_deprecate(*args, **kwargs):
    ...
```

14.6.2 Deprecating a call argument

When deprecating a call argument you should use `warn_deprecated` and set the argument to deprecate to a default value (e.g. `"deprecated"`) to check against. Note that for this use case we need to set `check_qual_names=(False, False)` which will deactivate the import testing. This might not always be possible, e.g. if the argument is positional only, so it might make more sense to deprecate the whole callable, just discuss what to do with our trusted maintainers.

Listing 2: glotaran/some_module.py

```
from glotaran.deprecation import deprecate

def function_to_deprecate(args1, new_arg="new_default_behavior", deprecated_arg=
↪"deprecated", **kwargs):
    if deprecated_arg != "deprecated":
        warn_deprecated(
            deprecated_qual_name_usage="deprecated_arg",
            new_qual_name_usage='new_arg="legacy"',
            to_be_removed_in_version="0.6.0",
            check_qual_names=(False, False)
        )
        new_arg = "legacy"
    ...
```

14.6.3 Deprecating a module attribute

Sometimes it might be necessary to remove an attribute (function, class, or constant) from a module to prevent circular imports or just to streamline the API. In those cases you would use `deprecate_module_attribute` inside a module `__getattr__` function definition. This will import the attribute from the new location and return it when an import or use is requested.

Listing 3: glotaran/old_package/__init__.py

```
def __getattr__(attribute_name: str):
    from glotaran.deprecation import deprecate_module_attribute

    if attribute_name == "deprecated_attribute":
        return deprecate_module_attribute(
            deprecated_qual_name="glotaran.old_package.deprecated_attribute",
            new_qual_name="glotaran.new_package.new_attribute_name",
            to_be_removed_in_version="0.6.0",
        )

    raise AttributeError(f"module {__name__} has no attribute {attribute_name}")
```

14.6.4 Deprecating a submodule

For a better logical structure, it might be needed to move modules to a different location in the project. In those cases, you would use `deprecate_submodule`, which imports the module from the new location, add it to `sys.modules` and as an attribute to the parent package.

Listing 4: glotaran/old_package/__init__.py

```
from glotaran.deprecation import deprecate_submodule

module_name = deprecate_submodule(
    deprecated_module_name="glotaran.old_package.module_name",
    new_module_name="glotaran.new_package.new_module_name",
    to_be_removed_in_version="0.6.0",
)
```

14.6.5 Deprecating dict entries

The possible dict deprecation actions are:

- Swapping of keys `{"foo": 1} -> {"bar": 1}` (done via `swap_keys=("foo", "bar")`)
- Replacing of matching values `{"foo": 1} -> {"foo": 2}` (done via `replace_rules=({"foo": 1}, {"foo": 2})`)
- Replacing of matching values and swapping of keys `{"foo": 1} -> {"bar": 2}` (done via `replace_rules=({"foo": 1}, {"bar": 2})`)

For full examples have a look at the examples from the docstring (*deprecate_dict_entry()*).

14.6.6 Deprecation Errors

In some cases deprecations cannot have a replacement with the original behavior maintained. This will be mostly the case when at this point in time and in the object hierarchy there isn't enough information available to calculate the appropriate values. Rather than using a 'dummy' value not to break the API, which could cause undefined behavior down the line, those cases should throw an error which informs the users about the new usage. In general this should only be used if it is unavoidable due to massive refactoring of the internal structure and tried to avoid by any means in a reasonable context.

If you have one of those rare cases you can use *raise_deprecation_error()*.

14.7 Testing Result consistency

To test the consistency of results locally you need to clone the pyglotaran-examples and run them:

```
$ git clone https://github.com/glotaran/pyglotaran-examples
$ cd pyglotaran-examples
$ python scripts/run_examples.py run-all --headless
```

Note: Make sure you got the the latest version (`git pull`) and are on the correct branch for both `pyglotaran` and `pyglotaran-examples`.

The results from the examples will be saved in you home folder under `pyglotaran_examples_results`. Those results than will be compared to the 'gold standard' defined by the maintainers.

To test the result consistency run:

```
$ pytest .github/test_result_consistency.py
```

If needed this will clone the 'gold standard' results to the folder `comparison-results`, update them and test your current results against them.

14.8 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst), the version number only needs to be changed in `glotaran/__init__.py`.

Then make a new release on GitHub and give the tag a proper name, e.g. `0.3.0` since might be included in a citation.

Github Actions will then deploy to PyPI if the tests pass.

CHAPTER

FIFTEEN

API DOCUMENTATION

The API Documentation for pyglotaran is automatically created from its docstrings.

glotaran	Glotaran package __init__.py

15.1 glotaran

Glotaran package __init__.py

Modules

glotaran.analysis	This package contains functions for model simulation and fitting.
glotaran.builtin	This package contains builtin plugins.
glotaran.cli	
glotaran.deprecation	Deprecation helpers and place to put deprecated implementations till removing.
glotaran.examples	
glotaran.io	Functions for data IO
glotaran.model	Glotaran Model Package
glotaran.parameter	The glotaran parameter package.
glotaran.plugin_system	Plugin system package containing all plugin related implementations.
glotaran.project	The glotaran project package.
glotaran.testing	Testing framework package for glotaran itself and plugins.
glotaran.typing	Glotaran specific typing module.
glotaran.utils	Glotaran utility function/class package.

15.1.1 analysis

This package contains functions for model simulation and fitting.

Modules

glotaran.analysis.nnls	Functions for calculating conditionally linear parameters and residual with the non-negative least-squares method.
glotaran.analysis.optimization_group	
glotaran.analysis.optimization_group_calculator	
glotaran.analysis.optimization_group_calculator_linked	
glotaran.analysis.optimization_group_calculator_unlinked	
glotaran.analysis.optimize	
glotaran.analysis.simulation	Functions for simulating a global analysis model.
glotaran.analysis.util	
glotaran.analysis.variable_projection	Functions for calculating conditionally linear parameters and residual with the variable projection method.

nnls

Functions for calculating conditionally linear parameters and residual with the non-negative least-squares method.

Functions

Summary

residual_nnls	Calculate the conditionally linear parameters and residual with the nnls method.

residual_nnls

glotaran.analysis.nnls.**residual_nnls**(*matrix: numpy.ndarray*, *data: numpy.ndarray*) → Tuple[numpy.ndarray, numpy.ndarray]

Calculate the conditionally linear parameters and residual with the nnls method.

nnls stands for 'non-negative least-squares'.

Parameters

- **matrix** – The model matrix.
- **data** (*np.ndarray*) – The data to analyze.

optimization_group

Classes

Summary

OptimizationGroup	Create OptimizationGroup instance from a scheme (`glotaran.analysis.scheme.Scheme`)

OptimizationGroup

class glotaran.analysis.optimization_group.**OptimizationGroup**(*scheme:* glotaran.project.scheme.Scheme, *dataset_group:* glotaran.model.dataset_group.DatasetGroup)

Bases: `object`

Create OptimizationGroup instance from a scheme (`glotaran.analysis.scheme.Scheme`)

Args:

scheme (Scheme): An instance of `glotaran.analysis.scheme.Scheme` which defines your model, parameters, and data

Attributes Summary

additional_penalty	
clps	
cost	
data	
dataset_models	
full_penalty	
matrices	
model	Property providing access to the used model
parameters	
reduced_clps	
reduced_matrices	
residuals	

continues on next page

Table 6 – continued from previous page

weighted_residuals

additional_penalty

OptimizationGroup.**additional_penalty**

clps

OptimizationGroup.**clps**

cost

OptimizationGroup.**cost**

data

OptimizationGroup.**data**

dataset_models

OptimizationGroup.**dataset_models**

full_penalty

OptimizationGroup.**full_penalty**

matrices

OptimizationGroup.**matrices**

model

OptimizationGroup.**model**

Property providing access to the used model

The model is a subclass of `glotaran.model.Model` decorated with the *@model* decorator `glotaran.model.model_decorator.model` For an example implementation see e.g. `glotaran.builtin.models.kinetic_spectrum`

Returns:

Model: A subclass of `glotaran.model.Model` The model must be decorated with the *@model* decorator `glotaran.model.model_decorator.model`

parameters

OptimizationGroup.**parameters**

reduced_clps

OptimizationGroup.**reduced_clps**

reduced_matrices

OptimizationGroup.**reduced_matrices**

residuals

OptimizationGroup.**residuals**

weighted_residuals

OptimizationGroup.**weighted_residuals**

Methods Summary

create_result_data	
create_result_dataset	
reset	Resets all results and *DatasetModels*.

create_result_data

OptimizationGroup.**create_result_data**(*parameter_history: ParameterHistory = None*, *copy: bool = True*, *success: bool = True*, *add_svd: bool = True*) → dict[str, xr.Dataset]

create_result_dataset

OptimizationGroup.**create_result_dataset**(*label: str*, *copy: bool = True*, *add_svd: bool = True*) → xarray.core.dataset.Dataset

reset

OptimizationGroup.**reset**()
Resets all results and *DatasetModels*. Use after updating parameters.

Methods Documentation

property additional_penalty: dict[str, list[float]]

property clps: dict[str, list[np.ndarray]]

property cost: float

create_result_data(*parameter_history: ParameterHistory = None*, *copy: bool = True*, *success: bool = True*, *add_svd: bool = True*) → dict[str, xr.Dataset]

create_result_dataset(*label: str*, *copy: bool = True*, *add_svd: bool = True*) → xarray.core.dataset.Dataset

property data: dict[str, xr.Dataset]

property dataset_models: dict[str, DatasetModel]

property full_penalty: numpy.ndarray

property matrices: dict[str, np.ndarray | list[np.ndarray]]

property model: *glotaran.model.model.Model*
Property providing access to the used model

The model is a subclass of `glotaran.model.Model` decorated with the *@model* decorator `glotaran.model.model_decorator.model` For an example implementation see e.g. `glotaran.builtin.models.kinetic_spectrum`
Returns:
Model: A subclass of `glotaran.model.Model` The model must be decorated with the *@model* decorator `glotaran.model.model_decorator.model`

property parameters: *glotaran.parameter.parameter_group.ParameterGroup*

property reduced_clps: dict[str, list[np.ndarray]]

property reduced_matrices: dict[str, np.ndarray] | dict[str, list[np.ndarray]] | list[np.ndarray]

reset()
Resets all results and *DatasetModels*. Use after updating parameters.

property residuals: dict[str, list[np.ndarray]]

property weighted_residuals: dict[str, list[np.ndarray]]

Exceptions

Exception Summary

InitialParameterError
ParameterNotInitializedError

InitialParameterError

exception glotaran.analysis.optimization_group.**InitialParameterError**

ParameterNotInitializedError

exception glotaran.analysis.optimization_group.**ParameterNotInitializedError**

optimization_group_calculator

Classes

Summary

OptimizationGroupCalculator	A Problem class

OptimizationGroupCalculator

class glotaran.analysis.optimization_group_calculator.**OptimizationGroupCalculator**(*group: OptimizationGroup*)

Bases: object

A Problem class

Methods Summary

calculate_full_penalty
calculate_matrices
calculate_residual

continues on next page

Table 10 – continued from previous page

create_index_dependent_result_dataset	Creates a result datasets for index dependent matrices.
create_index_independent_result_dataset	Creates a result datasets for index independent matrices.
prepare_result_creation	

calculate_full_penalty

OptimizationGroupCalculator.**calculate_full_penalty**() → numpy.ndarray

calculate_matrices

OptimizationGroupCalculator.**calculate_matrices**()

calculate_residual

OptimizationGroupCalculator.**calculate_residual**()

create_index_dependent_result_dataset

OptimizationGroupCalculator.**create_index_dependent_result_dataset**(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Creates a result datasets for index dependent matrices.

create_index_independent_result_dataset

OptimizationGroupCalculator.**create_index_independent_result_dataset**(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Creates a result datasets for index independent matrices.

prepare_result_creation

OptimizationGroupCalculator.**prepare_result_creation**()

Methods Documentation

calculate_full_penalty() → numpy.ndarray

calculate_matrices()

calculate_residual()

create_index_dependent_result_dataset(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Creates a result datasets for index dependent matrices.

create_index_independent_result_dataset(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Creates a result datasets for index independent matrices.

prepare_result_creation()

optimization_group_calculator_linked

Functions

Summary

combine_matrices

combine_matrices

glotaran.analysis.optimization_group_calculator_linked.**combine_matrices**(*matrices: list[CalculatedMatrix]*) → CalculatedMatrix

Classes

Summary

DatasetIndexModel	A model which contains a dataset label and index information.
DatasetIndexModelGroup	A model which contains information about a group of dataset with linked clp.
OptimizationGroupCalculatorLinked	A class to calculate a set of datasets with linked CLP.

DatasetIndexModel

class glotaran.analysis.optimization_group_calculator_linked.**DatasetIndexModel**(*label: str*, *indices: dict[str, int]*, *axis: dict[str, np.ndarray]*)

Bases: tuple

A model which contains a dataset label and index information.

Create new instance of DatasetIndexModel(label, indices, axis)

Attributes Summary

axis	Alias for field number 2
indices	Alias for field number 1
label	Alias for field number 0

axis

DatasetIndexModel.**axis: dict[str, np.ndarray]**
Alias for field number 2

indices

DatasetIndexModel.**indices: dict[str, int]**
Alias for field number 1

label

DatasetIndexModel.**label: str**
Alias for field number 0

Methods Summary

count	Return number of occurrences of value.
index	Return first index of value.

count

DatasetIndexModel.**count**(*value*, */*)
Return number of occurrences of value.

index

DatasetIndexModel.**index**(*value*, *start=0*, *stop=sys.maxsize*, */*)
Return first index of value.

Raises ValueError if the value is not present.

Methods Documentation

axis: dict[str, np.ndarray]
Alias for field number 2

count(*value*, */*)
Return number of occurrences of value.

index(*value*, *start=0*, *stop=sys.maxsize*, */*)
Return first index of value.

Raises ValueError if the value is not present.

indices: dict[str, int]
Alias for field number 1

label: str
Alias for field number 0

DatasetIndexModelGroup

class glotaran.analysis.optimization_group_calculator_linked.**DatasetIndexModelGroup**(*data: np.ndarray*, *weight: np.ndarray*, *has_scaling: bool*, *group: str*, *data_sizes: list[int]*, *dataset_models: list[*DatasetIndexM

Bases: tuple

A model which contains information about a group of dataset with linked clp.

Create new instance of DatasetIndexModelGroup(data, weight, has_scaling, group, data_sizes, dataset_models)

Attributes Summary

data	Alias for field number 0
data_sizes	Holds the sizes of the concatenated datasets.
dataset_models	Alias for field number 5
group	The concatenated labels of the involved datasets.
has_scaling	Indicates if at least one dataset in the group needs scaling.
weight	Alias for field number 1

data

DatasetIndexModelGroup.**data: np.ndarray**
Alias for field number 0

data_sizes

DatasetIndexModelGroup.**data_sizes: list[int]**
Holds the sizes of the concatenated datasets.

dataset_models

DatasetIndexModelGroup.**dataset_models: list[DatasetIndexModel]**
Alias for field number 5

group

DatasetIndexModelGroup.**group: str**
The concatenated labels of the involved datasets.

has_scaling

DatasetIndexModelGroup.**has_scaling: bool**
Indicates if at least one dataset in the group needs scaling.

weight

DatasetIndexModelGroup.**weight: np.ndarray**
Alias for field number 1

Methods Summary

count	Return number of occurrences of value.
index	Return first index of value.

count

DatasetIndexModelGroup.**count**(*value*, */*)
Return number of occurrences of value.

index

DatasetIndexModelGroup.**index**(*value*, *start=0*, *stop=sys.maxsize*, */*)
Return first index of value.

Raises ValueError if the value is not present.

Methods Documentation

count(*value*, */*)
Return number of occurrences of value.

data: np.ndarray
Alias for field number 0

data_sizes: list[int]
Holds the sizes of the concatenated datasets.

dataset_models: list[*DatasetIndexModel*]
Alias for field number 5

group: str
The concatenated labels of the involved datasets.

has_scaling: bool
Indicates if at least one dataset in the group needs scaling.

index(*value*, *start=0*, *stop=sys.maxsize*, */*)
Return first index of value.

Raises ValueError if the value is not present.

weight: np.ndarray
Alias for field number 1

OptimizationGroupCalculatorLinked

class glotaran.analysis.optimization_group_calculator_linked.**OptimizationGroupCalculatorLinked**(*grou*
Op-
ti-
miza
tion-
Grou

Bases: *glotaran.analysis.optimization_group_calculator.OptimizationGroupCalculator*

A class to calculate a set of datasets with linked CLP.

Attributes Summary

bag
groups

bag

OptimizationGroupCalculatorLinked.**bag**

groups

OptimizationGroupCalculatorLinked.**groups**

Methods Summary

calculate_full_penalty	
calculate_index_dependent_matrices	Calculates the index dependent model matrices.
calculate_index_independent_matrices	Calculates the index independent model matrices.
calculate_matrices	
calculate_residual	
create_index_dependent_result_dataset	Creates a result datasets for index dependent matrices.
create_index_independent_result_dataset	Creates a result datasets for index independent matrices.
init_bag	Initializes a grouped problem bag.
prepare_result_creation	

calculate_full_penalty

OptimizationGroupCalculatorLinked.**calculate_full_penalty**() → numpy.ndarray

calculate_index_dependent_matrices

OptimizationGroupCalculatorLinked.**calculate_index_dependent_matrices**() → tuple[dict[str, list[CalculatedMatrix]], list[CalculatedMatrix]]

Calculates the index dependent model matrices.

calculate_index_independent_matrices

OptimizationGroupCalculatorLinked.**calculate_index_independent_matrices**() → tuple[dict[str, CalculatedMatrix], dict[str, CalculatedMatrix]]

Calculates the index independent model matrices.

calculate_matrices

OptimizationGroupCalculatorLinked.**calculate_matrices**()

calculate_residual

OptimizationGroupCalculatorLinked.**calculate_residual**()

create_index_dependent_result_dataset

OptimizationGroupCalculatorLinked.**create_index_dependent_result_dataset**(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Creates a result datasets for index dependent matrices.

create_index_independent_result_dataset

OptimizationGroupCalculatorLinked.**create_index_independent_result_dataset**(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Creates a result datasets for index independent matrices.

init_bag

OptimizationGroupCalculatorLinked.**init_bag**()
Initializes a grouped problem bag.

prepare_result_creation

OptimizationGroupCalculatorLinked.**prepare_result_creation**()

Methods Documentation

property bag: Deque[*glotaran.analysis.optimization_group_calculator_linked.DatasetIndexModelGroup*]

calculate_full_penalty() → numpy.ndarray

calculate_index_dependent_matrices() → tuple[dict[str, list[CalculatedMatrix]], list[CalculatedMatrix]]
Calculates the index dependent model matrices.

calculate_index_independent_matrices() → tuple[dict[str, CalculatedMatrix], dict[str, CalculatedMatrix]]
Calculates the index independent model matrices.

calculate_matrices()

calculate_residual()

create_index_dependent_result_dataset(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset
Creates a result datasets for index dependent matrices.

create_index_independent_result_dataset(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset
Creates a result datasets for index independent matrices.

property groups: dict[str, list[str]]

init_bag()
Initializes a grouped problem bag.

prepare_result_creation()

optimization_group_calculator_unlinked

Classes

Summary

OptimizationGroupCalculatorUnlinked	Represents a problem where the clps are not linked.

OptimizationGroupCalculatorUnlinked

class glotaran.analysis.optimization_group_calculator_unlinked.**OptimizationGroupCalculatorUnlinked**(

Bases: *glotaran.analysis.optimization_group_calculator.OptimizationGroupCalculator*

Represents a problem where the clps are not linked.

Attributes Summary

global_matrices	

global_matrices

OptimizationGroupCalculatorUnlinked.**global_matrices**

Methods Summary

calculate_full_penalty	
calculate_matrices	Calculates the model matrices.
calculate_residual	Calculates the residuals.
create_index_dependent_result_dataset	Creates a result datasets for index dependent matrices.
create_index_independent_result_dataset	Creates a result datasets for index independent matrices.
prepare_result_creation	

calculate_full_penalty

OptimizationGroupCalculatorUnlinked.**calculate_full_penalty**() → numpy.ndarray

calculate_matrices

OptimizationGroupCalculatorUnlinked.**calculate_matrices**() → tuple[dict[str, CalculatedMatrix | list[CalculatedMatrix]], dict[str, CalculatedMatrix | list[CalculatedMatrix]]]

Calculates the model matrices.

calculate_residual

OptimizationGroupCalculatorUnlinked.**calculate_residual**() → tuple[dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]]]

Calculates the residuals.

create_index_dependent_result_dataset

OptimizationGroupCalculatorUnlinked.**create_index_dependent_result_dataset**(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Creates a result datasets for index dependent matrices.

create_index_independent_result_dataset

OptimizationGroupCalculatorUnlinked.**create_index_independent_result_dataset**(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Creates a result datasets for index independent matrices.

prepare_result_creation

OptimizationGroupCalculatorUnlinked.**prepare_result_creation**()

Methods Documentation

calculate_full_penalty() → numpy.ndarray

calculate_matrices() → tuple[dict[str, CalculatedMatrix | list[CalculatedMatrix]], dict[str, CalculatedMatrix | list[CalculatedMatrix]]]

Calculates the model matrices.

calculate_residual() → tuple[dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]], dict[str, list[np.ndarray]]]

Calculates the residuals.

create_index_dependent_result_dataset(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Creates a result datasets for index dependent matrices.

create_index_independent_result_dataset(*label: str*, *dataset: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Creates a result datasets for index independent matrices.

property global_matrices: dict[str, CalculatedMatrix]

prepare_result_creation()

optimize

Functions

Summary

optimize

optimize

glotaran.analysis.optimize.**optimize**(*scheme:* glotaran.project.scheme.Scheme, *verbose: bool = True*, *raise_exception: bool = False*) → *glotaran.project.result.Result*

simulation

Functions for simulating a global analysis model.

Functions

Summary

simulate	Simulates a model.
simulate_clp	
simulate_global_model	Simulates a global model.

simulate

glotaran.analysis.simulation.**simulate**(*model: Model*, *dataset: str*, *parameters: ParameterGroup*, *coordinates: dict[str, np.ndarray]*, *clp: xr.DataArray | None = None*, *noise: bool = False*, *noise_std_dev: float = 1.0*, *noise_seed: int | None = None*)

Simulates a model.

Parameters

- **model** – The model to simulate.
- **parameter** – The parameters for the simulation.
- **dataset** – Label of the dataset to simulate
- **axes** – A dictionary with axes for simulation.
- **clp** – conditionally linear parameters. Will be used instead of *model.global_matrix* if given.
- **noise** – Add noise to the simulation.
- **noise_std_dev** – The standard deviation for noise simulation.
- **noise_seed** – The seed for the noise simulation.

simulate_clp

glotaran.analysis.simulation.**simulate_clp**(*dataset_model: DatasetModel*, *parameters: ParameterGroup*, *clp: xr.DataArray*)

simulate_global_model

glotaran.analysis.simulation.**simulate_global_model**(*dataset_model: DatasetModel*, *parameters: ParameterGroup*, *clp: xr.DataArray = None*)

Simulates a global model.

util

Functions

Summary

apply_constraints	
apply_relations	
apply_weight	
calculate_clp_penalties	
calculate_matrix	
combine_matrix	
find_closest_index	
find_overlap	
get_idx_from_interval	Retrieves start and end index of an interval on some axis :param interval: :type interval: A tuple of floats with begin and end of the interval :param axis: :type axis: Array like object which can be cast to np.array
get_min_max_from_interval	
reduce_matrix	
retrieve_clps	

apply_constraints

glotaran.analysis.util.**apply_constraints**(*matrix: CalculatedMatrix*, *model: Model*, *index: Any | None*) → CalculatedMatrix

apply_relations

glotaran.analysis.util.**apply_relations**(*matrix: CalculatedMatrix*, *model: Model*, *parameters: ParameterGroup*, *index: Any | None*) → CalculatedMatrix

apply_weight

glotaran.analysis.util.**apply_weight**(*matrix*, *weight*)

calculate_clp_penalties

glotaran.analysis.util.**calculate_clp_penalties**(*model: Model*, *parameters: ParameterGroup*, *clp_labels: list[list[str]] | list[str]*, *clps: list[np.ndarray]*, *global_axis: np.ndarray*, *dataset_models: dict[str, DatasetModel]*) → np.ndarray

calculate_matrix

glotaran.analysis.util.**calculate_matrix**(*dataset_model: DatasetModel*, *indices: dict[str, int]*, *as_global_model: bool = False*) → CalculatedMatrix

combine_matrix

glotaran.analysis.util.**combine_matrix**(*matrix*, *this_matrix*, *clp_labels*, *this_clp_labels*)

find_closest_index

glotaran.analysis.util.**find_closest_index**(*index: float*, *axis: numpy.ndarray*)

find_overlap

glotaran.analysis.util.**find_overlap**(*a*, *b*, *rtol=1e-05*, *atol=1e-08*)

get_idx_from_interval

glotaran.analysis.util.**get_idx_from_interval**(*interval: tuple[float, float]*, *axis: np.ndarray*) → tuple[int, int]

Retrieves start and end index of an interval on some axis :param interval: :type interval: A tuple of floats with begin and end of the interval :param axis: :type axis: Array like object which can be cast to np.array

Returns start, end

Return type tuple of int

get_min_max_from_interval

glotaran.analysis.util.**get_min_max_from_interval**(*interval*, *axis*)

reduce_matrix

glotaran.analysis.util.**reduce_matrix**(*matrix: CalculatedMatrix*, *model: Model*, *parameters: ParameterGroup*, *index: Any | None*) → CalculatedMatrix

retrieve_clps

glotaran.analysis.util.**retrieve_clps**(*model: Model*, *parameters: ParameterGroup*, *clp_labels: xr.DataArray*, *reduced_clp_labels: xr.DataArray*, *reduced_clps: xr.DataArray*, *index: Any | None*) → xr.DataArray

Classes

Summary

CalculatedMatrix

CalculatedMatrix

class glotaran.analysis.util.**CalculatedMatrix**(*clp_labels*, *matrix*)

Bases: tuple

Create new instance of CalculatedMatrix(clp_labels, matrix)

Attributes Summary

clp_labels	Alias for field number 0
matrix	Alias for field number 1

clp_labels

CalculatedMatrix.**clp_labels: list[str]**

Alias for field number 0

matrix

CalculatedMatrix.**matrix: np.ndarray**

Alias for field number 1

Methods Summary

count	Return number of occurrences of value.
index	Return first index of value.

count

CalculatedMatrix.**count**(*value*, */*)

Return number of occurrences of value.

index

CalculatedMatrix.**index**(*value*, *start=0*, *stop=sys.maxsize*, /)
Return first index of value.

Raises ValueError if the value is not present.

Methods Documentation

clp_labels: list[str]
Alias for field number 0

count(*value*, /)
Return number of occurrences of value.

index(*value*, *start=0*, *stop=sys.maxsize*, /)
Return first index of value.

Raises ValueError if the value is not present.

matrix: np.ndarray
Alias for field number 1

variable_projection

Functions for calculating conditionally linear parameters and residual with the variable projection method.

Functions

Summary

residual_variable_projection	Calculates the conditionally linear parameters and residual with the variable projection method.

residual_variable_projection

glotaran.analysis.variable_projection.**residual_variable_projection**(*matrix: numpy.ndarray*, *data: numpy.ndarray*) → Tuple[numpy.ndarray, numpy.ndarray]

Calculates the conditionally linear parameters and residual with the variable projection method.

Parameters

- **matrix** – The model matrix.
- **data** (*np.ndarray*) – The data to analyze.

15.1.2 builtin

This package contains builtin plugins.

Modules

glotaran.builtin.io
glotaran.builtin.megacomplexes

io

Modules

glotaran.builtin.io.ascii	
glotaran.builtin.io.csv	
glotaran.builtin.io.folder	Plugin to dump pyglotaran object as files in a folder.
glotaran.builtin.io.netCDF	
glotaran.builtin.io.sdt	
glotaran.builtin.io.yml	

ascii

Modules

glotaran.builtin.io.ascii.wavelength_time_explicit_file

wavelength_time_explicit_file

Functions

Summary

get_data_file_format
get_interval_number

get_data_file_format

glotaran.builtin.io.ascii.wavelength_time_explicit_file.**get_data_file_format**(*line*)

get_interval_number

glotaran.builtin.io.ascii.wavelength_time_explicit_file.**get_interval_number**(*line*)

Classes

Summary

AsciiDataIo	Initialize a Data IO plugin with the name of the format.
DataFileType	An enumeration.
ExplicitFile	Abstract class representing either a time- or wavelength-explicit file.
TimeExplicitFile	Represents a time explicit file
WavelengthExplicitFile	Represents a wavelength explicit file

AsciiDataIo

class glotaran.builtin.io.ascii.wavelength_time_explicit_file.**AsciiDataIo**(*format_name: str*)

Bases: *glotaran.io.interface.DataIoInterface*

Initialize a Data IO plugin with the name of the format.

Parameters **format_name** (*str*) – Name of the supported format an instance uses.

Methods Summary

load_dataset	Reads an ascii file in wavelength- or time-explicit format.
save_dataset	Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

load_dataset

AsciiDataIo.**load_dataset**(*file_name: str*) → xr.Dataset | xr.DataArray

Reads an ascii file in wavelength- or time-explicit format.

See [1] for documentation of this format.

Parameters **fname** (*str*) – Name of the ascii file.
Returns **dataset**
Return type xr.Dataset

Notes

save_dataset

AsciiDataIo.**save_dataset**(*dataset: xarray.core.dataarray.DataArray*, *file_name: str*, *, *comment: str = ''*, *file_format: glotaran.builtin.io.ascii.wavelength_time_explicit_file.DataFileType = DataFileType.time_explicit*, *number_format: str = '%.10e'*)

Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

Parameters

- **dataset** (*xr.Dataset*) – Dataset to be saved to file.
- **file_name** (*str*) – File to write the data to.

Methods Documentation

load_dataset(*file_name: str*) → xr.Dataset | xr.DataArray

Reads an ascii file in wavelength- or time-explicit format.

See [1] for documentation of this format.

Parameters **fname** (*str*) – Name of the ascii file.
Returns **dataset**
Return type xr.Dataset

Notes

save_dataset(*dataset: xarray.core.dataarray.DataArray*, *file_name: str*, *, *comment: str = ''*, *file_format: glotaran.builtin.io.ascii.wavelength_time_explicit_file.DataFileType = DataFileType.time_explicit*, *number_format: str = '%.10e'*)

Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

Parameters

- **dataset** (*xr.Dataset*) – Dataset to be saved to file.
- **file_name** (*str*) – File to write the data to.

DataFileType

class glotaran.builtin.io.ascii.wavelength_time_explicit_file.**DataFileType**(*value*)

Bases: enum.Enum

An enumeration.

Attributes Summary

time_explicit
wavelength_explicit

time_explicit

DataFileType.**time_explicit = 'Time explicit'**

wavelength_explicit

DataFileType.**wavelength_explicit = 'Wavelength explicit'**

time_explicit = 'Time explicit'

wavelength_explicit = 'Wavelength explicit'

ExplicitFile

class glotaran.builtin.io.ascii.wavelength_time_explicit_file.**ExplicitFile**(*filepath: Optional[str] = None*, *dataset: Optional[xarray.core.dataarray.D = None*)

Bases: object

Abstract class representing either a time- or wavelength-explicit file.

Methods Summary

dataset
get_data_row
get_explicit_axis
get_format_name
get_observations
get_secondary_axis
read
set_explicit_axis
write

dataset

ExplicitFile.**dataset**(*prepare: bool = True*) → xr.Dataset | xr.DataArray

get_data_row

ExplicitFile.**get_data_row**(*index*)

get_explicit_axis

ExplicitFile.**get_explicit_axis**()

get_format_name

ExplicitFile.**get_format_name**()

get_observations

ExplicitFile.**get_observations**(*index*)

get_secondary_axis

ExplicitFile.**get_secondary_axis**()

read

ExplicitFile.**read**(*prepare: bool = True*)

set_explicit_axis

ExplicitFile.**set_explicit_axis**(*axis*)

write

ExplicitFile.**write**(*overwrite=False*, *comment=''*, *file_format=DataFileType.time_explicit*, *number_format='%.10e'*)

Methods Documentation

dataset(*prepare: bool = True*) → xr.Dataset | xr.DataArray

get_data_row(*index*)

get_explicit_axis()

get_format_name()

get_observations(*index*)

get_secondary_axis()

read(*prepare: bool = True*)

set_explicit_axis(*axis*)

write(*overwrite=False*, *comment=''*, *file_format=DataFileType.time_explicit*, *number_format='%.10e'*)

TimeExplicitFile

class glotaran.builtin.io.ascii.wavelength_time_explicit_file.**TimeExplicitFile**(*filepath: Optional[str] = None*, *dataset: Optional[xarray.core.dataar = None*)

Bases: *glotaran.builtin.io.ascii.wavelength_time_explicit_file.ExplicitFile*

Represents a time explicit file

Methods Summary

add_data_row
dataset
get_data_row
get_explicit_axis
get_format_name
get_observations
get_secondary_axis
read
set_explicit_axis
write

add_data_row

TimeExplicitFile.**add_data_row**(*row*)

dataset

TimeExplicitFile.**dataset**(*prepare: bool = True*) → xr.Dataset | xr.DataArray

get_data_row

TimeExplicitFile.**get_data_row**(*index*)

get_explicit_axis

TimeExplicitFile.**get_explicit_axis**()

get_format_name

TimeExplicitFile.**get_format_name**()

get_observations

TimeExplicitFile.**get_observations**(*index*)

get_secondary_axis

TimeExplicitFile.**get_secondary_axis**()

read

TimeExplicitFile.**read**(*prepare: bool = True*)

set_explicit_axis

TimeExplicitFile.**set_explicit_axis**(*axies*)

write

TimeExplicitFile.**write**(*overwrite=False*, *comment=''*,
file_format=DataFileType.time_explicit, *number_format='%.10e'*)

Methods Documentation

add_data_row(*row*)

dataset(*prepare: bool = True*) → xr.Dataset | xr.DataArray

get_data_row(*index*)

get_explicit_axis()

get_format_name()

get_observations(*index*)

get_secondary_axis()

read(*prepare: bool = True*)

set_explicit_axis(*axies*)

write(*overwrite=False*, *comment=''*, *file_format=DataFileType.time_explicit*,
number_format='%.10e')

WavelengthExplicitFile

class glotaran.builtin.io.ascii.wavelength_time_explicit_file.**WavelengthExplicitFile**(*filepath: Optional[str] = None*, *dataset: Optional[xarray.core = None*)

Bases: *glotaran.builtin.io.ascii.wavelength_time_explicit_file.ExplicitFile*

Represents a wavelength explicit file

Methods Summary

add_data_row
dataset
get_data_row
get_explicit_axis
get_format_name
get_observations
get_secondary_axis
read
set_explicit_axis
times
wavelengths
write

add_data_row

WavelengthExplicitFile.**add_data_row**(*row*)

dataset

WavelengthExplicitFile.**dataset**(*prepare: bool = True*) → xr.Dataset | xr.DataArray

get_data_row

WavelengthExplicitFile.**get_data_row**(*index*)

get_explicit_axis

WavelengthExplicitFile.**get_explicit_axis**()

get_format_name

WavelengthExplicitFile.**get_format_name**()

get_observations

WavelengthExplicitFile.**get_observations**(*index*)

get_secondary_axis

WavelengthExplicitFile.**get_secondary_axis**()

read

WavelengthExplicitFile.**read**(*prepare: bool = True*)

set_explicit_axis

WavelengthExplicitFile.**set_explicit_axis**(*axis*)

times

WavelengthExplicitFile.**times**()

wavelengths

WavelengthExplicitFile.**wavelengths**()

write

WavelengthExplicitFile.**write**(*overwrite=False*, *comment=''*, *file_format=DataFileType.time_explicit*, *number_format='%.10e'*)

Methods Documentation

add_data_row(*row*)

dataset(*prepare: bool = True*) → xr.Dataset | xr.DataArray

get_data_row(*index*)

get_explicit_axis()

get_format_name()

get_observations(*index*)

get_secondary_axis()

read(*prepare: bool = True*)

set_explicit_axis(*axis*)

times()

wavelengths()

write(*overwrite=False*, *comment=''*, *file_format=DataFileType.time_explicit*, *number_format='%.10e'*)

csv

Modules

glotaran.builtin.io.csv.csv

csv

Classes

Summary

CsvProjectIo	Initialize a Project IO plugin with the name of the format.

CsvProjectIo

class glotaran.builtin.io.csv.csv.**CsvProjectIo**(*format_name: str*)

Bases: *glotaran.io.interface.ProjectIoInterface*

Initialize a Project IO plugin with the name of the format.

Parameters **format_name** (*str*) – Name of the supported format an instance uses.

Methods Summary

load_model	Create a Model instance from the specs defined in a file (**NOT IMPLEMENTED**).
load_parameters	Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).
load_result	Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).
load_scheme	Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).
save_model	Save a Model instance to a spec file (**NOT IMPLEMENTED**).
save_parameters	Save a ParameterGroup to a CSV file.

continues on next page

Table 41 – continued from previous page

save_result	Save a Result instance to a spec file (**NOT IMPLEMENTED**).
save_scheme	Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

load_model

CsvProjectIo.**load_model**(*file_name: str*) → Model

Create a Model instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the model specs.

Returns Model instance created from the file.

Return type *Model*

load_parameters

CsvProjectIo.**load_parameters**(*file_name: str*) → *glotaran.parameter.parameter_group.ParameterGroup*

Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns ParameterGroup instance created from the file.

Return type *ParameterGroup*

load_result

CsvProjectIo.**load_result**(*result_path: str*) → Result

Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **result_path** (*str*) – Path containing the result data.

Returns Result instance created from the file.

Return type *Result*

load_scheme

CsvProjectIo.**load_scheme**(*file_name: str*) → Scheme

Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns

- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model

CsvProjectIo.**save_model**(*model: Model*, *file_name: str*)

Save a Model instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **model** (Model) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters

CsvProjectIo.**save_parameters**(*parameters:* glotaran.parameter.parameter_group.ParameterGroup, *file_name: str*, *, *sep=','*)

Save a ParameterGroup to a CSV file.

save_result

CsvProjectIo.**save_result**(*result: Result*, *result_path: str*) → list[str] | None

Save a Result instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **result** (Result) – Result instance to save to specs file.
- **result_path** (*str*) – Path to write the result data to.

save_scheme

CsvProjectIo.**save_scheme**(*scheme: Scheme*, *file_name: str*)

Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

Methods Documentation

load_model(*file_name: str*) → Model

Create a Model instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the model specs.
Returns Model instance created from the file.
Return type *Model*

load_parameters(*file_name: str*) → *glotaran.parameter.parameter_group.ParameterGroup*

Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.
Returns ParameterGroup instance created from the file.
Return type *ParameterGroup*

load_result(*result_path: str*) → Result

Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **result_path** (*str*) – Path containing the result data.
Returns Result instance created from the file.
Return type *Result*

load_scheme(*file_name: str*) → Scheme
Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).
Parameters **file_name** (*str*) – File containing the parameter specs.
Returns
- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model(*model: Model*, *file_name: str*)
Save a Model instance to a spec file (**NOT IMPLEMENTED**).
Parameters
- **model** (Model) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters(*parameters:* glotaran.parameter.parameter_group.ParameterGroup, *file_name: str*, *, *sep=','*)
Save a ParameterGroup to a CSV file.

save_result(*result: Result*, *result_path: str*) → list[str] | None
Save a Result instance to a spec file (**NOT IMPLEMENTED**).
Parameters
- **result** (Result) – Result instance to save to specs file.
- **result_path** (*str*) – Path to write the result data to.

save_scheme(*scheme: Scheme*, *file_name: str*)
Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).
Parameters
- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

folder

Plugin to dump pyglotaran object as files in a folder.

Modules

glotaran.builtin.io.folder.folder_plugin	Implementation of the folder Io plugin.

folder_plugin

Implementation of the folder Io plugin.

The current implementation is an exact copy of how `Result.save(path)` worked in glotaran 0.3.x and meant as an compatibility function.

Classes

Summary

FolderProjectIo	Project Io plugin to save result data to a folder.

FolderProjectIo

class glotaran.builtin.io.folder.folder_plugin.**FolderProjectIo**(*format_name: str*)

Bases: *glotaran.io.interface.ProjectIoInterface*

Project Io plugin to save result data to a folder.

There won't be a serialization of the Result object, but simply a markdown summary output and the important data saved to files.

Initialize a Project IO plugin with the name of the format.

Parameters **format_name** (*str*) – Name of the supported format an instance uses.

Methods Summary

load_model	Create a Model instance from the specs defined in a file (**NOT IMPLEMENTED**).
load_parameters	Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).
load_result	Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).
load_scheme	Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).
save_model	Save a Model instance to a spec file (**NOT IMPLEMENTED**).
save_parameters	Save a ParameterGroup instance to a spec file (**NOT IMPLEMENTED**).
save_result	Save the result to a given folder.
save_scheme	Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

load_model

FolderProjectIo.**load_model**(*file_name: str*) → Model

Create a Model instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the model specs.

Returns Model instance created from the file.

Return type *Model*

load_parameters

FolderProjectIo.**load_parameters**(*file_name: str*) → ParameterGroup
Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).
Parameters **file_name** (*str*) – File containing the parameter specs.
Returns ParameterGroup instance created from the file.
Return type *ParameterGroup*

load_result

FolderProjectIo.**load_result**(*result_path: str*) → Result
Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).
Parameters **result_path** (*str*) – Path containing the result data.
Returns Result instance created from the file.
Return type *Result*

load_scheme

FolderProjectIo.**load_scheme**(*file_name: str*) → Scheme
Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).
Parameters **file_name** (*str*) – File containing the parameter specs.
Returns
- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model

FolderProjectIo.**save_model**(*model: Model*, *file_name: str*)
Save a Model instance to a spec file (**NOT IMPLEMENTED**).
Parameters
- **model** (Model) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters

FolderProjectIo.**save_parameters**(*parameters: ParameterGroup*, *file_name: str*)
Save a ParameterGroup instance to a spec file (**NOT IMPLEMENTED**).
Parameters
- **parameters** (ParameterGroup) – ParameterGroup instance to save to specs file.
- **file_name** (*str*) – File to write the parameter specs to.

save_result

FolderProjectIo.**save_result**(*result: Result*, *result_path: str*, *, *saving_options: SavingOptions = SavingOptions(data_filter=None, data_format='nc', parameter_format='csv', report=True)*) → list[str]

Save the result to a given folder.

Returns a list with paths of all saved items. The following files are saved if not configured otherwise: * *result.md*: The result with the model formatted as markdown text. * *model.yml*: Model spec file. * *scheme.yml*: Scheme spec file. * *initial_parameters.csv*: Initially used parameters. * *optimized_parameters.csv*: The optimized parameter as csv file. * *parameter_history.csv*: Parameter changes over the optimization * *{dataset_label}.nc*: The result data for each dataset as NetCDF file.

Note: As a side effect it populates the file path properties of `result` which can be used in other plugins (e.g. the `yml` save_result).

Parameters
- **result** (Result) – Result instance to be saved.
- **result_path** (*str*) – The path to the folder in which to save the result.
- **saving_options** (SavingOptions) – Options for saving the the result.

Returns List of file paths which were created.
Return type list[str]
Raises **ValueError** – If `result_path` is a file.

save_scheme

FolderProjectIo.**save_scheme**(*scheme: Scheme*, *file_name: str*)

Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

Parameters
- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

Methods Documentation

load_model(*file_name: str*) → Model

Create a Model instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the model specs.
Returns Model instance created from the file.
Return type *Model*

load_parameters(*file_name: str*) → ParameterGroup

Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.
Returns ParameterGroup instance created from the file.
Return type *ParameterGroup*

load_result(*result_path: str*) → Result

Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **result_path** (*str*) – Path containing the result data.
Returns Result instance created from the file.

Return type *Result*

`load_scheme`(*file_name: str*) → Scheme

Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns

- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

`save_model`(*model: Model*, *file_name: str*)

Save a Model instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **model** (Model) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

`save_parameters`(*parameters: ParameterGroup*, *file_name: str*)

Save a ParameterGroup instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **parameters** (ParameterGroup) – ParameterGroup instance to save to specs file.
- **file_name** (*str*) – File to write the parameter specs to.

`save_result`(*result: Result*, *result_path: str*, ***, *saving_options: SavingOptions = SavingOptions(data_filter=None, data_format='nc', parameter_format='csv', report=True)*) → list[str]

Save the result to a given folder.

Returns a list with paths of all saved items. The following files are saved if not configured otherwise: * *result.md*: The result with the model formatted as markdown text. * *model.yml*: Model spec file. * *scheme.yml*: Scheme spec file. * *initial_parameters.csv*: Initially used parameters. * *optimized_parameters.csv*: The optimized parameter as csv file. * *parameter_history.csv*: Parameter changes over the optimization * *{dataset_label}.nc*: The result data for each dataset as NetCDF file.

Note: As a side effect it populates the file path properties of `result` which can be used in other plugins (e.g. the `yml` save_result).

Parameters

- **result** (Result) – Result instance to be saved.
- **result_path** (*str*) – The path to the folder in which to save the result.
- **saving_options** (SavingOptions) – Options for saving the the result.

Returns List of file paths which were created.

Return type list[str]

Raises **ValueError** – If `result_path` is a file.

`save_scheme`(*scheme: Scheme*, *file_name: str*)

Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

netCDF

Modules

glotaran.builtin.io.netCDF.netCDF

netCDF

Classes

Summary

NetCDFDataIo	Initialize a Data IO plugin with the name of the format.

NetCDFDataIo

class glotaran.builtin.io.netCDF.netCDF.**NetCDFDataIo**(*format_name: str*)

Bases: *glotaran.io.interface.DataIoInterface*

Initialize a Data IO plugin with the name of the format.

Parameters **format_name** (*str*) – Name of the supported format an instance uses.

Methods Summary

load_dataset	Read data from a file to xarray.Dataset or xarray.DataArray (**NOT IMPLEMENTED**).
save_dataset	Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

load_dataset

NetCDFDataIo.**load_dataset**(*file_name: str*) → xr.Dataset | xr.DataArray

Read data from a file to xarray.Dataset or xarray.DataArray (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the data.

Returns Data loaded from the file.

Return type xr.Dataset|xr.DataArray

save_dataset

NetCDFDataIo.**save_dataset**(*dataset: xr.Dataset*, *file_name: str*, ***, *data_filters: list[str] | None = None*)

Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

Parameters

- **dataset** (*xr.Dataset*) – Dataset to be saved to file.
- **file_name** (*str*) – File to write the data to.

Methods Documentation

load_dataset(*file_name: str*) → xr.Dataset | xr.DataArray

Read data from a file to xarray.Dataset or xarray.DataArray (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the data.

Returns Data loaded from the file.

Return type xr.Dataset|xr.DataArray

save_dataset(*dataset: xr.Dataset*, *file_name: str*, ***, *data_filters: list[str] | None = None*)

Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

Parameters

- **dataset** (*xr.Dataset*) – Dataset to be saved to file.
- **file_name** (*str*) – File to write the data to.

sdt

Modules

glotaran.builtin.io.sdt.sdt_file_reader	Glotarans module to read files

sdt_file_reader

Glotarans module to read files

Classes

Summary

SdtDataIo	Initialize a Data IO plugin with the name of the format.

SdtDataIo

class glotaran.builtin.io.sdt.sdt_file_reader.**SdtDataIo**(*format_name: str*)
Bases: *glotaran.io.interface.DataIoInterface*

Initialize a Data IO plugin with the name of the format.

Parameters **format_name** (*str*) – Name of the supported format an instance uses.

Methods Summary

load_dataset	Reads a **.sdt* file and returns a pd.DataFrame (*return_dataframe==True*), a SpectralTemporalDataset (*type_of_data=='st'*) or a FLIMDataset (*type_of_data=='flim'*).
save_dataset	Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

load_dataset

SdtDataIo.**load_dataset**(*file_name: str*, *, *index: np.ndarray | None = None*, *flim: bool = False*, *dataset_index: int | None = None*, *swap_axis: bool = False*, *orig_time_axis_index: int = 2*) → xr.Dataset

Reads a **.sdt* file and returns a pd.DataFrame (*return_dataframe==True*), a SpectralTemporalDataset (*type_of_data=='st'*) or a FLIMDataset (*type_of_data=='flim'*).

Parameters

- **file_name** (*str*) – Path to the sdt file which should be read.
- **index** (*list, np.ndarray*) – This is only needed if *type_of_data=="st"*, since **.sdt* files, which only contain spectral temporal data, lack the spectral information. Thus for the spectral axis data need to be given by the user.
- **flim** – Set true if reading a result from a FLIM measurement.
- **dataset_index** (*int: default 0*) – If the **.sdt* file contains multiple datasets the index will used to select the wanted one
- **swap_axis** (*bool, default False*) – Flag to switch a wavelength explicit *input_df* to time explicit *input_df*, before generating the SpectralTemporalDataset.
- **orig_time_axis_index** (*int*) – Index of the axis which corresponds to the time axis. I.e. for data of shape (64, 64, 256), which are a 64x64 pixel map with 256 time steps, orig_time_axis_index=2.

Raises **IndexError:** – If the length of the index array is incompatible with the data.

save_dataset

SdtDataIo.**save_dataset**(*dataset: xr.Dataset | xr.DataArray*, *file_name: str*)

Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

Parameters

- **dataset** (*xr.Dataset*) – Dataset to be saved to file.
- **file_name** (*str*) – File to write the data to.

Methods Documentation

load_dataset(*file_name: str*, *, *index: np.ndarray | None = None*, *flim: bool = False*, *dataset_index: int | None = None*, *swap_axis: bool = False*, *orig_time_axis_index: int = 2*) → xr.Dataset

Reads a **.sdt* file and returns a pd.DataFrame (*return_dataframe==True*), a SpectralTemporalDataset (*type_of_data=='st'*) or a FLIMDataset (*type_of_data=='flim'*).

Parameters

- **file_name** (*str*) – Path to the sdt file which should be read.
- **index** (*list, np.ndarray*) – This is only needed if *type_of_data=="st"*, since **.sdt* files, which only contain spectral temporal data, lack the spectral information. Thus for the spectral axis data need to be given by the user.
- **flim** – Set true if reading a result from a FLIM measurement.
- **dataset_index** (*int: default 0*) – If the **.sdt* file contains multiple datasets the index will used to select the wanted one
- **swap_axis** (*bool, default False*) – Flag to switch a wavelength explicit *input_df* to time explicit *input_df*, before generating the SpectralTemporalDataset.
- **orig_time_axis_index** (*int*) – Index of the axis which corresponds to the time axis. I.e. for data of shape (64, 64, 256), which are a 64x64 pixel map with 256 time steps, orig_time_axis_index=2.

Raises **IndexError:** – If the length of the index array is incompatible with the data.

save_dataset(*dataset: xr.Dataset | xr.DataArray*, *file_name: str*)

Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

Parameters

- **dataset** (*xr.Dataset*) – Dataset to be saved to file.
- **file_name** (*str*) – File to write the data to.

yml

Modules

glotaran.builtin.io.yml.yml

yml

Classes

Summary

YmlProjectIo	Initialize a Project IO plugin with the name of the format.

YmlProjectIo

class glotaran.builtin.io.yml.yml.**YmlProjectIo**(*format_name: str*)
Bases: *glotaran.io.interface.ProjectIoInterface*

Initialize a Project IO plugin with the name of the format.

Parameters **format_name** (*str*) – Name of the supported format an instance uses.

Methods Summary

load_model	parse_yaml_file reads the given file and parses its content as YML.
load_parameters	Create a ParameterGroup instance from the specs defined in a file.
load_result	Create a `Result` instance from the specs defined in a file.
load_scheme	Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).
save_model	Save a Model instance to a spec file.
save_parameters	Save a ParameterGroup instance to a spec file (**NOT IMPLEMENTED**).
save_result	Write a `Result` instance to a spec file.
save_scheme	Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

load_model

YmlProjectIo.**load_model**(*file_name: str*) → *glotaran.model.model.Model*
parse_yaml_file reads the given file and parses its content as YML.
Parameters **filename** (*str*) – filename is the of the file to parse.
Returns The content of the file as dictionary.
Return type *Model*

load_parameters

YmlProjectIo.**load_parameters**(*file_name: str*) →
glotaran.parameter.parameter_group.ParameterGroup
Create a ParameterGroup instance from the specs defined in a file. :param file_name: File containing the parameter specs. :type file_name: str
Returns ParameterGroup instance created from the file.
Return type *ParameterGroup*

load_result

YmlProjectIo.**load_result**(*result_path: str*) → *glotaran.project.result.Result*

Create a Result instance from the specs defined in a file.

Parameters **result_path** (*str | PathLike[str]*) – Path containing the result data.

Returns Result instance created from the saved format.

Return type *Result*

load_scheme

YmlProjectIo.**load_scheme**(*file_name: str*) → *glotaran.project.scheme.Scheme*

Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns

- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model

YmlProjectIo.**save_model**(*model:* glotaran.model.model.Model, *file_name: str*)

Save a Model instance to a spec file. :param model: Model instance to save to specs file. :type model: Model :param file_name: File to write the model specs to. :type file_name: str

save_parameters

YmlProjectIo.**save_parameters**(*parameters: ParameterGroup*, *file_name: str*)

Save a ParameterGroup instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **parameters** (ParameterGroup) – ParameterGroup instance to save to specs file.
- **file_name** (*str*) – File to write the parameter specs to.

save_result

YmlProjectIo.**save_result**(*result:* glotaran.project.result.Result, *result_path: str*)

Write a Result instance to a spec file.

Parameters

- **result** (Result) – Result instance to write.
- **result_path** (*str | PathLike[str]*) – Path to write the result data to.

save_scheme

YmlProjectIo.**save_scheme**(*scheme:* glotaran.project.scheme.Scheme, *file_name: str*)
Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

Parameters
- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

Methods Documentation

load_model(*file_name: str*) → *glotaran.model.model.Model*
parse_yaml_file reads the given file and parses its content as YML.

Parameters **filename** (*str*) – filename is the of the file to parse.
Returns The content of the file as dictionary.
Return type *Model*

load_parameters(*file_name: str*) → *glotaran.parameter.parameter_group.ParameterGroup*
Create a ParameterGroup instance from the specs defined in a file. :param file_name: File containing the parameter specs. :type file_name: str

Returns ParameterGroup instance created from the file.
Return type *ParameterGroup*

load_result(*result_path: str*) → *glotaran.project.result.Result*
Create a Result instance from the specs defined in a file.

Parameters **result_path** (*str | PathLike[str]*) – Path containing the result data.
Returns Result instance created from the saved format.
Return type *Result*

load_scheme(*file_name: str*) → *glotaran.project.scheme.Scheme*
Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.
Returns
- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model(*model:* glotaran.model.model.Model, *file_name: str*)
Save a Model instance to a spec file. :param model: Model instance to save to specs file. :type model: Model :param file_name: File to write the model specs to. :type file_name: str

save_parameters(*parameters: ParameterGroup*, *file_name: str*)
Save a ParameterGroup instance to a spec file (**NOT IMPLEMENTED**).

Parameters
- **parameters** (ParameterGroup) – ParameterGroup instance to save to specs file.
- **file_name** (*str*) – File to write the parameter specs to.

save_result(*result:* glotaran.project.result.Result, *result_path: str*)
Write a Result instance to a spec file.

Parameters
- **result** (Result) – Result instance to write.
- **result_path** (*str | PathLike[str]*) – Path to write the result data to.

save_scheme(*scheme:* glotaran.project.scheme.Scheme, *file_name: str*)
Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

Parameters

- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

megacomplexes

Modules

glotaran.builtin.megacomplexes.baseline
glotaran.builtin.megacomplexes.coherent_artifact
glotaran.builtin.megacomplexes.damped_oscillation
glotaran.builtin.megacomplexes.decay
glotaran.builtin.megacomplexes.spectral

baseline

Modules

glotaran.builtin.megacomplexes.baseline.baseline_megacomplex

baseline_megacomplex

Classes

Summary

BaselineMegacomplex

BaselineMegacomplex

class glotaran.builtin.megacomplexes.baseline.baseline_megacomplex.**BaselineMegacomplex**

Bases: glotaran.model.megacomplex.Megacomplex

Attributes Summary

dimension	ModelProperty is an extension of the property decorator.
label	ModelProperty is an extension of the property decorator.
name	
type	ModelProperty is an extension of the property decorator.

dimension

BaselineMegacomplex.**dimension**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

label

BaselineMegacomplex.**label**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

name

BaselineMegacomplex.**name = 'baseline'**

type

BaselineMegacomplex.**type**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

as_dict
calculate_matrix
fill
finalize_data
from_dict

continues on next page

Table 58 – continued from previous page

glotaran_dataset_model_items
glotaran_dataset_properties
glotaran_model_items
glotaran_unique
index_dependent
markdown
validate

as_dict

BaselineMegacomplex.**as_dict**() → dict

calculate_matrix

BaselineMegacomplex.**calculate_matrix**(*dataset_model: DatasetModel*, *indices: dict[str, int]*, ***kwargs*)

fill

BaselineMegacomplex.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

finalize_data

BaselineMegacomplex.**finalize_data**(*dataset_model:* glotaran.model.dataset_model.DatasetModel, *dataset: xarray.core.dataset.Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

from_dict

classmethod BaselineMegacomplex.**from_dict**(*values: dict*) → cls

glotaran_dataset_model_items

classmethod BaselineMegacomplex.**glotaran_dataset_model_items**() → str

glotaran_dataset_properties

classmethod BaselineMegacomplex.**glotaran_dataset_properties**() → str

glotaran_model_items

classmethod BaselineMegacomplex.**glotaran_model_items**() → str

glotaran_unique

classmethod BaselineMegacomplex.**glotaran_unique**() → bool

index_dependent

BaselineMegacomplex.**index_dependent**(*dataset_model:* glotaran.model.dataset_model.DatasetModel) → bool

markdown

BaselineMegacomplex.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

validate

BaselineMegacomplex.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

as_dict() → dict

calculate_matrix(*dataset_model: DatasetModel*, *indices: dict[str, int]*, ***kwargs*)

property dimension: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

fill(*model: Model*, *parameters: ParameterGroup*) → cls

finalize_data(*dataset_model:* glotaran.model.dataset_model.DatasetModel, *dataset: xarray.core.dataset.Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

classmethod from_dict(*values: dict*) → cls

classmethod glotaran_dataset_model_items() → str

classmethod glotaran_dataset_properties() → str

classmethod glotaran_model_items() → str

classmethod glotaran_unique() → bool

index_dependent(*dataset_model:* glotaran.model.dataset_model.DatasetModel) → bool

property label: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

name = 'baseline'

property type: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

coherent_artifact

Modules

glotaran.builtin.megacomplexes.coherent_artifact.coherent_artifact_megacomplex	This package contains the kinetic megacomplex item.

coherent_artifact_megacomplex

This package contains the kinetic megacomplex item.

Classes

Summary

CoherentArtifactMegacomplex

CoherentArtifactMegacomplex

class glotaran.builtin.megacomplexes.coherent_artifact.coherent_artifact_megacomplex.**CoherentArtifactMegacomplex**

Bases: glotaran.model.megacomplex.Megacomplex

Attributes Summary

dimension	ModelProperty is an extension of the property decorator.
label	ModelProperty is an extension of the property decorator.
name	
order	ModelProperty is an extension of the property decorator.
type	ModelProperty is an extension of the property decorator.
width	ModelProperty is an extension of the property decorator.

dimension

CoherentArtifactMegacomplex.**dimension**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

label

CoherentArtifactMegacomplex.**label**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

name

CoherentArtifactMegacomplex.**name = 'coherent-artifact'**

order

CoherentArtifactMegacomplex.**order**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

type

CoherentArtifactMegacomplex.**type**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

width

CoherentArtifactMegacomplex.**width**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

as_dict
calculate_matrix
compartments
fill

continues on next page

Table 62 – continued from previous page

finalize_data
from_dict
glotaran_dataset_model_items
glotaran_dataset_properties
glotaran_model_items
glotaran_unique
index_dependent
markdown
validate

as_dict

CoherentArtifactMegacomplex.**as_dict**() → dict

calculate_matrix

CoherentArtifactMegacomplex.**calculate_matrix**(*dataset_model: DatasetModel*, *indices: dict[str, int]*, ***kwargs*)

compartments

CoherentArtifactMegacomplex.**compartments**()

fill

CoherentArtifactMegacomplex.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

finalize_data

CoherentArtifactMegacomplex.**finalize_data**(*dataset_model:* glotaran.model.dataset_model.DatasetModel, *dataset: xarray.core.dataset.Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

from_dict

classmethod CoherentArtifactMegacomplex.**from_dict**(*values: dict*) → cls

glotaran_dataset_model_items

classmethod CoherentArtifactMegacomplex.**glotaran_dataset_model_items**() → str

glotaran_dataset_properties

classmethod CoherentArtifactMegacomplex.**glotaran_dataset_properties**() → str

glotaran_model_items

classmethod CoherentArtifactMegacomplex.**glotaran_model_items**() → str

glotaran_unique

classmethod CoherentArtifactMegacomplex.**glotaran_unique**() → bool

index_dependent

CoherentArtifactMegacomplex.**index_dependent**(*dataset_model:* glotaran.model.dataset_model.DatasetModel) → bool

markdown

CoherentArtifactMegacomplex.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

validate

CoherentArtifactMegacomplex.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

as_dict() → dict

calculate_matrix(*dataset_model: DatasetModel*, *indices: dict[str, int]*, ***kwargs*)

compartments()

property dimension: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

fill(*model: Model*, *parameters: ParameterGroup*) → cls

finalize_data(*dataset_model: glotaran.model.dataset_model.DatasetModel*, *dataset: xarray.core.dataset.Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

classmethod from_dict(*values: dict*) → cls

classmethod glotaran_dataset_model_items() → str

classmethod glotaran_dataset_properties() → str

classmethod glotaran_model_items() → str

classmethod glotaran_unique() → bool

index_dependent(*dataset_model: glotaran.model.dataset_model.DatasetModel*) → bool

property label: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

name = 'coherent-artifact'

property order: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property type: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

property width: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

damped_oscillation

Modules

glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation_megacomplex

damped_oscillation_megacomplex

Functions

Summary

calculate_damped_oscillation_matrix_gaussian_irf	Calculate the damped oscillation matrix taking into account a gaussian irf
calculate_damped_oscillation_matrix_no_irf	

calculate_damped_oscillation_matrix_gaussian_irf

glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation_megacomplex.**calculate_damped_o**

Calculate the damped oscillation matrix taking into account a gaussian irf

Parameters

- **frequencies** (*np.ndarray*) – an array of frequencies in THz, one per oscillation
- **rates** (*np.ndarray*) – an array of rates, one per oscillation
- **model_axis** (*np.ndarray*) – the model axis (time)
- **center** (*float*) – the center of the gaussian IRF
- **width** (*float*) – the width () parameter of the the IRF
- **shift** (*float*) – a shift parameter per item on the global axis
- **scale** (*float*) – the scale parameter to scale the matrix by

Returns An array of the real and imaginary part of the oscillation matrix, the shape being (len(model_axis), 2*len(frequencies)), with the first half of the second dimension representing the real part, and the other the imagine part of the oscillation

Return type np.ndarray

calculate_damped_oscillation_matrix_no_irf

glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation_megacomplex.**calculate_damped_o**

Classes

Summary

DampedOscillationMegacomplex

DampedOscillationMegacomplex

class glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation_megacomplex.**DampedOscillationMegacomplex**

Bases: glotaran.model.megacomplex.Megacomplex

Attributes Summary

dimension	ModelProperty is an extension of the property decorator.
frequencies	ModelProperty is an extension of the property decorator.
label	ModelProperty is an extension of the property decorator.
labels	ModelProperty is an extension of the property decorator.
name	
rates	ModelProperty is an extension of the property decorator.
type	ModelProperty is an extension of the property decorator.

dimension

DampedOscillationMegacomplex.**dimension**

ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

frequencies

DampedOscillationMegacomplex.**frequencies**

ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

label

DampedOscillationMegacomplex.**label**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

labels

DampedOscillationMegacomplex.**labels**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

name

DampedOscillationMegacomplex.**name = 'damped-oscillation'**

rates

DampedOscillationMegacomplex.**rates**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

type

DampedOscillationMegacomplex.**type**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

as_dict
calculate_matrix
ensure_oscillation_parameter
fill
finalize_data
from_dict
glotaran_dataset_model_items

continues on next page

Table 67 – continued from previous page

glotaran_dataset_properties
glotaran_model_items
glotaran_unique
index_dependent
markdown
validate

as_dict

DampedOscillationMegacomplex.**as_dict**() → dict

calculate_matrix

DampedOscillationMegacomplex.**calculate_matrix**(*dataset_model: DatasetModel*, *indices: dict[str, int]*, ***kwargs*)

ensure_oscillation_parameter

DampedOscillationMegacomplex.**ensure_oscillation_parameter**(*model: Model*) → list[str]

fill

DampedOscillationMegacomplex.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

finalize_data

DampedOscillationMegacomplex.**finalize_data**(*dataset_model: glotaran.model.dataset_model.DatasetModel*, *dataset: xarray.core.dataset.Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

from_dict

classmethod DampedOscillationMegacomplex.**from_dict**(*values: dict*) → cls

glotaran_dataset_model_items

classmethod DampedOscillationMegacomplex.**glotaran_dataset_model_items**() → str

glotaran_dataset_properties

classmethod DampedOscillationMegacomplex.**glotaran_dataset_properties**() → str

glotaran_model_items

classmethod DampedOscillationMegacomplex.**glotaran_model_items**() → str

glotaran_unique

classmethod DampedOscillationMegacomplex.**glotaran_unique**() → bool

index_dependent

DampedOscillationMegacomplex.**index_dependent**(*dataset_model:* glotaran.model.dataset_model.DatasetModel) → bool

markdown

DampedOscillationMegacomplex.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

validate

DampedOscillationMegacomplex.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

as_dict() → dict

calculate_matrix(*dataset_model: DatasetModel*, *indices: dict[str, int]*, ***kwargs*)

property dimension: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

ensure_oscillation_parameter(*model: Model*) → list[str]

fill(*model: Model*, *parameters: ParameterGroup*) → cls

finalize_data(*dataset_model:* glotaran.model.dataset_model.DatasetModel, *dataset: xarray.core.dataset.Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

property frequencies: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

classmethod from_dict(*values: dict*) → cls

classmethod glotaran_dataset_model_items() → str

classmethod glotaran_dataset_properties() → str

classmethod glotaran_model_items() → str

classmethod glotaran_unique() → bool

index_dependent(*dataset_model:* glotaran.model.dataset_model.DatasetModel) → bool

property label: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property labels: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

name = 'damped-oscillation'

property rates: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property type: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

decay

Modules

glotaran.builtin.megacomplexes.decay.decay_megacomplex	This package contains the decay megacomplex item.
glotaran.builtin.megacomplexes.decay.initial_concentration	This package contains the initial concentration item.
glotaran.builtin.megacomplexes.decay.irf	This package contains irf items.
glotaran.builtin.megacomplexes.decay.k_matrix	K-Matrix
glotaran.builtin.megacomplexes.decay.util	

decay_megacomplex

This package contains the decay megacomplex item.

Classes

Summary

DecayMegacomplex	A Megacomplex with one or more K-Matrices.

DecayMegacomplex

class glotaran.builtin.megacomplexes.decay.decay_megacomplex.**DecayMegacomplex**
Bases: glotaran.model.megacomplex.Megacomplex

A Megacomplex with one or more K-Matrices.

Attributes Summary

dimension	ModelProperty is an extension of the property decorator.
involved_compartments	
k_matrix	ModelProperty is an extension of the property decorator.
label	ModelProperty is an extension of the property decorator.
name	
type	ModelProperty is an extension of the property decorator.

dimension

DecayMegacomplex.**dimension**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

involved_compartments

DecayMegacomplex.**involved_compartments**

k_matrix

DecayMegacomplex.**k_matrix**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

label

DecayMegacomplex.**label**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

name

DecayMegacomplex.**name = 'decay'**

type

DecayMegacomplex.**type**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

as_dict
calculate_matrix
fill
finalize_data
from_dict
full_k_matrix
glotaran_dataset_model_items
glotaran_dataset_properties
glotaran_model_items
glotaran_unique
has_k_matrix

continues on next page

Table 71 – continued from previous page

index_dependent
markdown
validate

as_dict

DecayMegacomplex.**as_dict**() → dict

calculate_matrix

DecayMegacomplex.**calculate_matrix**(*dataset_model: DatasetModel*, *indices: dict[str, int]*, ***kwargs*)

fill

DecayMegacomplex.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

finalize_data

DecayMegacomplex.**finalize_data**(*dataset_model:* glotaran.model.dataset_model.DatasetModel, *dataset: xarray.core.dataset.Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

from_dict

classmethod DecayMegacomplex.**from_dict**(*values: dict*) → cls

full_k_matrix

DecayMegacomplex.**full_k_matrix**(*model=None*)

glotaran_dataset_model_items

classmethod DecayMegacomplex.**glotaran_dataset_model_items**() → str

glotaran_dataset_properties

classmethod DecayMegacomplex.**glotaran_dataset_properties**() → str

glotaran_model_items

classmethod DecayMegacomplex.**glotaran_model_items**() → str

glotaran_unique

classmethod DecayMegacomplex.**glotaran_unique**() → bool

has_k_matrix

DecayMegacomplex.**has_k_matrix**() → bool

index_dependent

DecayMegacomplex.**index_dependent**(*dataset_model:* glotaran.model.dataset_model.DatasetModel) → bool

markdown

DecayMegacomplex.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

validate

DecayMegacomplex.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

as_dict() → dict

calculate_matrix(*dataset_model: DatasetModel*, *indices: dict[str, int]*, ***kwargs*)

property dimension: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

fill(*model: Model*, *parameters: ParameterGroup*) → cls

finalize_data(*dataset_model:* glotaran.model.dataset_model.DatasetModel, *dataset: xarray.core.dataset.Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

classmethod from_dict(*values: dict*) → cls

full_k_matrix(*model=None*)

classmethod glotaran_dataset_model_items() → str

classmethod glotaran_dataset_properties() → str

classmethod glotaran_model_items() → str

classmethod glotaran_unique() → bool

has_k_matrix() → bool

index_dependent(*dataset_model:* glotaran.model.dataset_model.DatasetModel) → bool

property involved_compartments

property k_matrix: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property label: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

name = 'decay'

property type: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

initial_concentration

This package contains the initial concentration item.

Classes

Summary

InitialConcentration	An initial concentration describes the population of the compartments at the beginning of an experiment.

InitialConcentration

class glotaran.builtin.megacomplexes.decay.initial_concentration.**InitialConcentration**
Bases: object

An initial concentration describes the population of the compartments at the beginning of an experiment.

Attributes Summary

compartments	ModelProperty is an extension of the property decorator.
exclude_from_normalize	ModelProperty is an extension of the property decorator.
label	ModelProperty is an extension of the property decorator.
parameters	ModelProperty is an extension of the property decorator.

compartments

InitialConcentration.**compartments**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

exclude_from_normalize

InitialConcentration.**exclude_from_normalize**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

label

InitialConcentration.**label**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

parameters

InitialConcentration.**parameters**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

as_dict
fill
from_dict
markdown
normalized
validate

as_dict

InitialConcentration.**as_dict**() → dict

fill

InitialConcentration.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

from_dict

classmethod InitialConcentration.**from_dict**(*values: dict*) → cls

markdown

InitialConcentration.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

normalized

InitialConcentration.**normalized**() → *glotaran.builtin.megacomplexes.decay.initial_concentration.InitialConcentration*

validate

InitialConcentration.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

as_dict() → dict

property compartments: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property exclude_from_normalize: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

fill(*model: Model*, *parameters: ParameterGroup*) → cls

classmethod from_dict(*values: dict*) → cls

property label: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

normalized() → *glotaran.builtin.megacomplexes.decay.initial_concentration.InitialConcentration*

property parameters: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

irf

This package contains irf items.

Classes

Summary

Irf	Represents an IRF.
IrfGaussian	
IrfMeasured	A measured IRF.
IrfMultiGaussian	Represents a gaussian IRF.
IrfSpectralGaussian	
IrfSpectralMultiGaussian	Represents a gaussian IRF.

Irf

class glotaran.builtin.megacomplexes.decay.irf.**Irf**
Bases: object

Represents an IRF.

Methods Summary

add_type	
get_default_type	

add_type

classmethod Irf.**add_type**(*type_name: str*, *attribute_type: type*)

get_default_type

classmethod Irf.**get_default_type**() → str

Methods Documentation

classmethod **add_type**(*type_name: str*, *attribute_type: type*)

classmethod **get_default_type**() → str

IrfGaussian

class glotaran.builtin.megacomplexes.decay.irf.**IrfGaussian**
Bases: *glotaran.builtin.megacomplexes.decay.irf.IrfMultiGaussian*

Attributes Summary

backsweep	ModelProperty is an extension of the property decorator.
backsweep_period	ModelProperty is an extension of the property decorator.
center	ModelProperty is an extension of the property decorator.
label	ModelProperty is an extension of the property decorator.
normalize	ModelProperty is an extension of the property decorator.
scale	ModelProperty is an extension of the property decorator.
shift	ModelProperty is an extension of the property decorator.

continues on next page

Table 77 – continued from previous page

type	ModelProperty is an extension of the property decorator.
width	ModelProperty is an extension of the property decorator.

backsweep

IrfGaussian.**backsweep**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

backsweep_period

IrfGaussian.**backsweep_period**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

center

IrfGaussian.**center**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

label

IrfGaussian.**label**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

normalize

IrfGaussian.**normalize**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

scale

IrfGaussian.**scale**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

shift

IrfGaussian.**shift**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

type

IrfGaussian.**type**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

width

IrfGaussian.**width**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

as_dict	
calculate	
fill	
from_dict	
is_index_dependent	
markdown	
parameter	Returns the properties of the irf with shift applied.
validate	

as_dict

IrfGaussian.**as_dict**() → dict

calculate

IrfGaussian.**calculate**(*index: int*, *global_axis: numpy.ndarray*, *model_axis: numpy.ndarray*) → numpy.ndarray

fill

IrfGaussian.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

from_dict

classmethod IrfGaussian.**from_dict**(*values: dict*) → cls

is_index_dependent

IrfGaussian.**is_index_dependent**()

markdown

IrfGaussian.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

parameter

IrfGaussian.**parameter**(*global_index: int*, *global_axis: numpy.ndarray*) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, float, bool, float]

Returns the properties of the irf with shift applied.

validate

IrfGaussian.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

as_dict() → dict

property backsweep: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property backsweep_period: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

calculate(*index: int*, *global_axis: numpy.ndarray*, *model_axis: numpy.ndarray*) → numpy.ndarray

property center: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

fill(*model: Model*, *parameters: ParameterGroup*) → cls

classmethod from_dict(*values: dict*) → cls

is_index_dependent()

property label: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

property normalize: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

parameter(*global_index: int*, *global_axis: numpy.ndarray*) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, float, bool, float]
Returns the properties of the irf with shift applied.

property scale: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property shift: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property type: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

property width: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

IrfMeasured

class glotaran.builtin.megacomplexes.decay.irf.**IrfMeasured**
Bases: object

A measured IRF. The data must be supplied by the dataset.

Attributes Summary

label	ModelProperty is an extension of the property decorator.
type	ModelProperty is an extension of the property decorator.

label

IrfMeasured.**label**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

type

IrfMeasured.**type**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

as_dict
fill
from_dict
markdown
validate

as_dict

IrfMeasured.**as_dict**() → dict

fill

IrfMeasured.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

from_dict

classmethod IrfMeasured.**from_dict**(*values: dict*) → cls

markdown

IrfMeasured.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

validate

IrfMeasured.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

as_dict() → dict

fill(*model: Model*, *parameters: ParameterGroup*) → cls

classmethod from_dict(*values: dict*) → cls

property label: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

property type: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

IrfMultiGaussian

class glotaran.builtin.megacomplexes.decay.irf.**IrfMultiGaussian**
Bases: object

Represents a gaussian IRF.

One width and one center is a single gauss.

One center and multiple widths is a multiple gaussian.

Multiple center and multiple widths is Double-, Triple- , etc. Gaussian.

Parameters

- **label** – label of the irf
- **center** – one or more center of the irf as parameter indices
- **width** – one or more widths of the gaussian as parameter index
- **center_dispersion_coefficients** – polynomial coefficients for the dispersion of the center as list of parameter indices. None for no dispersion.
- **width_dispersion_coefficients** – polynomial coefficients for the dispersion of the width as parameter indices. None for no dispersion.

Attributes Summary

backsweep	ModelProperty is an extension of the property decorator.
backsweep_period	ModelProperty is an extension of the property decorator.
center	ModelProperty is an extension of the property decorator.
label	ModelProperty is an extension of the property decorator.
normalize	ModelProperty is an extension of the property decorator.
scale	ModelProperty is an extension of the property decorator.
shift	ModelProperty is an extension of the property decorator.
type	ModelProperty is an extension of the property decorator.
width	ModelProperty is an extension of the property decorator.

backsweep

IrfMultiGaussian.**backsweep**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

backsweep_period

IrfMultiGaussian.**backsweep_period**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

center

IrfMultiGaussian.**center**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

label

IrfMultiGaussian.**label**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

normalize

IrfMultiGaussian.**normalize**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

scale

IrfMultiGaussian.**scale**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

shift

IrfMultiGaussian.**shift**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

type

IrfMultiGaussian.**type**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

width

IrfMultiGaussian.**width**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

as_dict	
calculate	
fill	
from_dict	
is_index_dependent	
markdown	
parameter	Returns the properties of the irf with shift applied.
validate	

as_dict

IrfMultiGaussian.**as_dict**() → dict

calculate

IrfMultiGaussian.**calculate**(*index: int*, *global_axis: numpy.ndarray*, *model_axis: numpy.ndarray*) → numpy.ndarray

fill

IrfMultiGaussian.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

from_dict

classmethod IrfMultiGaussian.**from_dict**(*values: dict*) → cls

is_index_dependent

IrfMultiGaussian.**is_index_dependent**()

markdown

IrfMultiGaussian.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

parameter

IrfMultiGaussian.**parameter**(*global_index: int*, *global_axis: numpy.ndarray*) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, float, bool, float]

Returns the properties of the irf with shift applied.

validate

IrfMultiGaussian.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

as_dict() → dict

property backsweep: model_property.glotaran_property_type

ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property backsweep_period: model_property.glotaran_property_type

ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

calculate(*index: int*, *global_axis: numpy.ndarray*, *model_axis: numpy.ndarray*) → numpy.ndarray

property center: model_property.glotaran_property_type

ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

fill(*model: Model*, *parameters: ParameterGroup*) → cls

classmethod from_dict(*values: dict*) → cls

is_index_dependent()

property label: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

property normalize: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

parameter(*global_index: int*, *global_axis: numpy.ndarray*) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, float, bool, float]
Returns the properties of the irf with shift applied.

property scale: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property shift: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property type: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

property width: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

IrfSpectralGaussian

class glotaran.builtin.megacomplexes.decay.irf.**IrfSpectralGaussian**
Bases: *glotaran.builtin.megacomplexes.decay.irf.IrfSpectralMultiGaussian*

Attributes Summary

backsweep	ModelProperty is an extension of the property decorator.
backsweep_period	ModelProperty is an extension of the property decorator.
center	ModelProperty is an extension of the property decorator.

continues on next page

Table 83 – continued from previous page

center_dispersion_coefficients	ModelProperty is an extension of the property decorator.
dispersion_center	ModelProperty is an extension of the property decorator.
label	ModelProperty is an extension of the property decorator.
model_dispersion_with_wavenumber	ModelProperty is an extension of the property decorator.
normalize	ModelProperty is an extension of the property decorator.
scale	ModelProperty is an extension of the property decorator.
shift	ModelProperty is an extension of the property decorator.
type	ModelProperty is an extension of the property decorator.
width	ModelProperty is an extension of the property decorator.
width_dispersion_coefficients	ModelProperty is an extension of the property decorator.

backsweep

IrfSpectralGaussian.**backsweep**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

backsweep_period

IrfSpectralGaussian.**backsweep_period**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

center

IrfSpectralGaussian.**center**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

center_dispersion_coefficients

IrfSpectralGaussian.**center_dispersion_coefficients**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

dispersion_center

IrfSpectralGaussian.**dispersion_center**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

label

IrfSpectralGaussian.**label**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

model_dispersion_with_wavenumber

IrfSpectralGaussian.**model_dispersion_with_wavenumber**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

normalize

IrfSpectralGaussian.**normalize**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

scale

IrfSpectralGaussian.**scale**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

shift

IrfSpectralGaussian.**shift**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

type

IrfSpectralGaussian.**type**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

width

IrfSpectralGaussian.**width**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

width_dispersion_coefficients

IrfSpectralGaussian.**width_dispersion_coefficients**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

as_dict	
calculate	
calculate_dispersion	
fill	
from_dict	
is_index_dependent	
markdown	
parameter	Returns the properties of the irf with shift and dispersion applied.
validate	

as_dict

IrfSpectralGaussian.**as_dict**() → dict

calculate

IrfSpectralGaussian.**calculate**(*index: int*, *global_axis: numpy.ndarray*, *model_axis: numpy.ndarray*) → numpy.ndarray

calculate_dispersion

IrfSpectralGaussian.**calculate_dispersion**(*axis*)

fill

IrfSpectralGaussian.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

from_dict

classmethod IrfSpectralGaussian.**from_dict**(*values: dict*) → cls

is_index_dependent

IrfSpectralGaussian.**is_index_dependent**()

markdown

IrfSpectralGaussian.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

parameter

IrfSpectralGaussian.**parameter**(*global_index: int*, *global_axis: numpy.ndarray*)
Returns the properties of the irf with shift and dispersion applied.

validate

IrfSpectralGaussian.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

as_dict() → dict

property backsweep: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property backsweep_period: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

calculate(*index: int*, *global_axis: numpy.ndarray*, *model_axis: numpy.ndarray*) → numpy.ndarray

calculate_dispersion(*axis*)

property center: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property center_dispersion_coefficients: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property dispersion_center: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

fill(*model: Model*, *parameters: ParameterGroup*) → cls

classmethod from_dict(*values: dict*) → cls

is_index_dependent()

property label: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

property model_dispersion_with_wavenumber: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property normalize: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

parameter(*global_index: int*, *global_axis: numpy.ndarray*)
Returns the properties of the irf with shift and dispersion applied.

property scale: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property shift: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property type: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

property width: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property width_dispersion_coefficients: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

IrfSpectralMultiGaussian

class glotaran.builtin.megacomplexes.decay.irf.**IrfSpectralMultiGaussian**
Bases: *glotaran.builtin.megacomplexes.decay.irf.IrfMultiGaussian*

Represents a gaussian IRF.

One width and one center is a single gauss.

One center and multiple widths is a multiple gaussian.

Multiple center and multiple widths is Double-, Triple- , etc. Gaussian.

Parameters

- **label** – label of the irf
- **center** – one or more center of the irf as parameter indices
- **width** – one or more widths of the gaussian as parameter index
- **center_dispersion_coefficients** – list of parameters with polynomial coefficients describing the dispersion of the irf center location. None for no dispersion.
- **width_dispersion_coefficients** – list of parameters with polynomial coefficients describing the dispersion of the width of the irf. None for no dispersion.

Attributes Summary

backsweep	ModelProperty is an extension of the property decorator.
backsweep_period	ModelProperty is an extension of the property decorator.
center	ModelProperty is an extension of the property decorator.
center_dispersion_coefficients	ModelProperty is an extension of the property decorator.
dispersion_center	ModelProperty is an extension of the property decorator.
label	ModelProperty is an extension of the property decorator.
model_dispersion_with_wavenumber	ModelProperty is an extension of the property decorator.
normalize	ModelProperty is an extension of the property decorator.
scale	ModelProperty is an extension of the property decorator.
shift	ModelProperty is an extension of the property decorator.
type	ModelProperty is an extension of the property decorator.
width	ModelProperty is an extension of the property decorator.
width_dispersion_coefficients	ModelProperty is an extension of the property decorator.

backsweep

IrfSpectralMultiGaussian.**backsweep**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

backsweep_period

IrfSpectralMultiGaussian.**backsweep_period**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

center

IrfSpectralMultiGaussian.**center**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

center_dispersion_coefficients

IrfSpectralMultiGaussian.**center_dispersion_coefficients**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

dispersion_center

IrfSpectralMultiGaussian.**dispersion_center**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

label

IrfSpectralMultiGaussian.**label**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

model_dispersion_with_wavenumber

IrfSpectralMultiGaussian.**model_dispersion_with_wavenumber**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

normalize

IrfSpectralMultiGaussian.**normalize**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

scale

IrfSpectralMultiGaussian.**scale**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

shift

IrfSpectralMultiGaussian.**shift**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

type

IrfSpectralMultiGaussian.**type**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

width

IrfSpectralMultiGaussian.**width**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

width_dispersion_coefficients

IrfSpectralMultiGaussian.**width_dispersion_coefficients**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

as_dict	
calculate	
calculate_dispersion	
fill	
from_dict	
is_index_dependent	
markdown	
parameter	Returns the properties of the irf with shift and dispersion applied.
validate	

as_dict

IrfSpectralMultiGaussian.**as_dict**() → dict

calculate

IrfSpectralMultiGaussian.**calculate**(*index: int*, *global_axis: numpy.ndarray*, *model_axis: numpy.ndarray*) → numpy.ndarray

calculate_dispersion

IrfSpectralMultiGaussian.**calculate_dispersion**(*axis*)

fill

IrfSpectralMultiGaussian.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

from_dict

classmethod IrfSpectralMultiGaussian.**from_dict**(*values: dict*) → cls

is_index_dependent

IrfSpectralMultiGaussian.**is_index_dependent**()

markdown

IrfSpectralMultiGaussian.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

parameter

IrfSpectralMultiGaussian.**parameter**(*global_index: int*, *global_axis: numpy.ndarray*)
Returns the properties of the irf with shift and dispersion applied.

validate

IrfSpectralMultiGaussian.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

as_dict() → dict

property backsweep: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property backsweep_period: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

calculate(*index: int*, *global_axis: numpy.ndarray*, *model_axis: numpy.ndarray*) → numpy.ndarray

calculate_dispersion(*axis*)

property center: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property center_dispersion_coefficients: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property dispersion_center: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

fill(*model: Model*, *parameters: ParameterGroup*) → cls

classmethod from_dict(*values: dict*) → cls

is_index_dependent()

property label: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

property model_dispersion_with_wavenumber: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property normalize: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

parameter(*global_index: int*, *global_axis: numpy.ndarray*)
Returns the properties of the irf with shift and dispersion applied.

property scale: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property shift: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property type: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

property width: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property width_dispersion_coefficients: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

k_matrix

K-Matrix

Classes

Summary

KMatrix	A K-Matrix represents a first order differental system.

KMatrix

class glotaran.builtin.megacomplexes.decay.k_matrix.**KMatrix**
Bases: object

A K-Matrix represents a first order differental system.

Attributes Summary

label	ModelProperty is an extension of the property decorator.
matrix	ModelProperty is an extension of the property decorator.

label

KMatrix.**label**

ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

matrix

KMatrix.**matrix**

ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

a_matrix	The resulting A matrix of the KMatrix.
a_matrix_as_markdown	Returns the A Matrix as markdown formatted table.
a_matrix_non_unibranch	The resulting A matrix of the KMatrix for a non-unibranched model.
a_matrix_unibranch	The resulting A matrix of the KMatrix for an unibranched model.
as_dict	
combine	Creates a combined matrix.
eigen	Returns the eigenvalues and eigenvectors of the k matrix.
empty	Creates an empty K-Matrix.
fill	
from_dict	
full	The full representation of the KMatrix as numpy array.
involved_compartments	A list of all compartments in the Matrix.
is_unibranched	Returns true in the KMatrix represents an unibranched model.
markdown	
matrix_as_markdown	Returns the KMatrix as markdown formatted table.
rates	The resulting rates of the matrix.
reduced	The reduced representation of the KMatrix as numpy array.
validate	

a_matrix

KMatrix.**a_matrix**(*initial_concentration:* glotaran.builtin.megacomplexes.decay.initial_concentration.InitialConcentration) → numpy.ndarray

The resulting A matrix of the KMatrix.

Parameters **initial_concentration** – The initial concentration.

a_matrix_as_markdown

KMatrix.**a_matrix_as_markdown**(*initial_concentration:* glotaran.builtin.megacomplexes.decay.initial_concentration.InitialConcentration) → *glotaran.utils.ipython.MarkdownStr*

Returns the A Matrix as markdown formatted table.

Parameters **initial_concentration** – The initial concentration.

a_matrix_non_unibranch

KMatrix.**a_matrix_non_unibranch**(*initial_concentration:* glotaran.builtin.megacomplexes.decay.initial_concentration.InitialConcentration) → numpy.ndarray

The resulting A matrix of the KMatrix for a non-unibranched model.

Parameters **initial_concentration** – The initial concentration.

a_matrix_unibranch

KMatrix.**a_matrix_unibranch**(*initial_concentration:* glotaran.builtin.megacomplexes.decay.initial_concentration.InitialConcentration) → numpy.ndarray

The resulting A matrix of the KMatrix for an unibranched model.

Parameters **initial_concentration** – The initial concentration.

as_dict

KMatrix.**as_dict**() → dict

combine

KMatrix.**combine**(*k_matrix:* glotaran.builtin.megacomplexes.decay.k_matrix.KMatrix) → *glotaran.builtin.megacomplexes.decay.k_matrix.KMatrix*

Creates a combined matrix.

When combining k-matrices km1 and km2 (km1.combine(km2)), entries in km1 will be overwritten by corresponding entries in km2.

Parameters **k_matrix** – KMatrix to combine with.

Returns The combined KMatrix.

Return type combined

eigen

KMatrix.**eigen**(*compartments: list[str]*) → tuple[np.ndarray, np.ndarray]

Returns the eigenvalues and eigenvectors of the k matrix.

Parameters **compartments** – The compartment order.

empty

classmethod KMatrix.**empty**(*label: str*, *compartments: list[str]*) → KMatrix

Creates an empty K-Matrix. Useful for combining.

Parameters

- **label** – Label of the K-Matrix
- **compartments** – A list of all compartments in the model.

fill

KMatrix.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

from_dict

classmethod KMatrix.**from_dict**(*values: dict*) → cls

full

KMatrix.**full**(*compartments: list[str]*) → np.ndarray

The full representation of the KMatrix as numpy array.

Parameters **compartments** – The compartment order.

involved_compartments

KMatrix.**involved_compartments**() → list[str]

A list of all compartments in the Matrix.

is_unibranched

KMatrix.**is_unibranched**(*initial_concentration:*
glotaran.builtin.megacomplexes.decay.initial_concentration.InitialConcentration)
→ bool

Returns true in the KMatrix represents an unibranched model.

Parameters **initial_concentration** – The initial concentration.

markdown

KMatrix.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

matrix_as_markdown

KMatrix.**matrix_as_markdown**(*compartments: list[str] = None*, *fill_parameters: bool = False*) → MarkdownStr

Returns the KMatrix as markdown formatted table.

Parameters

- **compartments** – (default = None) An optional list defining the desired order of compartments.
- **fill_parameters** (*bool*) – (default = False) If true, the entries will be filled with the actual parameter values instead of labels.

rates

KMatrix.**rates**(*initial_concentration:* glotaran.builtin.megacomplexes.decay.initial_concentration.InitialConcentration) → numpy.ndarray

The resulting rates of the matrix.

Parameters **initial_concentration** – The initial concentration.

reduced

KMatrix.**reduced**(*compartments: list[str]*) → np.ndarray

The reduced representation of the KMatrix as numpy array.

Parameters **compartments** – The compartment order.

validate

KMatrix.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

a_matrix(*initial_concentration:* glotaran.builtin.megacomplexes.decay.initial_concentration.InitialConcentration) → numpy.ndarray

The resulting A matrix of the KMatrix.

Parameters **initial_concentration** – The initial concentration.

a_matrix_as_markdown(*initial_concentration:* glotaran.builtin.megacomplexes.decay.initial_concentration.InitialConcentration) → *glotaran.utils.ipython.MarkdownStr*

Returns the A Matrix as markdown formatted table.

Parameters **initial_concentration** – The initial concentration.

a_matrix_non_unibranch(*initial_concentration:* glotaran.builtin.megacomplexes.decay.initial_concentration.InitialConcentration) → numpy.ndarray

The resulting A matrix of the KMatrix for a non-unibranched model.

Parameters **initial_concentration** – The initial concentration.

a_matrix_unibranch(*initial_concentration:* glotaran.builtin.megacomplexes.decay.initial_concentration.InitialConcentration) → numpy.ndarray

The resulting A matrix of the KMatrix for an unibranched model.

Parameters **initial_concentration** – The initial concentration.

as_dict() → dict

combine(*k_matrix:* glotaran.builtin.megacomplexes.decay.k_matrix.KMatrix) → *glotaran.builtin.megacomplexes.decay.k_matrix.KMatrix*

Creates a combined matrix.

When combining k-matrices km1 and km2 (km1.combine(km2)), entries in km1 will be overwritten by corresponding entries in km2.

Parameters **k_matrix** – KMatrix to combine with.

Returns The combined KMatrix.

Return type combined

eigen(*compartments: list[str]*) → tuple[np.ndarray, np.ndarray]

Returns the eigenvalues and eigenvectors of the k matrix.

Parameters **compartments** – The compartment order.

classmethod empty(*label: str*, *compartments: list[str]*) → *KMatrix*

Creates an empty K-Matrix. Useful for combining.

Parameters

- **label** – Label of the K-Matrix
- **compartments** – A list of all compartments in the model.

fill(*model: Model*, *parameters: ParameterGroup*) → cls

classmethod from_dict(*values: dict*) → cls

full(*compartments: list[str]*) → np.ndarray

The full representation of the KMatrix as numpy array.

Parameters **compartments** – The compartment order.

involved_compartments() → list[str]

A list of all compartments in the Matrix.

is_unibranched(*initial_concentration:* glotaran.builtin.megacomplexes.decay.initial_concentration.InitialConcentration) → bool

Returns true in the KMatrix represents an unibranched model.

Parameters **initial_concentration** – The initial concentration.

property label: model_property.glotaran_property_type

ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

property matrix: model_property.glotaran_property_type

ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

matrix_as_markdown(*compartments: list[str] = None*, *fill_parameters: bool = False*) → MarkdownStr

Returns the KMatrix as markdown formatted table.

Parameters

- **compartments** – (default = None) An optional list defining the desired order of compartments.
- **fill_parameters** (*bool*) – (default = False) If true, the entries will be filled with the actual parameter values instead of labels.

rates(*initial_concentration: glotaran.builtin.megacomplexes.decay.initial_concentration.InitialConcentration*) → numpy.ndarray

The resulting rates of the matrix.

Parameters **initial_concentration** – The initial concentration.

reduced(*compartments: list[str]*) → np.ndarray

The reduced representation of the KMatrix as numpy array.

Parameters **compartments** – The compartment order.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

util

Functions

Summary

calculate_decay_matrix_gaussian_irf	Calculates a decay matrix with a gaussian irf.
calculate_decay_matrix_no_irf	
decay_matrix_implementation	
retrieve_decay_associated_data	
retrieve_irf	
retrieve_species_associated_data	

calculate_decay_matrix_gaussian_irf

glotaran.builtin.megacomplexes.decay.util.**calculate_decay_matrix_gaussian_irf**(*matrix*, *rates*, *times*, *center*, *width*, *scale*, *backsweep*, *backsweep_period*)

Calculates a decay matrix with a gaussian irf.

calculate_decay_matrix_no_irf

glotaran.builtin.megacomplexes.decay.util.**calculate_decay_matrix_no_irf**(*matrix*, *rates*, *times*)

decay_matrix_implementation

glotaran.builtin.megacomplexes.decay.util.**decay_matrix_implementation**(*matrix: numpy.ndarray*, *rates: numpy.ndarray*, *global_index: int*, *global_axis: numpy.ndarray*, *model_axis: numpy.ndarray*, *dataset_model: glotaran.model.dataset_model.Datase*

retrieve_decay_associated_data

glotaran.builtin.megacomplexes.decay.util.**retrieve_decay_associated_data**(*megacomplex: DecayMegacomplex*, *dataset_model: DatasetModel*, *dataset: xr.Dataset*, *global_dimension: str*, *name: str*, *multiple_complexes: bool*)

retrieve_irf

glotaran.builtin.megacomplexes.decay.util.**retrieve_irf**(*dataset_model: glotaran.model.dataset_model.DatasetModel*, *dataset: xarray.core.dataset.Dataset*, *global_dimension: str*)

retrieve_species_associated_data

glotaran.builtin.megacomplexes.decay.util.**retrieve_species_associated_data**(*dataset_model: glotaran.model.dataset_model.*, *dataset: xarray.core.dataset.Dataset*, *species_dimension: str*, *global_dimension: str*, *name: str*, *is_full_model: bool*, *as_global: bool*)

spectral

Modules

glotaran.builtin.megacomplexes.spectral.shape	This package contains the spectral shape item.
glotaran.builtin.megacomplexes.spectral.spectral_megacomplex	

shape

This package contains the spectral shape item.

Classes

Summary

SpectralShape	Base class for spectral shapes
SpectralShapeGaussian	A Gaussian spectral shape
SpectralShapeOne	A constant spectral shape with value 1
SpectralShapeSkewedGaussian	A skewed Gaussian spectral shape
SpectralShapeZero	A constant spectral shape with value 0

SpectralShape

class glotaran.builtin.megacomplexes.spectral.shape.**SpectralShape**

Bases: object

Base class for spectral shapes

Methods Summary

add_type
get_default_type

add_type

classmethod SpectralShape.**add_type**(*type_name: str*, *attribute_type: type*)

get_default_type

classmethod SpectralShape.**get_default_type**() → str

Methods Documentation

classmethod add_type(*type_name: str*, *attribute_type: type*)

classmethod get_default_type() → str

SpectralShapeGaussian

class glotaran.builtin.megacomplexes.spectral.shape.**SpectralShapeGaussian**
Bases: object

A Gaussian spectral shape

Attributes Summary

amplitude	ModelProperty is an extension of the property decorator.
label	ModelProperty is an extension of the property decorator.
location	ModelProperty is an extension of the property decorator.
type	ModelProperty is an extension of the property decorator.
width	ModelProperty is an extension of the property decorator.

amplitude

SpectralShapeGaussian.**amplitude**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

label

SpectralShapeGaussian.**label**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

location

SpectralShapeGaussian.**location**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

type

SpectralShapeGaussian.**type**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

width

SpectralShapeGaussian.**width**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

as_dict	
calculate	Calculate a normal Gaussian shape for a given `axis`.
fill	
from_dict	
markdown	
validate	

as_dict

SpectralShapeGaussian.**as_dict**() → dict

calculate

SpectralShapeGaussian.**calculate**(*axis: numpy.ndarray*) → numpy.ndarray

Calculate a normal Gaussian shape for a given `axis`.

The following equation is used for the calculation:

$$f(x, A, x_0, \Delta) = A \exp\left(-\frac{\log(2)\,(2(x - x_0))^2}{\Delta^2}\right)$$

The parameters of the equation represent the following attributes of the shape:

- x : `axis`
- A : `amplitude`
- x_0 : `location`
- Δ : `width`

In this formalism, Δ represents the full width at half maximum (FWHM). Compared to the more common definition $\exp\left(-(x-\mu)^2/(2\sigma^2)\right)$ we have $\sigma = \Delta/(2\sqrt{2\ln(2)}) = \Delta/2.35482$

Parameters **axis** (*np.ndarray*) – The axis to calculate the shape for.
Returns An array representing a Gaussian shape.
Return type np.ndarray

fill

SpectralShapeGaussian.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

from_dict

classmethod SpectralShapeGaussian.**from_dict**(*values: dict*) → cls

markdown

SpectralShapeGaussian.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

validate

SpectralShapeGaussian.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

property amplitude: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

as_dict() → dict

calculate(*axis: numpy.ndarray*) → numpy.ndarray
Calculate a normal Gaussian shape for a given `axis`.

The following equation is used for the calculation:

$$f(x, A, x_0, \Delta) = A \exp\left(-\frac{\log(2)\,(2(x - x_0))^2}{\Delta^2}\right)$$

The parameters of the equation represent the following attributes of the shape:
- x : `axis`
- A : `amplitude`
- x_0 : `location`
- Δ : `width`

In this formalism, Δ represents the full width at half maximum (FWHM). Compared to the more common definition $\exp\left(-(x-\mu)^2/(2\sigma^2)\right)$ we have $\sigma = \Delta/(2\sqrt{2\ln(2)}) = \Delta/2.35482$

Parameters **axis** (*np.ndarray*) – The axis to calculate the shape for.
Returns An array representing a Gaussian shape.
Return type np.ndarray

fill(*model: Model*, *parameters: ParameterGroup*) → cls

classmethod from_dict(*values: dict*) → cls

property label: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property location: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

property type: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

property width: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

SpectralShapeOne

class glotaran.builtin.megacomplexes.spectral.shape.**SpectralShapeOne**
Bases: object

A constant spectral shape with value 1

Attributes Summary

label	ModelProperty is an extension of the property decorator.
type	ModelProperty is an extension of the property decorator.

label

SpectralShapeOne.**label**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

type

SpectralShapeOne.**type**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

as_dict	
calculate	calculate calculates the shape.
fill	
from_dict	
markdown	
validate	

as_dict

SpectralShapeOne.**as_dict**() → dict

calculate

SpectralShapeOne.**calculate**(*axis: numpy.ndarray*) → numpy.ndarray
calculate calculates the shape.
Parameters **axis** (*np.ndarray*) – The axis to calculate the shape on.
Returns **shape**
Return type numpy.ndarray

fill

SpectralShapeOne.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

from_dict

classmethod SpectralShapeOne.**from_dict**(*values: dict*) → cls

markdown

SpectralShapeOne.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

validate

SpectralShapeOne.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

as_dict() → dict

calculate(*axis: numpy.ndarray*) → numpy.ndarray
calculate calculates the shape.
Parameters **axis** (*np.ndarray*) – The axis to calculate the shape on.
Returns **shape**
Return type numpy.ndarray

fill(*model: Model*, *parameters: ParameterGroup*) → cls

classmethod from_dict(*values: dict*) → cls

property label: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

property type: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

SpectralShapeSkewedGaussian

class glotaran.builtin.megacomplexes.spectral.shape.**SpectralShapeSkewedGaussian**
Bases: *glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeGaussian*

A skewed Gaussian spectral shape

Attributes Summary

amplitude	ModelProperty is an extension of the property decorator.
label	ModelProperty is an extension of the property decorator.
location	ModelProperty is an extension of the property decorator.
skewness	ModelProperty is an extension of the property decorator.
type	ModelProperty is an extension of the property decorator.
width	ModelProperty is an extension of the property decorator.

amplitude

SpectralShapeSkewedGaussian.**amplitude**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

label

SpectralShapeSkewedGaussian.**label**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

location

SpectralShapeSkewedGaussian.**location**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

skewness

SpectralShapeSkewedGaussian.**skewness**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

type

SpectralShapeSkewedGaussian.**type**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

width

SpectralShapeSkewedGaussian.**width**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

as_dict	
calculate	Calculate the skewed Gaussian shape for `axis`.
fill	
from_dict	
markdown	
validate	

as_dict

SpectralShapeSkewedGaussian.**as_dict**() → dict

calculate

SpectralShapeSkewedGaussian.**calculate**(*axis: numpy.ndarray*) → numpy.ndarray

Calculate the skewed Gaussian shape for `axis`.

The following equation is used for the calculation:

$$f(x, x_0, A, \Delta, b) = \begin{cases} 0 & \text{if } \theta \leq 0 \\ A \exp\left(-\frac{\log(2) \log\left(\theta(x, x_0, \Delta, b)\right)^2}{b^2}\right) & \text{if } \theta > 0 \end{cases}$$

With:

$$\theta(x, x_0, \Delta, b) = \frac{2b\,(x - x_0) + \Delta}{\Delta}$$

The parameters of the equation represent the following attributes of the shape:

- x : `axis`
- A : `amplitude`
- x_0 : `location`
- Δ : `width`
- b : `skewness`

Where Δ represents the full width at half maximum (FWHM), see `calculate_gaussian()`.

Note that in the limit of skewness parameter b equal to zero $f(x, x_0, A, \Delta, b)$ simplifies to a normal gaussian (since $\lim_{b \to 0} \frac{\ln(1+bx)}{b} = x$), see the definition in *SpectralShapeGaussian.calculate()*.

Parameters **axis** (*np.ndarray*) – The axis to calculate the shape for.
Returns An array representing a skewed Gaussian shape.
Return type np.ndarray

fill

SpectralShapeSkewedGaussian.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

from_dict

classmethod SpectralShapeSkewedGaussian.**from_dict**(*values: dict*) → cls

markdown

SpectralShapeSkewedGaussian.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

validate

SpectralShapeSkewedGaussian.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

property amplitude: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

as_dict() → dict

calculate(*axis: numpy.ndarray*) → numpy.ndarray
Calculate the skewed Gaussian shape for `axis`.

The following equation is used for the calculation:

$$f(x, x_0, A, \Delta, b) = \begin{cases} 0 & \text{if } \theta \leq 0 \\ A \exp\left(-\frac{\log(2)\log\left(\theta(x, x_0, \Delta, b)\right)^2}{b^2}\right) & \text{if } \theta > 0 \end{cases}$$

With:

$$\theta(x, x_0, \Delta, b) = \frac{2b\,(x - x_0) + \Delta}{\Delta}$$

The parameters of the equation represent the following attributes of the shape:
- x : `axis`
- A : `amplitude`
- x_0 : `location`
- Δ : `width`
- b : `skewness`

Where Δ represents the full width at half maximum (FWHM), see `calculate_gaussian()`.

Note that in the limit of skewness parameter b equal to zero $f(x, x_0, A, \Delta, b)$ simplifies to a normal gaussian (since $\lim_{b \to 0} \frac{\ln(1+bx)}{b} = x$), see the definition in *SpectralShapeGaussian.calculate()*.

Parameters **axis** (*np.ndarray*) – The axis to calculate the shape for.
Returns An array representing a skewed Gaussian shape.
Return type np.ndarray

fill(*model: Model*, *parameters: ParameterGroup*) → cls

classmethod from_dict(*values: dict*) → cls

property label: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property location: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

property skewness: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property type: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

property width: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

SpectralShapeZero

class glotaran.builtin.megacomplexes.spectral.shape.**SpectralShapeZero**
Bases: object

A constant spectral shape with value 0

Attributes Summary

label	ModelProperty is an extension of the property decorator.
type	ModelProperty is an extension of the property decorator.

label

SpectralShapeZero.**label**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

type

SpectralShapeZero.**type**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

as_dict	
calculate	calculate calculates the shape.
fill	
from_dict	
markdown	
validate	

as_dict

SpectralShapeZero.**as_dict**() → dict

calculate

SpectralShapeZero.**calculate**(*axis: numpy.ndarray*) → numpy.ndarray
calculate calculates the shape.

Only works after calling `fill`.
Parameters **axis** (*np.ndarray*) – The axis to calculate the shape on.
Returns **shape**
Return type numpy.ndarray

fill

SpectralShapeZero.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

from_dict

classmethod SpectralShapeZero.**from_dict**(*values: dict*) → cls

markdown

SpectralShapeZero.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

validate

SpectralShapeZero.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

as_dict() → dict

calculate(*axis: numpy.ndarray*) → numpy.ndarray
calculate calculates the shape.

Only works after calling `fill`.
Parameters **axis** (*np.ndarray*) – The axis to calculate the shape on.
Returns **shape**
Return type numpy.ndarray

fill(*model: Model*, *parameters: ParameterGroup*) → cls

classmethod from_dict(*values: dict*) → cls

property label: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

property type: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

spectral_megacomplex

Classes

Summary

SpectralMegacomplex

SpectralMegacomplex

class glotaran.builtin.megacomplexes.spectral.spectral_megacomplex.**SpectralMegacomplex**

Bases: glotaran.model.megacomplex.Megacomplex

Attributes Summary

dimension	ModelProperty is an extension of the property decorator.
label	ModelProperty is an extension of the property decorator.
name	
shape	ModelProperty is an extension of the property decorator.
type	ModelProperty is an extension of the property decorator.

dimension

SpectralMegacomplex.**dimension**

ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

label

SpectralMegacomplex.**label**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

name

SpectralMegacomplex.**name = 'spectral'**

shape

SpectralMegacomplex.**shape**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

type

SpectralMegacomplex.**type**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

as_dict
calculate_matrix
fill
finalize_data
from_dict
glotaran_dataset_model_items
glotaran_dataset_properties
glotaran_model_items
glotaran_unique
index_dependent
markdown

continues on next page

Table 104 – continued from previous page

validate

as_dict

SpectralMegacomplex.**as_dict**() → dict

calculate_matrix

SpectralMegacomplex.**calculate_matrix**(*dataset_model: DatasetModel*, *indices: dict[str, int]*, ***kwargs*)

fill

SpectralMegacomplex.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

finalize_data

SpectralMegacomplex.**finalize_data**(*dataset_model:* glotaran.model.dataset_model.DatasetModel, *dataset: xarray.core.dataset.Dataset*, *is_full_model: bool = False*, *as_global: bool = False*)

from_dict

classmethod SpectralMegacomplex.**from_dict**(*values: dict*) → cls

glotaran_dataset_model_items

classmethod SpectralMegacomplex.**glotaran_dataset_model_items**() → str

glotaran_dataset_properties

classmethod SpectralMegacomplex.**glotaran_dataset_properties**() → str

glotaran_model_items

classmethod SpectralMegacomplex.**glotaran_model_items**() → str

glotaran_unique

classmethod SpectralMegacomplex.**glotaran_unique**() → bool

index_dependent

SpectralMegacomplex.**index_dependent**(*dataset_model:* glotaran.model.dataset_model.DatasetModel) → bool

markdown

SpectralMegacomplex.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

validate

SpectralMegacomplex.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

as_dict() → dict

calculate_matrix(*dataset_model: DatasetModel*, *indices: dict[str, int]*, ***kwargs*)

property dimension: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

fill(*model: Model*, *parameters: ParameterGroup*) → cls

finalize_data(*dataset_model:* glotaran.model.dataset_model.DatasetModel, *dataset:* *xarray.core.dataset.Dataset*, *is_full_model:* *bool* = *False*, *as_global:* *bool* = *False*)

classmethod from_dict(*values:* *dict*) → cls

classmethod glotaran_dataset_model_items() → str

classmethod glotaran_dataset_properties() → str

classmethod glotaran_model_items() → str

classmethod glotaran_unique() → bool

index_dependent(*dataset_model:* glotaran.model.dataset_model.DatasetModel) → bool

property label: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup* = *None*, *initial_parameters: ParameterGroup* = *None*) → MarkdownStr

name = 'spectral'

property shape: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property type: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup* | *None* = *None*) → list[str]

15.1.3 cli

Modules

glotaran.cli.commands	
glotaran.cli.main	The glotaran CLI main function.

commands

Modules

glotaran.cli.commands.explore
glotaran.cli.commands.export
glotaran.cli.commands.optimize
glotaran.cli.commands.pluginlist
glotaran.cli.commands.print
glotaran.cli.commands.util
glotaran.cli.commands.validate

explore

Functions

Summary

export	Exports data from netCDF4 to ascii.

export

glotaran.cli.commands.explore.**export**(*filename: str*, *select*, *out: str*, *name: str*)
Exports data from netCDF4 to ascii.

export

optimize

Functions

Summary

optimize_cmd	Optimizes a model.

optimize_cmd

glotaran.cli.commands.optimize.**optimize_cmd**(*dataformat: str*, *data: List[str]*, *out: str*, *outformat: str*, *nfev: int*, *nnls: bool*, *yes: bool*, *parameters_file: str*, *model_file: str*, *scheme_file: str*)

Optimizes a model. e.g.: glotaran optimize –

pluginlist

Functions

Summary

plugin_list_cmd	Prints a list of installed plugins.

plugin_list_cmd

glotaran.cli.commands.pluginlist.**plugin_list_cmd**()

Prints a list of installed plugins.

print

Functions

Summary

print_cmd	Parses scheme, a model or a parameter file and prints the result as a Markdown formatted string.

print_cmd

glotaran.cli.commands.print.**print_cmd**(*parameters_file: str*, *model_file: str*, *scheme_file: str*)

Parses scheme, a model or a parameter file and prints the result as a Markdown formatted string.

util

Functions

Summary

load_dataset_file	

continues on next page

Table 111 – continued from previous page

load_model_file	
load_parameter_file	
load_scheme_file	
project_io_list_supporting_plugins	List all project-io plugin that implement `method_name`.
select_data	
select_name	
signature_analysis	
write_data	

load_dataset_file

glotaran.cli.commands.util.**load_dataset_file**(*filename*, *fmt=None*, *verbose=False*)

load_model_file

glotaran.cli.commands.util.**load_model_file**(*filename*, *verbose=False*)

load_parameter_file

glotaran.cli.commands.util.**load_parameter_file**(*filename*, *fmt=None*, *verbose=False*)

load_scheme_file

glotaran.cli.commands.util.**load_scheme_file**(*filename*, *verbose=False*)

project_io_list_supporting_plugins

glotaran.cli.commands.util.**project_io_list_supporting_plugins**(*method_name: str*, *block_list: Iterable[str] | None = None*) → Iterable[str]

List all project-io plugin that implement `method_name`.

Parameters

- **method_name** (*str*) – Name of the method which should be supported.

- **block_list** (*Iterable[str]*) – Iterable of plugin names which should be omitted.

select_data

glotaran.cli.commands.util.**select_data**(*data*, *dim*, *selection*)

select_name

glotaran.cli.commands.util.**select_name**(*filename*, *dataset*)

signature_analysis

glotaran.cli.commands.util.**signature_analysis**(*cmd*)

write_data

glotaran.cli.commands.util.**write_data**(*data*, *out*)

Classes

Summary

ValOrRangeOrList

ValOrRangeOrList

class glotaran.cli.commands.util.**ValOrRangeOrList**
Bases: click.types.ParamType

Attributes Summary

arity	
envvar_list_splitter	if a list of this type is expected and the value is pulled from a string environment variable, this is what splits it up.
is_composite	

continues on next page

Table 113 – continued from previous page

name	the descriptive name of this type

arity

ValOrRangeOrList.**arity: ClassVar[int] = 1**

envvar_list_splitter

ValOrRangeOrList.**envvar_list_splitter: ClassVar[Optional[str]] = None**
if a list of this type is expected and the value is pulled from a string environment variable, this is what splits it up. *None* means any whitespace. For all parameters the general rule is that whitespace splits them up. The exception are paths and files which are split by `os.path.pathsep` by default (":" on Unix and ";" on Windows).

is_composite

ValOrRangeOrList.**is_composite: ClassVar[bool] = False**

name

ValOrRangeOrList.**name: str = 'number or range or list'**
the descriptive name of this type

Methods Summary

convert	Convert the value to the correct type.
fail	Helper method to fail with an invalid value message.
get_metavar	Returns the metavar default for this param if it provides one.
get_missing_message	Optionally might return extra information about a missing parameter.
shell_complete	Return a list of `CompletionItem` objects for the incomplete value.
split_envvar_value	Given a value from an environment variable this splits it up into small chunks depending on the defined envvar list splitter.
to_info_dict	Gather information that could be useful for a tool generating user-facing documentation.

convert

ValOrRangeOrList.**convert**(*value*, *param*, *ctx*)

Convert the value to the correct type. This is not called if the value is `None` (the missing value).

This must accept string values from the command line, as well as values that are already the correct type. It may also convert other compatible types.

The `param` and `ctx` arguments may be `None` in certain situations, such as when converting prompt input.

If the value cannot be converted, call *fail()* with a descriptive message.

Parameters

- **value** – The value to convert.
- **param** – The parameter that is using this type to convert its value. May be `None`.
- **ctx** – The current context that arrived at this value. May be `None`.

fail

ValOrRangeOrList.**fail**(*message: str*, *param: Optional[Parameter] = None*, *ctx: Optional[Context] = None*) → t.NoReturn

Helper method to fail with an invalid value message.

get_metavar

ValOrRangeOrList.**get_metavar**(*param: Parameter*) → Optional[str]

Returns the metavar default for this param if it provides one.

get_missing_message

ValOrRangeOrList.**get_missing_message**(*param: Parameter*) → Optional[str]

Optionally might return extra information about a missing parameter.

New in version 2.0.

shell_complete

ValOrRangeOrList.**shell_complete**(*ctx: Context*, *param: Parameter*, *incomplete: str*) → List[CompletionItem]

Return a list of `CompletionItem` objects for the incomplete value. Most types do not provide completions, but some do, and this allows custom types to provide custom completions as well.

Parameters

- **ctx** – Invocation context for this command.
- **param** – The parameter that is requesting completion.
- **incomplete** – Value being completed. May be empty.

New in version 8.0.

split_envvar_value

ValOrRangeOrList.**split_envvar_value**(*rv: str*) → Sequence[str]

Given a value from an environment variable this splits it up into small chunks depending on the defined envvar list splitter.

If the splitter is set to *None*, which means that whitespace splits, then leading and trailing whitespace is ignored. Otherwise, leading and trailing splitters usually lead to empty items being included.

to_info_dict

ValOrRangeOrList.**to_info_dict**() → Dict[str, Any]

Gather information that could be useful for a tool generating user-facing documentation.

Use `click.Context.to_info_dict()` to traverse the entire CLI structure.

New in version 8.0.

Methods Documentation

arity: ClassVar[int] = 1

convert(*value*, *param*, *ctx*)

Convert the value to the correct type. This is not called if the value is `None` (the missing value).

This must accept string values from the command line, as well as values that are already the correct type. It may also convert other compatible types.

The `param` and `ctx` arguments may be `None` in certain situations, such as when converting prompt input.

If the value cannot be converted, call *fail()* with a descriptive message.

Parameters

- **value** – The value to convert.
- **param** – The parameter that is using this type to convert its value. May be `None`.
- **ctx** – The current context that arrived at this value. May be `None`.

envvar_list_splitter: ClassVar[Optional[str]] = None

if a list of this type is expected and the value is pulled from a string environment variable, this is what splits it up. *None* means any whitespace. For all parameters the general rule is that whitespace splits them up. The exception are paths and files which are split by `os.path.pathsep` by default (":" on Unix and ";" on Windows).

fail(*message: str*, *param: Optional[Parameter] = None*, *ctx: Optional[Context] = None*) → t.NoReturn

Helper method to fail with an invalid value message.

get_metavar(*param: Parameter*) → Optional[str]

Returns the metavar default for this param if it provides one.

get_missing_message(*param: Parameter*) → Optional[str]

Optionally might return extra information about a missing parameter.

New in version 2.0.

is_composite: ClassVar[bool] = False

name: str = 'number or range or list'
the descriptive name of this type

shell_complete(*ctx: Context*, *param: Parameter*, *incomplete: str*) → List[CompletionItem]
Return a list of `CompletionItem` objects for the incomplete value. Most types do not provide completions, but some do, and this allows custom types to provide custom completions as well.

Parameters

- **ctx** – Invocation context for this command.
- **param** – The parameter that is requesting completion.
- **incomplete** – Value being completed. May be empty.

New in version 8.0.

split_envvar_value(*rv: str*) → Sequence[str]
Given a value from an environment variable this splits it up into small chunks depending on the defined envvar list splitter.

If the splitter is set to *None*, which means that whitespace splits, then leading and trailing whitespace is ignored. Otherwise, leading and trailing splitters usually lead to empty items being included.

to_info_dict() → Dict[str, Any]
Gather information that could be useful for a tool generating user-facing documentation.

Use `click.Context.to_info_dict()` to traverse the entire CLI structure.

New in version 8.0.

validate

Functions

Summary

validate_cmd	Validates a model file and optionally a parameter file.

validate_cmd

glotaran.cli.commands.validate.**validate_cmd**(*parameters_file: str*, *model_file: str*, *scheme_file: str*)
Validates a model file and optionally a parameter file.

main

glotaran.cli.**main = <Cli main>**
The glotaran CLI main function.

15.1.4 deprecation

Deprecation helpers and place to put deprecated implementations till removing.

Modules

glotaran.deprecation.deprecation_utils	Helper functions to give deprecation warnings.
glotaran.deprecation.modules	Package containing deprecated implementations which were removed.

deprecation_utils

Helper functions to give deprecation warnings.

Functions

Summary

check_overdue	Check if a deprecation is overdue for removal.
check_qualnames_in_tests	Test that qualnames import path exists when running tests.
deprecate	Decorate a function, method or class to deprecate it.
deprecate_dict_entry	Replace dict entry inplace and warn about usage change, if present in the dict.
deprecate_module_attribute	Import and return and anttribute from the new location.
deprecate_submodule	Create a module at runtime which retrieves attributes from new module.
glotaran_version	Version of the distribution.
module_attribute	Import and return the attribute (e.g.
parse_version	Parse version string to tuple of three ints for comparison.
raise_deprecation_error	Raise `GlotaranDeprectedApiError` error, with formatted message.
warn_deprecated	Raise deprecation warning with change information.

check_overdue

glotaran.deprecation.deprecation_utils.**check_overdue**(*deprecated_qual_name_usage: str*, *to_be_removed_in_version: str*) → None

Check if a deprecation is overdue for removal.

Parameters

- **deprecated_qual_name_usage** (*str*) – Old usage with fully qualified name e.g.: `'glotaran.read_model_from_yaml(model_yml_str)'`
- **to_be_removed_in_version** (*str*) – Version the support for this usage will be removed.

Raises **OverDueDeprecation** – If the current version is greater or equal to `to_be_removed_in_version`.

check_qualnames_in_tests

glotaran.deprecation.deprecation_utils.**check_qualnames_in_tests**(*qual_names: Sequence[str]*, *importable_indices: Sequence[int]*)

Test that qualnames import path exists when running tests.

All deprecations should be tested anyway in order to get the proper errors when a deprecation is overdue. This helperfunction also helps to ensure that at least the import paths (`qual_names`) of the old and new usage exist.

Parameters

- **qual_names** (*Sequence[str]*) – Sequence of fully qualified module attribute names, optionally with call arguments.
- **importable_indices** (*Sequence[int]*) – Indices of corresponding to `qual_names` indicating how to slice each `qual_name` split at `.`, for the import and attribute checking.

See also:

warn_deprecated, *deprecate*

deprecate

glotaran.deprecation.deprecation_utils.**deprecate**(*, *deprecated_qual_name_usage: str*, *new_qual_name_usage: str*, *to_be_removed_in_version: str*, *has_glotaran_replacement: bool = True*, *importable_indices: tuple[int, int] = (1, 1)*) → Callable[[DecoratedCallable], DecoratedCallable]

Decorate a function, method or class to deprecate it.

This raises deprecation warning with old / new usage information and end of support version.

Parameters

- **deprecated_qual_name_usage** (*str*) – Old usage with fully qualified name e.g.: `'glotaran.read_model_from_yaml(model_yml_str)'`
- **new_qual_name_usage** (*str*) – New usage as fully qualified name e.g.: `'glotaran.io.load_model(model_yml_str, format_name="yml_str")'`
- **to_be_removed_in_version** (*str*) – Version the support for this usage will be removed.
- **has_glotaran_replacement** (*bool*) – Whether or not this functionality has a replacement in core pyglotaran. This will be mapped to the second entry of `check_qualnames` in *warn_deprecated()*.
- **importable_indices** (*Sequence[int]*) – Indices from right for most nested item which is importable for `deprecated_qual_name_usage` and `new_qual_name_usage` after splitting at `.`. This is used when the old or new usage is a method or mapping access. E.g. let `deprecated_qual_name_usage` be `package.module.class.mapping["key"]`, then you would use `importable_indices=(2, 1)`, this way func:*check_qualnames_in_tests* will import `package.module.class` and check if `class` has an attribute `mapping`. Default

Returns Original function or class throwing a Deprecation warning when used.

Return type DecoratedCallable

Raises **OverDueDeprecation** – If the current version is greater or equal to `to_be_removed_in_version`.

See also:

warn_deprecated, *deprecate_module_attribute*, *deprecate_submodule*, *check_qualnames_in_tests*

Examples

This is the way the old `read_parameters_from_yaml_file` was deprecated and the usage of `load_model` was promoted instead.

Listing 1: glotaran/deprecation/modules/glotaran_root.py

```
@deprecate(
    deprecated_qualname_usage="glotaran.read_parameters_from_yaml_
↪file(model_path)",
    new_qualname_usage="glotaran.io.load_model(model_path)",
    to_be_removed_in_version="0.6.0",
)
def read_parameters_from_yaml_file(model_path: str):
    return load_model(model_path)
```

deprecate_dict_entry

glotaran.deprecation.deprecation_utils.**deprecate_dict_entry**(*, *dict_to_check: MutableMapping[Hashable, Any]*, *deprecated_usage: str*, *new_usage: str*, *to_be_removed_in_version: str*, *swap_keys: tuple[Hashable, Hashable] | None = None*, *replace_rules: tuple[Mapping[Hashable, Any], Mapping[Hashable, Any]] | None = None*, *stacklevel: int = 3*) → None

Replace dict entry inplace and warn about usage change, if present in the dict.

Parameters

- **dict_to_check** (*MutableMapping[Hashable, Any]*) – Dict which should be checked.
- **deprecated_usage** (*str*) – Old usage to inform user (only used in warning).
- **new_usage** (*str*) – New usage to inform user (only used in warning).
- **to_be_removed_in_version** (*str*) – Version the support for this usage will be removed.
- **swap_keys** (*tuple[Hashable, Hashable]*) – (old_key, new_key), `dict_to_check[new_key]` will be assigned the value `dict_to_check[old_key]` and `old_key` will be removed from the dict. by default None
- **replace_rules** (*Mapping[Hashable, tuple[Any, Any]]*) – ({old_key: old_value}, {new_key: new_value}), If `dict_to_check[old_key]` has the value `old_value`, `dict_to_check[new_key]` it will be set to `new_value`. `old_key` will be removed from the dict if `old_key` and `new_key` aren't equal. by default None
- **stacklevel** (*int*) – Stack at which the warning should be shown as raise. , by default 3

Raises

- **ValueError** – If both `swap_keys` and `replace_rules` are None (default) or not None.
- **OverDueDeprecation** – If the current version is greater or equal to `to_be_removed_in_version`.

See also:

warn_deprecated

Notes

To prevent confusion exactly one of `replace_rules` and `swap_keys` needs to be passed.

Examples

For readability sake the warnings won't be shown in the examples.

Swapping key names:

```
>>> dict_to_check = {"foo": 123}
>>> deprecate_dict_entry(
        dict_to_check=dict_to_check,
        deprecated_usage="foo",
        new_usage="bar",
        to_be_removed_in_version="0.6.0",
        swap_keys=("foo", "bar")
    )
>>> dict_to_check
{"bar": 123}
```

Changing values:

```
>>> dict_to_check = {"foo": 123}
>>> deprecate_dict_entry(
        dict_to_check=dict_to_check,
        deprecated_usage="foo: 123",
        new_usage="foo: 123.0",
        to_be_removed_in_version="0.6.0",
        replace_rules=({"foo": 123}, {"foo": 123.0})
    )
>>> dict_to_check
{"foo": 123.0}
```

Swapping key names AND changing values:

```
>>> dict_to_check = {"type": "kinetic-spectrum"}
>>> deprecate_dict_entry(
        dict_to_check=dict_to_check,
        deprecated_usage="type: kinectic-spectrum",
        new_usage="default_megacomplex: decay",
        to_be_removed_in_version="0.6.0",
        replace_rules=({"type": "kinetic-spectrum"}, {"default_megacomplex
↪": "decay"})
```

(continues on next page)

(continued from previous page)

```
    )
>>> dict_to_check
{"default_megacomplex": "decay"}
```

deprecate_module_attribute

glotaran.deprecation.deprecation_utils.**deprecate_module_attribute**(*, *deprecated_qual_name: str*, *new_qual_name: str*, *to_be_removed_in_version: str*) → Any

Import and return and anttribute from the new location.

This needs to be wrapped in the definition of a module wide `__getattr__` function so it won't throw warnings all the time (see example).

Parameters

- **deprecated_qual_name** (*str*) – Fully qualified name of the deprecated attribute e.g.: `glotaran.ParameterGroup`
- **new_qual_name** (*str*) – Fully qualified name of the new attribute e.g.: `glotaran.parameter.ParameterGroup`
- **to_be_removed_in_version** (*str*) – Version the support for this usage will be removed.

Returns Module attribute from its new location.

Return type Any

Raises **OverDueDeprecation** – If the current version is greater or equal to `to_be_removed_in_version`.

See also:

deprecate, *warn_deprecated*, *deprecate_submodule*

Examples

When deprecating the usage of `ParameterGroup` the root of `glotaran` and promoting to import it from `glotaran.parameter` the following code was added to the root `__init__.py`.

Listing 2: glotaran/__init__.py

```
def __getattr__(attribute_name: str):
    from glotaran.deprecation import deprecate_module_attribute

    if attribute_name == "ParameterGroup":
        return deprecate_module_attribute(
            deprecated_qual_name="glotaran.ParameterGroup",
            new_qual_name="glotaran.parameter.ParameterGroup",
            to_be_removed_in_version="0.6.0",
```

(continues on next page)

(continued from previous page)

```
        )

    raise AttributeError(f"module {__name__} has no attribute {attribute_
↪name}")
```

deprecate_submodule

glotaran.deprecation.deprecation_utils.**deprecate_submodule**(*, *deprecated_module_name: str*, *new_module_name: str*, *to_be_removed_in_version: str*) → module

Create a module at runtime which retrieves attributes from new module.

When moving a module, create a variable with the modules name in the parent packages `__init__.py`, so imports will be redirected to the new module location and a deprecation warning will be given, to help the user adjust the outdated code. Each time an attribute is retrieved there will be a deprecation warning.

Parameters

- **deprecated_module_name** (*str*) – Fully qualified name of the deprecated module e.g.: `'glotaran.analysis.result'`
- **new_module_name** (*str*) – Fully qualified name of the new module e.g.: `'glotaran.project.result'`
- **to_be_removed_in_version** (*str*) – Version the support for this usage will be removed.

Returns Module containing

Return type ModuleType

Raises **OverDueDeprecation** – If the current version is greater or equal to `to_be_removed_in_version`.

See also:

deprecate, *deprecate_module_attribute*

Examples

When moving the module `result` from `glotaran.analysis.result` to `glotaran.project.result` the following code was added to the old parent packages (`glotaran.analysis`) `__init__.py`.

Listing 3: glotaran/analysis/__init__.py

```
from glotaran.deprecation.deprecation_utils import deprecate_submodule

result = deprecate_submodule(
    deprecated_module_name="glotaran.analysis.result",
    new_module_name="glotaran.project.result",
```

(continues on next page)

(continued from previous page)

```
    to_be_removed_in_version="0.6.0",
)
```

glotaran_version

glotaran.deprecation.deprecation_utils.**glotaran_version**() → str

Version of the distribution.

This is basically the same as `glotaran.__version__` but independent from glotaran. This way all of the deprecation functionality can be used even in `glotaran.__init__.py` without moving the import below the definition of `__version__` or causeing a circular import issue.

Returns The version string.

Return type str

module_attribute

glotaran.deprecation.deprecation_utils.**module_attribute**(*module_qual_name: str*, *attribute_name: str*) → Any

Import and return the attribute (e.g. function or class) of a module.

This is basically the same as `from module_name import attribute_name as return_value` where this function returns `return_value`.

Parameters

- **module_qual_name** (*str*) – Fully qualified name for a module e.g. `glotaran.model.base_model`
- **attribute_name** (*str*) – Name of the attribute e.g. `Model`

Returns Attribute of the module, e.g. a function or class.

Return type Any

parse_version

glotaran.deprecation.deprecation_utils.**parse_version**(*version_str: str*) → tuple[int, int, int]

Parse version string to tuple of three ints for comparison.

Parameters **version_str** (*str*) – Fully qualified version string of the form 'major.minor.patch'.

Returns Version as tuple.

Return type tuple[int, int, int]

Raises

- **ValueError** – If `version_str` has less that three elements separated by `.`.
- **ValueError** – If `version_str` 's first three elements can not be casted to int.

raise_deprecation_error

glotaran.deprecation.deprecation_utils.**raise_deprecation_error**(*, *deprecated_qual_name_usage: str*, *new_qual_name_usage: str*, *to_be_removed_in_version: str*) → NoReturn

Raise `GlotaranDeprectedApiError` error, with formatted message.

This should only be used if there is no reasonable way to keep the deprecated usage functional!

Parameters

- **deprecated_qual_name_usage** (*str*) – Old usage with fully qualified name e.g.: `'glotaran.read_model_from_yaml(model_yml_str)'`
- **new_qual_name_usage** (*str*) – New usage as fully qualified name e.g.: `'glotaran.io.load_model(model_yml_str, format_name="yml_str")'`
- **to_be_removed_in_version** (*str*) – Version the support for this usage will be removed.

Raises

- **OverDueDeprecation** – If the current version is greater or equal to `to_be_removed_in_version`.
- **GlotaranDeprectedApiError** – If `OverDueDeprecation` wasn't raised before.

warn_deprecated

glotaran.deprecation.deprecation_utils.**warn_deprecated**(*, *deprecated_qual_name_usage: str*, *new_qual_name_usage: str*, *to_be_removed_in_version: str*, *check_qual_names: tuple[bool, bool] = (True, True)*, *stacklevel: int = 2*, *importable_indices: tuple[int, int] = (1, 1)*) → None

Raise deprecation warning with change information.

The change information are old / new usage information and end of support version.

Parameters

- **deprecated_qual_name_usage** (*str*) – Old usage with fully qualified name e.g.: `'glotaran.read_model_from_yaml(model_yml_str)'`
- **new_qual_name_usage** (*str*) – New usage as fully qualified name e.g.: `'glotaran.io.load_model(model_yml_str, format_name="yml_str")'`
- **to_be_removed_in_version** (*str*) – Version the support for this usage will be removed.
- **check_qual_names** (*tuple[bool, bool]*) – Whether or not to check for the existence `deprecated_qual_name_usage` and `deprecated_qual_name_usage`

- Set the first value to False to prevent infinite recursion error when changing a module attribute import.
- Set the second value to False if the new usage in in a different package or there is none.

- **stacklevel** (*int*) – Stack at which the warning should be shown as raise. Default: 2
- **importable_indices** (*tuple[int, int]*) – Indices from right for most nested item which is importable for `deprecated_qual_name_usage` and `new_qual_name_usage` after splitting at `.`. This is used when the old or new usage is a method or mapping access. E.g. let `deprecated_qual_name_usage` be `package.module.class.mapping["key"]`, then you would use `importable_indices=(2, 1)`, this way func:*check_qualnames_in_tests* will import `package.module.class` and check if `class` has an attribute `mapping`.

Raises **OverDueDeprecation** – If the current version is greater or equal to `to_be_removed_in_version`.

See also:

deprecate, *deprecate_module_attribute*, *deprecate_submodule*, *check_qualnames_in_tests*

Examples

This is the way the old `read_parameters_from_yaml_file` could deprecated and the usage of `load_model` being promoted instead.

Listing 4: glotaran/deprecation/modules/glotaran_root.py

```
def read_parameters_from_yaml_file(model_path: str):
    warn_deprecated(
        deprecated_qual_name_usage="glotaran.read_parameters_from_yaml_
↪file(model_path)",
        new_qual_name_usage="glotaran.io.load_model.load_model(model_path)
↪",
        to_be_removed_in_version="0.6.0",
    )
    return load_model(model_path)
```

Exceptions

Exception Summary

GlotaranApiDeprecationWarning	Warning to give users about API changes.
GlotaranDeprectedApiError	Exception raised when a deprecation has no replacement.
OverDueDeprecation	Error thrown when a deprecation should have been removed.

GlotaranApiDeprecationWarning

exception glotaran.deprecation.deprecation_utils.**GlotaranApiDeprecationWarning**
Warning to give users about API changes.

See also:

deprecate, *warn_deprecated*, *deprecate_module_attribute*, *deprecate_submodule*, *deprecate_dict_entry*

GlotaranDeprectedApiError

exception glotaran.deprecation.deprecation_utils.**GlotaranDeprectedApiError**
Exception raised when a deprecation has no replacement.

See also:

deprecate, *warn_deprecated*, *deprecate_module_attribute*, *deprecate_submodule*, *deprecate_dict_entry*

OverDueDeprecation

exception glotaran.deprecation.deprecation_utils.**OverDueDeprecation**
Error thrown when a deprecation should have been removed.

See also:

deprecate, *warn_deprecated*, *deprecate_module_attribute*, *deprecate_submodule*, *deprecate_dict_entry*

modules

Package containing deprecated implementations which were removed.

To keep things organized the filenames should be like the relative import path from glotaran root, but with _ instead of `.`. E.g. `glotaran.analysis.scheme` would map to `analysis_scheme.py`

The only exceptions to this rule are the root `__init__.py` which is named `glotaran_root.py` and testing changed imports which should be placed in `test_changed_imports.py`.

Modules

glotaran.deprecation.modules.builtin_io_yml	Deprecation functions for the yaml parser.
glotaran.deprecation.modules.glotaran_root	Deprecated attributes from `glotaran.__init__` which are removed.

builtin_io_yml

Deprecation functions for the yaml parser.

Functions

Summary

model_spec_deprecations	Check deprecations in the model specification spec dict.
scheme_spec_deprecations	Check deprecations in the scheme specification spec dict.

model_spec_deprecations

glotaran.deprecation.modules.builtin_io_yml.**model_spec_deprecations**(*spec: MutableMapping[Any, Any]*) → None

Check deprecations in the model specification spec dict.

Parameters **spec** (*MutableMapping[Any, Any]*) – Model specification dictionary

scheme_spec_deprecations

glotaran.deprecation.modules.builtin_io_yml.**scheme_spec_deprecations**(*spec: MutableMapping[Any, Any]*) → None

Check deprecations in the scheme specification spec dict.

Parameters **spec** (*MutableMapping[Any, Any]*) – Scheme specification dictionary

glotaran_root

Deprecated attributes from glotaran.__init__ which are removed.

Functions

Summary

read_model_from_yaml	Parse yaml string to Model.
read_model_from_yaml_file	Parse model.yaml file to Model.
read_parameters_from_csv_file	Parse parameters_file to ParameterGroup.
read_parameters_from_yaml	Parse yaml string to ParameterGroup.
read_parameters_from_yaml_file	Parse parameters_file to ParameterGroup.

read_model_from_yaml

glotaran.deprecation.modules.glotaran_root.**read_model_from_yaml**(*model_yml_str: str*) → Model

Parse `yaml` string to `Model`.

> **Warning:** Deprecated use `glotaran.io.load_model(model_yml_str, format_name="yml_str")` instead.

Parameters **model_yml_str** (*str*) – Model spec description in yaml.

Returns Model described in `model_yml_str`.

Return type *Model*

read_model_from_yaml_file

glotaran.deprecation.modules.glotaran_root.**read_model_from_yaml_file**(*model_file: str*) → Model

Parse `model.yaml` file to `Model`.

> **Warning:** Deprecated use `glotaran.io.load_model(model_file)` instead.

Parameters **model_file** (*str*) – File with model spec description as yaml.

Returns Model described in `model_file`.

Return type *Model*

read_parameters_from_csv_file

glotaran.deprecation.modules.glotaran_root.**read_parameters_from_csv_file**(*parameters_file: str*) → ParameterGroup

Parse `parameters_file` to `ParameterGroup`.

> **Warning:** Deprecated use `glotaran.io.load_parameters(parameters_file)` instead.

Parameters **parameters_file** (*str*) – File with parameters in csv.

Returns ParameterGroup described in `parameters_file`.

Return type *ParameterGroup*

read_parameters_from_yaml

glotaran.deprecation.modules.glotaran_root.**read_parameters_from_yaml**(*parameters_yml_str: str*) → ParameterGroup

Parse `yaml` string to `ParameterGroup`.

> **Warning:** Deprecated use `glotaran.io.load_parameters(parameters_yml_str, format_name="yml_str")` instead.

Parameters **parameters_yml_str** (*str*) – PArameter spec description in yaml.

Returns ParameterGroup described in `parameters_yml_str`.

Return type *ParameterGroup*

read_parameters_from_yaml_file

glotaran.deprecation.modules.glotaran_root.**read_parameters_from_yaml_file**(*parameters_file: str*) → ParameterGroup

Parse `parameters_file` to `ParameterGroup`.

> **Warning:** Deprecated use `glotaran.io.load_parameters(parameters_file)` instead.

Parameters **parameters_file** (*str*) – File with parameters in yaml.

Returns ParameterGroup described in `parameters_file`.

Return type *ParameterGroup*

15.1.5 examples

Modules

glotaran.examples.sequential

sequential

15.1.6 io

Functions for data IO

Note:

Since Io functionality is purely plugin based this package mostly reexports functions from the pluginsystem from a common place.

Modules

glotaran.io.interface	Baseclasses to create Data/Project IO plugins from.
glotaran.io.prepare_dataset	

interface

Baseclasses to create Data/Project IO plugins from.

The main purpose of those classes are to guarantee a consistent API via typechecker like `mypy` and demonstarate with methods are accessed by highlevel convenience functions for a given type of plugin.

To add additional options to a method, those options need to be keyword only arguments. See: https://www.python.org/dev/peps/pep-3102/

Classes

Summary

DataIoInterface	Baseclass for Data IO plugins.
ProjectIoInterface	Baseclass for Project IO plugins.

DataIoInterface

class glotaran.io.interface.**DataIoInterface**(*format_name: str*)

Bases: `object`

Baseclass for Data IO plugins.

Initialize a Data IO plugin with the name of the format.

Parameters **format_name** (*str*) – Name of the supported format an instance uses.

Methods Summary

load_dataset	Read data from a file to xarray.Dataset or xarray.DataArray (**NOT IMPLEMENTED**).
save_dataset	Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).

load_dataset

DataIoInterface.**load_dataset**(*file_name: str*) → xr.Dataset | xr.DataArray
Read data from a file to xarray.Dataset or xarray.DataArray (**NOT IMPLEMENTED**).
Parameters **file_name** (*str*) – File containing the data.
Returns Data loaded from the file.
Return type xr.Dataset|xr.DataArray

save_dataset

DataIoInterface.**save_dataset**(*dataset: xr.Dataset | xr.DataArray*, *file_name: str*)
Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).
Parameters
- **dataset** (*xr.Dataset*) – Dataset to be saved to file.
- **file_name** (*str*) – File to write the data to.

Methods Documentation

load_dataset(*file_name: str*) → xr.Dataset | xr.DataArray
Read data from a file to xarray.Dataset or xarray.DataArray (**NOT IMPLEMENTED**).
Parameters **file_name** (*str*) – File containing the data.
Returns Data loaded from the file.
Return type xr.Dataset|xr.DataArray

save_dataset(*dataset: xr.Dataset | xr.DataArray*, *file_name: str*)
Save data from xarray.Dataset to a file (**NOT IMPLEMENTED**).
Parameters
- **dataset** (*xr.Dataset*) – Dataset to be saved to file.
- **file_name** (*str*) – File to write the data to.

ProjectIoInterface

class glotaran.io.interface.**ProjectIoInterface**(*format_name: str*)
Bases: object

Baseclass for Project IO plugins.

Initialize a Project IO plugin with the name of the format.

Parameters **format_name** (*str*) – Name of the supported format an instance uses.

Methods Summary

load_model	Create a Model instance from the specs defined in a file (**NOT IMPLEMENTED**).
load_parameters	Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).
load_result	Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).
load_scheme	Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).
save_model	Save a Model instance to a spec file (**NOT IMPLEMENTED**).
save_parameters	Save a ParameterGroup instance to a spec file (**NOT IMPLEMENTED**).
save_result	Save a Result instance to a spec file (**NOT IMPLEMENTED**).
save_scheme	Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).

load_model

ProjectIoInterface.**load_model**(*file_name: str*) → Model

Create a Model instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the model specs.

Returns Model instance created from the file.

Return type *Model*

load_parameters

ProjectIoInterface.**load_parameters**(*file_name: str*) → ParameterGroup

Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **file_name** (*str*) – File containing the parameter specs.

Returns ParameterGroup instance created from the file.

Return type *ParameterGroup*

load_result

ProjectIoInterface.**load_result**(*result_path: str*) → Result

Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).

Parameters **result_path** (*str*) – Path containing the result data.

Returns Result instance created from the file.

Return type *Result*

load_scheme

ProjectIoInterface.**load_scheme**(*file_name: str*) → Scheme
Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).
Parameters **file_name** (*str*) – File containing the parameter specs.
Returns
- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model

ProjectIoInterface.**save_model**(*model: Model*, *file_name: str*)
Save a Model instance to a spec file (**NOT IMPLEMENTED**).
Parameters
- **model** (Model) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters

ProjectIoInterface.**save_parameters**(*parameters: ParameterGroup*, *file_name: str*)
Save a ParameterGroup instance to a spec file (**NOT IMPLEMENTED**).
Parameters
- **parameters** (ParameterGroup) – ParameterGroup instance to save to specs file.
- **file_name** (*str*) – File to write the parameter specs to.

save_result

ProjectIoInterface.**save_result**(*result: Result*, *result_path: str*) → list[str] | None
Save a Result instance to a spec file (**NOT IMPLEMENTED**).
Parameters
- **result** (Result) – Result instance to save to specs file.
- **result_path** (*str*) – Path to write the result data to.

save_scheme

ProjectIoInterface.**save_scheme**(*scheme: Scheme*, *file_name: str*)
Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).
Parameters
- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

Methods Documentation

load_model(*file_name: str*) → Model
Create a Model instance from the specs defined in a file (**NOT IMPLEMENTED**).
Parameters **file_name** (*str*) – File containing the model specs.
Returns Model instance created from the file.
Return type *Model*

load_parameters(*file_name: str*) → ParameterGroup
Create a ParameterGroup instance from the specs defined in a file (**NOT IMPLEMENTED**).
Parameters **file_name** (*str*) – File containing the parameter specs.
Returns ParameterGroup instance created from the file.
Return type *ParameterGroup*

load_result(*result_path: str*) → Result
Create a Result instance from the specs defined in a file (**NOT IMPLEMENTED**).
Parameters **result_path** (*str*) – Path containing the result data.
Returns Result instance created from the file.
Return type *Result*

load_scheme(*file_name: str*) → Scheme
Create a Scheme instance from the specs defined in a file (**NOT IMPLEMENTED**).
Parameters **file_name** (*str*) – File containing the parameter specs.
Returns
- *Scheme* – Scheme instance created from the file.
- **.. # noqa** (*DAR202*)
- **.. # noqa** (*DAR401*)

save_model(*model: Model*, *file_name: str*)
Save a Model instance to a spec file (**NOT IMPLEMENTED**).
Parameters
- **model** (Model) – Model instance to save to specs file.
- **file_name** (*str*) – File to write the model specs to.

save_parameters(*parameters: ParameterGroup*, *file_name: str*)
Save a ParameterGroup instance to a spec file (**NOT IMPLEMENTED**).
Parameters
- **parameters** (ParameterGroup) – ParameterGroup instance to save to specs file.
- **file_name** (*str*) – File to write the parameter specs to.

save_result(*result: Result*, *result_path: str*) → list[str] | None
Save a Result instance to a spec file (**NOT IMPLEMENTED**).
Parameters
- **result** (Result) – Result instance to save to specs file.
- **result_path** (*str*) – Path to write the result data to.

save_scheme(*scheme: Scheme*, *file_name: str*)
Save a Scheme instance to a spec file (**NOT IMPLEMENTED**).
Parameters
- **scheme** (Scheme) – Scheme instance to save to specs file.
- **file_name** (*str*) – File to write the scheme specs to.

prepare_dataset

Functions

Summary

add_svd_to_dataset	Add the SVD of a dataset inplace as Data variables to the dataset.
prepare_time_trace_dataset	Prepares a time trace for global analysis.

add_svd_to_dataset

glotaran.io.prepare_dataset.**add_svd_to_dataset**(*dataset: xr.Dataset*, *name: str = 'data'*, *lsv_dim: Hashable = 'time'*, *rsv_dim: Hashable = 'spectral'*, *data_array: xr.DataArray = None*)

Add the SVD of a dataset inplace as Data variables to the dataset.

The SVD is only computed if it doesn't already exist on the dataset.

Parameters

- **dataset** (*xr.Dataset*) – Dataset the SVD values should be added to.
- **name** (*str*) – Key to access the datarray inside of the dataset, by default "data"
- **lsv_dim** (*Hashable*) – Name of the dimension for the left singular value, by default "time"
- **rsv_dim** (*Hashable*) – Name of the dimension for the right singular value, by default "spectral"
- **data_array** (*xr.DataArray*) – Dataarray to calculate the SVD for, when provided the data extraction from the dataset will be skipped, by default None

prepare_time_trace_dataset

glotaran.io.prepare_dataset.**prepare_time_trace_dataset**(*dataset: xr.DataArray | xr.Dataset*, *weight: np.ndarray = None*, *irf: np.ndarray | xr.DataArray = None*) → xr.Dataset

Prepares a time trace for global analysis.

Parameters

- **dataset** – The dataset.
- **weight** – A weight for the dataset.
- **irf** – An IRF for the dataset.

15.1.7 model

Glotaran Model Package

This package contains the Glotaran's base model object, the model decorators and common model items.

Modules

glotaran.model.clp_penalties	This package contains compartment constraint items.
glotaran.model.constraint	This package contains compartment constraint items.
glotaran.model.dataset_group	
glotaran.model.dataset_model	The DatasetModel class.
glotaran.model.interval_property	Helper functions.
glotaran.model.item	The model item decorator.
glotaran.model.megacomplex(*[, dimension, ...])	The *@megacomplex* decorator is intended to be used on subclasses of `glotaran.model.Megacomplex`.
glotaran.model.model	A base class for global analysis models.
glotaran.model.property	This module holds the model property class.
glotaran.model.relation	Glotaran Relation
glotaran.model.util	Helper functions.
glotaran.model.weight	The Weight property class.

clp_penalties

This package contains compartment constraint items.

Functions

Summary

apply_spectral_penalties
has_spectral_penalties

apply_spectral_penalties

glotaran.model.clp_penalties.**apply_spectral_penalties**(*model: Model*, *parameters: ParameterGroup*, *clp_labels: dict[str, list[str] | list[list[str]]]*, *clps: dict[str, list[np.ndarray]]*, *matrices: dict[str, np.ndarray | list[np.ndarray]]*, *data: dict[str, xr.Dataset]*, *group_tolerance: float*) → np.ndarray

has_spectral_penalties

glotaran.model.clp_penalties.**has_spectral_penalties**(*model: Model*) → bool

Classes

Summary

EqualAreaPenalty	An equal area constraint adds a the differenc of the sum of a compartments in the e matrix in one ore more intervals to the scaled sum of the e matrix of one or more target compartments to residual.

EqualAreaPenalty

class glotaran.model.clp_penalties.**EqualAreaPenalty**

Bases: object

An equal area constraint adds a the differenc of the sum of a compartments in the e matrix in one ore more intervals to the scaled sum of the e matrix of one or more target compartments to residual. The additional residual is scaled with the weight.

Attributes Summary

parameter	ModelProperty is an extension of the property decorator.
source	ModelProperty is an extension of the property decorator.
source_intervals	ModelProperty is an extension of the property decorator.
target	ModelProperty is an extension of the property decorator.
target_intervals	ModelProperty is an extension of the property decorator.
weight	ModelProperty is an extension of the property decorator.

parameter

EqualAreaPenalty.**parameter**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

source

EqualAreaPenalty.**source**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

source_intervals

EqualAreaPenalty.**source_intervals**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

target

EqualAreaPenalty.**target**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

target_intervals

EqualAreaPenalty.**target_intervals**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

weight

EqualAreaPenalty.**weight**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

applies	Returns true if the index is in one of the intervals.
as_dict	
fill	
from_dict	
markdown	
validate	

applies

EqualAreaPenalty.**applies**(*index: Any*) → bool

Returns true if the index is in one of the intervals.

Parameters **index** –

Returns **applies**

Return type bool

as_dict

EqualAreaPenalty.**as_dict**() → dict

fill

EqualAreaPenalty.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

from_dict

classmethod EqualAreaPenalty.**from_dict**(*values: dict*) → cls

markdown

EqualAreaPenalty.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

validate

EqualAreaPenalty.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

applies(*index: Any*) → bool
Returns true if the index is in one of the intervals.
Parameters **index** –
Returns **applies**
Return type bool

as_dict() → dict

fill(*model: Model*, *parameters: ParameterGroup*) → cls

classmethod from_dict(*values: dict*) → cls

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

property parameter: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property source: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property source_intervals: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property target: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property target_intervals: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

property weight: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

constraint

This package contains compartment constraint items.

Classes

Summary

Constraint	A constraint is applied on one clp on one or many intervals on the estimated axis type.
OnlyConstraint	A only constraint sets the calculated matrix row of a compartment to 0 outside the given intervals.
ZeroConstraint	A zero constraint sets the calculated matrix row of a compartment to 0 in the given intervals.

Constraint

class glotaran.model.constraint.**Constraint**

Bases: object

A constraint is applied on one clp on one or many intervals on the estimated axis type.

There are two types: zero and equal. See the documentation of the respective classes for details.

Methods Summary

add_type
get_default_type

add_type

classmethod Constraint.**add_type**(*type_name: str*, *attribute_type: type*)

get_default_type

classmethod Constraint.**get_default_type**() → str

Methods Documentation

classmethod **add_type**(*type_name: str*, *attribute_type: type*)

classmethod **get_default_type**() → str

OnlyConstraint

class glotaran.model.constraint.**OnlyConstraint**

Bases: *glotaran.model.interval_property.IntervalProperty*

A only constraint sets the calculated matrix row of a compartment to 0 outside the given intervals.

Attributes Summary

interval	ModelProperty is an extension of the property decorator.
target	ModelProperty is an extension of the property decorator.

interval

OnlyConstraint.**interval**

ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

target

OnlyConstraint.**target**

ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

applies	Returns true if `value` is in one of the intervals.
as_dict	
fill	
from_dict	
markdown	
validate	

applies

`OnlyConstraint.`**`applies`**(*value: float*) → bool

Returns true if `value` is in one of the intervals.

Parameters **index** (*float*) –
Returns **applies**
Return type bool

as_dict

`OnlyConstraint.`**`as_dict`**() → dict

fill

`OnlyConstraint.`**`fill`**(*model: Model*, *parameters: ParameterGroup*) → cls

from_dict

`classmethod` `OnlyConstraint.`**`from_dict`**(*values: dict*) → cls

markdown

`OnlyConstraint.`**`markdown`**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

validate

OnlyConstraint.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

applies(*value: float*) → bool
Returns true if `value` is in one of the intervals.
Parameters **index** (*float*) –
Returns **applies**
Return type bool

as_dict() → dict

fill(*model: Model*, *parameters: ParameterGroup*) → cls

classmethod from_dict(*values: dict*) → cls

property interval: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

property target: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

ZeroConstraint

class glotaran.model.constraint.**ZeroConstraint**
Bases: *glotaran.model.interval_property.IntervalProperty*

A zero constraint sets the calculated matrix row of a compartment to 0 in the given intervals.

Attributes Summary

interval	ModelProperty is an extension of the property decorator.
target	ModelProperty is an extension of the property decorator.

interval

ZeroConstraint.**interval**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

target

ZeroConstraint.**target**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

applies	Returns true if `value` is in one of the intervals.
as_dict	
fill	
from_dict	
markdown	
validate	

applies

ZeroConstraint.**applies**(*value: float*) → bool
Returns true if `value` is in one of the intervals.
Parameters **value** (*float*) –
Returns **applies**
Return type bool

as_dict

ZeroConstraint.**as_dict**() → dict

fill

ZeroConstraint.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

from_dict

classmethod ZeroConstraint.**from_dict**(*values: dict*) → cls

markdown

ZeroConstraint.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

validate

ZeroConstraint.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

applies(*value: float*) → bool
Returns true if `value` is in one of the intervals.
Parameters **value** (*float*) –
Returns **applies**
Return type bool

as_dict() → dict

fill(*model: Model*, *parameters: ParameterGroup*) → cls

classmethod **from_dict**(*values: dict*) → cls

property **interval: model_property.glotaran_property_type**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

property target: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

dataset_group

Classes

Summary

DatasetGroup	
DatasetGroupModel	A group of datasets which will evaluated independently.

DatasetGroup

class glotaran.model.dataset_group.**DatasetGroup**(*model: 'DatasetGroupModel'*, *dataset_models: 'dict[str, DatasetModel]' = <factory>*)

Bases: object

Attributes Summary

model	
dataset_models	

model

DatasetGroup.**model: DatasetGroupModel**

dataset_models

DatasetGroup.**dataset_models: dict[str, DatasetModel]**

Methods Summary

—

Methods Documentation

dataset_models: dict[str, DatasetModel]

model: *DatasetGroupModel*

DatasetGroupModel

class glotaran.model.dataset_group.**DatasetGroupModel**(*residual_function: Literal['variable_projection', 'non_negative_least_squares'] = 'variable_projection'*, *link_clp: bool | None = None*)

Bases: object

A group of datasets which will evaluated independently.

Attributes Summary

link_clp	Whether to link the clp parameter.
residual_function	The residual function to use.

link_clp

DatasetGroupModel.**link_clp: bool | None = None**

Whether to link the clp parameter.

residual_function

DatasetGroupModel.**residual_function: Literal['variable_projection', 'non_negative_least_squares'] = 'variable_projection'**
The residual function to use.

Methods Summary

—

Methods Documentation

link_clp: bool | None = None
Whether to link the clp parameter.

residual_function: Literal['variable_projection', 'non_negative_least_squares'] = 'variable_projection'
The residual function to use.

dataset_model

The DatasetModel class.

Functions

Summary

create_dataset_model_type

create_dataset_model_type

glotaran.model.dataset_model.**create_dataset_model_type**(*properties: dict[str, Any]*) → type[DatasetModel]

Classes

Summary

DatasetModel	A *DatasetModel* describes a dataset in terms of a glotaran model.

DatasetModel

class glotaran.model.dataset_model.**DatasetModel**
Bases: object

A *DatasetModel* describes a dataset in terms of a glotaran model. It contains references to model items which describe the physical model for a given dataset.

A general dataset descriptor assigns one or more megacomplexes and a scale parameter.

Methods Summary

ensure_unique_megacomplexes	Ensure that unique megacomplexes Are only used once per dataset.
finalize_data	
get_coordinates	Gets the dataset model's coordinates.
get_data	Gets the dataset model's data.
get_global_axis	Gets the dataset model's global axis.
get_global_dimension	Returns the dataset model's global dimension.
get_model_axis	Gets the dataset model's model axis.
get_model_dimension	Returns the dataset model's model dimension.
get_weight	Gets the dataset model's weight.
has_global_model	Indicates if the dataset model can model the global dimension.
is_index_dependent	Indicates if the dataset model is index dependent.
iterate_global_megacomplexes	Iterates of der dataset model's global megacomplexes.
iterate_megacomplexes	Iterates of der dataset model's megacomplexes.
overwrite_global_dimension	Overwrites the dataset model's global dimension.
overwrite_index_dependent	Overrides the index dependency of the dataset
overwrite_model_dimension	Overwrites the dataset model's model dimension.
set_coordinates	Sets the dataset model's coordinates.
set_data	Sets the dataset model's data.
swap_dimensions	Swaps the dataset model's global and model dimension.

ensure_unique_megacomplexes

DatasetModel.**ensure_unique_megacomplexes**(*model: Model*) → list[str]
Ensure that unique megacomplexes Are only used once per dataset.
Parameters **model** (Model) – Model object using this dataset model.
Returns Error messages to be shown when the model gets validated.
Return type list[str]

finalize_data

DatasetModel.**finalize_data**(*dataset: xarray.core.dataset.Dataset*) → None

get_coordinates

DatasetModel.**get_coordinates**() → dict[Hashable, np.ndarray]
Gets the dataset model's coordinates.

get_data

DatasetModel.**get_data**() → numpy.ndarray
Gets the dataset model's data.

get_global_axis

DatasetModel.**get_global_axis**() → numpy.ndarray
Gets the dataset model's global axis.

get_global_dimension

DatasetModel.**get_global_dimension**() → str
Returns the dataset model's global dimension.

get_model_axis

DatasetModel.**get_model_axis**() → numpy.ndarray
Gets the dataset model's model axis.

get_model_dimension

DatasetModel.**get_model_dimension**() → str
Returns the dataset model's model dimension.

get_weight

DatasetModel.**get_weight**() → np.ndarray | None
Gets the dataset model's weight.

has_global_model

DatasetModel.**has_global_model**() → bool
Indicates if the dataset model can model the global dimension.

is_index_dependent

DatasetModel.**is_index_dependent**() → bool
Indicates if the dataset model is index dependent.

iterate_global_megacomplexes

DatasetModel.**iterate_global_megacomplexes**() → Generator[tuple[Parameter | int | None, Megacomplex | str], None, None]
Iterates of der dataset model's global megacomplexes.

iterate_megacomplexes

DatasetModel.**iterate_megacomplexes**() → Generator[tuple[Parameter | int | None, Megacomplex | str], None, None]
Iterates of der dataset model's megacomplexes.

overwrite_global_dimension

DatasetModel.**overwrite_global_dimension**(*global_dimension: str*) → None
Overwrites the dataset model's global dimension.

overwrite_index_dependent

DatasetModel.**overwrite_index_dependent**(*index_dependent: bool*)
Overrides the index dependency of the dataset

overwrite_model_dimension

DatasetModel.**overwrite_model_dimension**(*model_dimension: str*) → None
Overwrites the dataset model's model dimension.

set_coordinates

DatasetModel.**set_coordinates**(*coords: dict[str, np.ndarray]*)
Sets the dataset model's coordinates.

set_data

DatasetModel.**set_data**(*dataset: xarray.core.dataset.Dataset*) → *glotaran.model.dataset_model.DatasetModel*

Sets the dataset model's data.

swap_dimensions

DatasetModel.**swap_dimensions**() → None

Swaps the dataset model's global and model dimension.

Methods Documentation

ensure_unique_megacomplexes(*model: Model*) → list[str]

Ensure that unique megacomplexes Are only used once per dataset.

Parameters **model** (Model) – Model object using this dataset model.
Returns Error messages to be shown when the model gets validated.
Return type list[str]

finalize_data(*dataset: xarray.core.dataset.Dataset*) → None

get_coordinates() → dict[Hashable, np.ndarray]

Gets the dataset model's coordinates.

get_data() → numpy.ndarray

Gets the dataset model's data.

get_global_axis() → numpy.ndarray

Gets the dataset model's global axis.

get_global_dimension() → str

Returns the dataset model's global dimension.

get_model_axis() → numpy.ndarray

Gets the dataset model's model axis.

get_model_dimension() → str

Returns the dataset model's model dimension.

get_weight() → np.ndarray | None

Gets the dataset model's weight.

has_global_model() → bool

Indicates if the dataset model can model the global dimension.

is_index_dependent() → bool

Indicates if the dataset model is index dependent.

iterate_global_megacomplexes() → Generator[tuple[Parameter | int | None, Megacomplex | str], None, None]

Iterates of der dataset model's global megacomplexes.

iterate_megacomplexes() → Generator[tuple[Parameter | int | None, Megacomplex | str], None, None]

Iterates of der dataset model's megacomplexes.

overwrite_global_dimension(*global_dimension: str*) → None
Overwrites the dataset model's global dimension.

overwrite_index_dependent(*index_dependent: bool*)
Overrides the index dependency of the dataset

overwrite_model_dimension(*model_dimension: str*) → None
Overwrites the dataset model's model dimension.

set_coordinates(*coords: dict[str, np.ndarray]*)
Sets the dataset model's coordinates.

set_data(*dataset: xarray.core.dataset.Dataset*) → *glotaran.model.dataset_model.DatasetModel*
Sets the dataset model's data.

swap_dimensions() → None
Swaps the dataset model's global and model dimension.

interval_property

Helper functions.

Classes

Summary

IntervalProperty	Applies a relation between clps as

IntervalProperty

class glotaran.model.interval_property.**IntervalProperty**
Bases: object

Applies a relation between clps as

$source = parameter * target$.

Attributes Summary

interval	ModelProperty is an extension of the property decorator.

interval

IntervalProperty.**interval**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

applies	Returns true if `value` is in one of the intervals.
as_dict	
fill	
from_dict	
markdown	
validate	

applies

IntervalProperty.**applies**(*value: float*) → bool
Returns true if `value` is in one of the intervals.
Parameters **value** (*float*) –
Returns **applies**
Return type bool

as_dict

IntervalProperty.**as_dict**() → dict

fill

IntervalProperty.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

from_dict

classmethod IntervalProperty.**from_dict**(*values: dict*) → cls

markdown

IntervalProperty.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

validate

IntervalProperty.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

applies(*value: float*) → bool
Returns true if `value` is in one of the intervals.
Parameters **value** (*float*) –
Returns **applies**
Return type bool

as_dict() → dict

fill(*model: Model*, *parameters: ParameterGroup*) → cls

classmethod **from_dict**(*values: dict*) → cls

property **interval: model_property.glotaran_property_type**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

item

The model item decorator.

Functions

Summary

model_item	The @*model_item* decorator adds the given properties to the class.
model_item_typed	The model_item_typed decorator adds attributes to the class to enable the glotaran model parser to infer the correct class for an item when there are multiple variants.
model_item_validator	The model_item_validator marks a method of a model item as validation function

model_item

glotaran.model.item.**model_item**(*properties: None | dict[str, dict[str, Any]] = None*, *has_type: bool = False*, *has_label: bool = True*) → Callable

The @*model_item* decorator adds the given properties to the class. Further it adds classmethods for deserialization, validation and printing.

By default, a *label* property is added.

The *properties* dictionary contains the name of the properties as keys. The values must be either a *type* or dictionary with the following values:

- type: a *type* (required)
- doc: a string for documentation (optional)
- default: a default value (optional)
- allow_none: if *True*, the property can be set to None (optional)

Classes with the *model_item* decorator intended to be used in glotaran models.

Parameters

- **properties** – A dictionary of property names and options.
- **has_type** – If true, a type property will added. Used for model attributes, which can have more then one type.
- **has_label** – If false no label property will be added.

model_item_typed

glotaran.model.item.**model_item_typed**(*, *types: dict[str, Any]*, *has_label: bool = True*, *default_type: str = None*)

The model_item_typed decorator adds attributes to the class to enable the glotaran model parser to infer the correct class for an item when there are multiple variants.

Parameters

- **types** – A dictionary of types and options.
- **has_label** – If *False* no label property will be added.

model_item_validator

glotaran.model.item.**model_item_validator**(*need_parameter: bool*)

The model_item_validator marks a method of a model item as validation function

megacomplex

glotaran.model.**megacomplex**(*, *dimension: str | None = None*, *model_items: dict[str, dict[str, Any]] = None*, *properties: Any | dict[str, dict[str, Any]] = None*, *dataset_model_items: dict[str, dict[str, Any]] = None*, *dataset_properties: Any | dict[str, dict[str, Any]] = None*, *unique: bool = False*, *register_as: str | None = None*)

The *@megacomplex* decorator is intended to be used on subclasses of `glotaran.model.Megacomplex`. It registers the megacomplex model and makes it available in analysis models.

model

A base class for global analysis models.

Classes

Summary

Model	A base class for global analysis models.

Model

class glotaran.model.model.**Model**(*, *megacomplex_types: dict[str, type[Megacomplex]]*, *default_megacomplex_type: str | None = None*, *dataset_group_models: dict[str, DatasetGroupModel] = None*)

Bases: `object`

A base class for global analysis models.

Attributes Summary

dataset_group_models	

default_megacomplex	The default megacomplex used by this model.
global_dimension	Deprecated use `Scheme.global_dimensions['<dataset_name>']` instead
global_megacomplex	Alias for *glotaran.model.megacomplex*.
megacomplex_types	The megacomplex types used by this model.
model_dimension	Deprecated use `Scheme.model_dimensions['<dataset_name>']` instead
model_items	The model_items types used by this model.

dataset_group_models

Model.**dataset_group_models**

default_megacomplex

Model.**default_megacomplex**
The default megacomplex used by this model.

global_dimension

Model.**global_dimension**
Deprecated use `Scheme.global_dimensions['<dataset_name>']` instead

global_megacomplex

Model.**global_megacomplex**
Alias for *glotaran.model.megacomplex*. Needed internally.

megacomplex_types

Model.**megacomplex_types**
The megacomplex types used by this model.

model_dimension

Model.**model_dimension**
Deprecated use `Scheme.model_dimensions['<dataset_name>']` instead

model_items

Model.**model_items**
The model_items types used by this model.

Methods Summary

as_dict	
from_dict	Creates a model from a dictionary.
get_dataset_groups	
get_parameters	
is_groupable	
loader	Create a Model instance from the specs defined in a file.
markdown	Formats the model as Markdown string.
need_index_dependent	Returns true if e.g.
problem_list	Returns a list with all problems in the model and missing parameters if specified.
valid	Returns *True* if the number problems in the model is 0, else *False*
validate	Returns a string listing all problems in the model and missing parameters if specified.

as_dict

Model.**as_dict**() → dict

from_dict

classmethod Model.**from_dict**(*model_dict: dict[str, Any]*, *, *megacomplex_types: dict[str, type[Megacomplex]] | None = None*, *default_megacomplex_type: str | None = None*) → Model
Creates a model from a dictionary.

Parameters

- **model_dict** (*dict[str, Any]*) – Dictionary containing the model.
- **megacomplex_types** (*dict[str, type[Megacomplex]] | None*) – Overwrite 'megacomplex_types' in `model_dict` for testing.

- **default_megacomplex_type** (*str | None*) – Overwrite 'default_megacomplex' in `model_dict` for testing.

get_dataset_groups

`Model.`**`get_dataset_groups`**`()` → dict[str, DatasetGroup]

get_parameters

`Model.`**`get_parameters`**`()` → list[str]

is_groupable

`Model.`**`is_groupable`**`(`*parameters: ParameterGroup*, *data: dict[str, xr.DataArray]*`)` → bool

loader

`Model.`**`loader`**`(`*format_name: str = None*, ***kwargs: Any*`)` → Model

Create a Model instance from the specs defined in a file.

Parameters

- **file_name** (*StrOrPath*) – File containing the model specs.
- **format_name** (*str*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_model` implementation of the project io plugin.

Returns Model instance created from the file.

Return type *Model*

markdown

`Model.`**`markdown`**`(`*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*, *initial_parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*, *base_heading_level: int = 1*`)` → *glotaran.utils.ipython.MarkdownStr*

Formats the model as Markdown string.

Parameters will be included if specified.

Parameters

- **parameter** (ParameterGroup) – Parameter to include.
- **initial_parameters** (ParameterGroup) – Initial values for the parameters.
- **base_heading_level** (*int*) – Base heading level of the markdown sections.

  E.g.:
  - If it is 1 the string will start with '# Model'.
  - If it is 3 the string will start with '### Model'.

need_index_dependent

Model.**need_index_dependent**() → bool
Returns true if e.g. clp_relations with intervals are present.

problem_list

Model.**problem_list**(*parameters: ParameterGroup = None*) → list[str]
Returns a list with all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

valid

Model.**valid**(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*) → bool
Returns *True* if the number problems in the model is 0, else *False*

Parameters **parameter** – The parameter to validate.

validate

Model.**validate**(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*, *raise_exception: bool = False*) → str
Returns a string listing all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

Methods Documentation

as_dict() → dict

property dataset_group_models: dict[str, DatasetGroupModel]

property default_megacomplex: str
The default megacomplex used by this model.

classmethod from_dict(*model_dict: dict[str, Any]*, ***, *megacomplex_types: dict[str, type[Megacomplex]] | None = None*, *default_megacomplex_type: str | None = None*) → *Model*
Creates a model from a dictionary.

Parameters

- **model_dict** (*dict[str, Any]*) – Dictionary containing the model.
- **megacomplex_types** (*dict[str, type[Megacomplex]] | None*) – Overwrite 'megacomplex_types' in `model_dict` for testing.
- **default_megacomplex_type** (*str | None*) – Overwrite 'default_megacomplex' in `model_dict` for testing.

get_dataset_groups() → dict[str, DatasetGroup]

get_parameters() → list[str]

property global_dimension
Deprecated use `Scheme.global_dimensions['<dataset_name>']` instead

property global_megacomplex: dict[str, Megacomplex]
Alias for *glotaran.model.megacomplex*. Needed internally.

is_groupable(*parameters: ParameterGroup*, *data: dict[str, xr.DataArray]*) → bool

loader(*format_name: str = None*, ***kwargs: Any*) → *Model*
Create a Model instance from the specs defined in a file.

Parameters

- **file_name** (*StrOrPath*) – File containing the model specs.
- **format_name** (*str*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_model` implementation of the project io plugin.

Returns Model instance created from the file.

Return type *Model*

markdown(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*, *initial_parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*, *base_heading_level: int = 1*) → *glotaran.utils.ipython.MarkdownStr*
Formats the model as Markdown string.

Parameters will be included if specified.

Parameters

- **parameter** (ParameterGroup) – Parameter to include.
- **initial_parameters** (ParameterGroup) – Initial values for the parameters.
- **base_heading_level** (*int*) – Base heading level of the markdown sections.

  E.g.:

  – If it is 1 the string will start with '# Model'.

  – If it is 3 the string will start with '### Model'.

property megacomplex_types: dict[str, type[Megacomplex]]
The megacomplex types used by this model.

property model_dimension
Deprecated use `Scheme.model_dimensions['<dataset_name>']` instead

property model_items: dict[str, type[object]]
The model_items types used by this model.

need_index_dependent() → bool
Returns true if e.g. clp_relations with intervals are present.

problem_list(*parameters: ParameterGroup = None*) → list[str]
Returns a list with all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

valid(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*) → bool

Returns *True* if the number problems in the model is 0, else *False*

Parameters **parameter** – The parameter to validate.

validate(*parameters: Optional[*glotaran.parameter.parameter_group.ParameterGroup*] = None*, *raise_exception: bool = False*) → str

Returns a string listing all problems in the model and missing parameters if specified.

Parameters **parameter** – The parameter to validate.

property

This module holds the model property class.

Classes

Summary

ModelProperty	ModelProperty is an extension of the property decorator.

ModelProperty

class glotaran.model.property.**ModelProperty**(*cls: type*, *name: str*, *property_type: type*, *doc: str*, *default: Any*, *allow_none: bool*)

Bases: property

ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Create a new model property.

Parameters

- **cls** (*type*) – The class the property is being attached to.
- **name** (*str*) – The name of the property.
- **property_type** (*type*) – The type of the property.
- **doc** (*str*) – A documentation string of for the property.
- **default** (*Any*) – The default value of the property.
- **allow_none** (*bool*) – Whether the property is allowed to be None.

Attributes Summary

fdel	
fget	
fset	
glotaran_allow_none	Check if the property is allowed to be None.
glotaran_is_mapping_property	Check if the type is mapping.
glotaran_is_parameter_property	Check if the subtype is parameter.
glotaran_is_scalar_property	Check if the type is scalar.
glotaran_is_sequence_property	Check if the type is a sequence.
glotaran_property_subtype	Get the subscribed type.
glotaran_property_type	Get the type of the property.

fdel

ModelProperty.**fdel**

fget

ModelProperty.**fget**

fset

ModelProperty.**fset**

glotaran_allow_none

ModelProperty.**glotaran_allow_none**
Check if the property is allowed to be None.

Returns Whether the property is allowed to be None.

Return type bool

glotaran_is_mapping_property

ModelProperty.**glotaran_is_mapping_property**
Check if the type is mapping.

Returns Whether the type is a mapping.

Return type bool

glotaran_is_parameter_property

ModelProperty.**glotaran_is_parameter_property**
Check if the subtype is parameter.

Returns Whether the subtype is parameter.

Return type bool

glotaran_is_scalar_property

ModelProperty.**glotaran_is_scalar_property**
Check if the type is scalar.

Scalar means the type is neither a sequence nor a mapping.

Returns Whether the type is scalar.

Return type bool

glotaran_is_sequence_property

ModelProperty.**glotaran_is_sequence_property**
Check if the type is a sequence.

Returns Whether the type is a sequence.

Return type bool

glotaran_property_subtype

ModelProperty.**glotaran_property_subtype**
Get the subscribed type.

If the type is scalar, the type itself will be returned. If the type is a mapping, the value type will be returned.

Returns The subscribed type.

Return type type

glotaran_property_type

ModelProperty.**glotaran_property_type**
Get the type of the property.

Returns The type of the property.

Return type type

Methods Summary

deleter	Descriptor to change the deleter on a property.
getter	Descriptor to change the getter on a property.
glotaran_fill	Fill a property with items from a model and parameters.
glotaran_format_value	Format a value to string.
glotaran_replace_parameter_with_label	Replace parameter values with their full label.
glotaran_validate	Validate a value against a model and optionally against parameters.
glotaran_value_as_markdown	Get a markdown representation of the property.
setter	Descriptor to change the setter on a property.

deleter

ModelProperty.**deleter**()
Descriptor to change the deleter on a property.

getter

ModelProperty.**getter**()
Descriptor to change the getter on a property.

glotaran_fill

ModelProperty.**glotaran_fill**(*value: Any*, *model: Model*, *parameter: ParameterGroup*) → Any
Fill a property with items from a model and parameters.

This replaces model item labels with the actual items and sets the parameter values.

Parameters

- **value** (*Any*) – The property value.
- **model** (Model) – The model to fill in.
- **parameter** (ParameterGroup) – The parameters to fill in.

Returns The filled value.

Return type Any

glotaran_format_value

ModelProperty.**glotaran_format_value**(*value: Any*, *all_parameters: ParameterGroup | None = None*, *initial_parameters: ParameterGroup | None = None*) → str

Format a value to string.

Parameters

- **value** (*Any*) – The value to format.
- **all_parameters** (ParameterGroup | *None*) – A parameter group containing the whole parameter set (used for expression lookup).
- **initial_parameters** (ParameterGroup | *None*) – The initial parameter.

Returns The formatted value.

Return type str

glotaran_replace_parameter_with_labels

ModelProperty.**glotaran_replace_parameter_with_labels**(*value: Any*) → Any

Replace parameter values with their full label.

A convenience function for serialization.

Parameters **value** (*Any*) – The value to replace.

Returns The value with parameters replaced by their labels.

Return type Any

glotaran_validate

ModelProperty.**glotaran_validate**(*value: Any*, *model: Model*, *parameters: ParameterGroup = None*) → list[str]

Validate a value against a model and optionally against parameters.

Parameters

- **value** (*Any*) – The value to validate.
- **model** (Model) – The model to validate against.
- **parameters** (ParameterGroup) – The parameters to validate against.

Returns A list of human readable list of messages of problems.

Return type list[str]

glotaran_value_as_markdown

ModelProperty.**glotaran_value_as_markdown**(*value: Any*, *all_parameters: ParameterGroup | None = None*, *initial_parameters: ParameterGroup | None = None*) → MarkdownStr

Get a markdown representation of the property.

Parameters

- **value** (*Any*) – The property value.
- **all_parameters** (ParameterGroup | None) – A parameter group containing the whole parameter set (used for expression lookup).
- **initial_parameters** (ParameterGroup | None) – The initial parameter.

Returns The property as markdown string.

Return type *MarkdownStr*

setter

ModelProperty.**setter**()

Descriptor to change the setter on a property.

Methods Documentation

deleter()

Descriptor to change the deleter on a property.

fdel

fget

fset

getter()

Descriptor to change the getter on a property.

property glotaran_allow_none: bool

Check if the property is allowed to be None.

Returns Whether the property is allowed to be None.

Return type bool

glotaran_fill(*value: Any*, *model: Model*, *parameter: ParameterGroup*) → Any

Fill a property with items from a model and parameters.

This replaces model item labels with the actual items and sets the parameter values.

Parameters

- **value** (*Any*) – The property value.
- **model** (Model) – The model to fill in.
- **parameter** (ParameterGroup) – The parameters to fill in.

Returns The filled value.

Return type Any

glotaran_format_value(*value: Any*, *all_parameters: ParameterGroup | None = None*, *initial_parameters: ParameterGroup | None = None*) → str

Format a value to string.

Parameters

- **value** (*Any*) – The value to format.
- **all_parameters** (ParameterGroup | *None*) – A parameter group containing the whole parameter set (used for expression lookup).
- **initial_parameters** (ParameterGroup | *None*) – The initial parameter.

Returns The formatted value.

Return type str

property glotaran_is_mapping_property: bool

Check if the type is mapping.

Returns Whether the type is a mapping.

Return type bool

property glotaran_is_parameter_property: bool

Check if the subtype is parameter.

Returns Whether the subtype is parameter.

Return type bool

property glotaran_is_scalar_property: bool

Check if the type is scalar.

Scalar means the type is neither a sequence nor a mapping.

Returns Whether the type is scalar.

Return type bool

property glotaran_is_sequence_property: bool

Check if the type is a sequence.

Returns Whether the type is a sequence.

Return type bool

property glotaran_property_subtype: type

Get the subscribed type.

If the type is scalar, the type itself will be returned. If the type is a mapping, the value type will be returned.

Returns The subscribed type.

Return type type

property glotaran_property_type: type

Get the type of the property.

Returns The type of the property.

Return type type

glotaran_replace_parameter_with_labels(*value: Any*) → Any
Replace parameter values with their full label.

A convenience function for serialization.

Parameters **value** (*Any*) – The value to replace.

Returns The value with parameters replaced by their labels.

Return type Any

glotaran_validate(*value: Any*, *model: Model*, *parameters: ParameterGroup = None*) → list[str]
Validate a value against a model and optionally against parameters.

Parameters

- **value** (*Any*) – The value to validate.
- **model** (Model) – The model to validate against.
- **parameters** (ParameterGroup) – The parameters to validate against.

Returns A list of human readable list of messages of problems.

Return type list[str]

glotaran_value_as_markdown(*value: Any*, *all_parameters: ParameterGroup | None = None*, *initial_parameters: ParameterGroup | None = None*) → MarkdownStr
Get a markdown representation of the property.

Parameters

- **value** (*Any*) – The property value.
- **all_parameters** (ParameterGroup | *None*) – A parameter group containing the whole parameter set (used for expression lookup).
- **initial_parameters** (ParameterGroup | *None*) – The initial parameter.

Returns The property as markdown string.

Return type *MarkdownStr*

setter()
Descriptor to change the setter on a property.

relation

Glotaran Relation

Classes

Summary

Relation	Applies a relation between clps as

Relation

class glotaran.model.relation.**Relation**

Bases: *glotaran.model.interval_property.IntervalProperty*

Applies a relation between clps as

$target = parameter * source$.

Attributes Summary

interval	ModelProperty is an extension of the property decorator.
parameter	ModelProperty is an extension of the property decorator.
source	ModelProperty is an extension of the property decorator.
target	ModelProperty is an extension of the property decorator.

interval

Relation.**interval**

ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

parameter

Relation.**parameter**

ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

source

Relation.**source**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

target

Relation.**target**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

applies	Returns true if `value` is in one of the intervals.
as_dict	
fill	
from_dict	
markdown	
validate	

applies

Relation.**applies**(*value: float*) → bool
Returns true if `value` is in one of the intervals.

Parameters **value** (*float*) –

Returns **applies**

Return type bool

as_dict

Relation.**as_dict**() → dict

fill

Relation.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

from_dict

classmethod Relation.**from_dict**(*values: dict*) → cls

markdown

Relation.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

validate

Relation.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

applies(*value: float*) → bool
Returns true if `value` is in one of the intervals.

Parameters **value** (*float*) –

Returns **applies**

Return type bool

as_dict() → dict

fill(*model: Model*, *parameters: ParameterGroup*) → cls

classmethod **from_dict**(*values: dict*) → cls

property **interval: model_property.glotaran_property_type**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

property parameter: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property source: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

property target: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

util

Helper functions.

Functions

Summary

get_subtype	Gets the subscribed type of a generic type.
is_mapping_type	Check if the type is mapping.
is_scalar_type	Check if the type is scalar.
is_sequence_type	Check if the type is a sequence.
wrap_func_as_method	A decorator to wrap a function as class method.

get_subtype

glotaran.model.util.**get_subtype**(*t: type*) → type
Gets the subscribed type of a generic type.

If the type is scalar, the type itself will be returned. If the type is a mapping, the value type will be returned.

Parameters **t** (*type*) – The origin type.

Returns The subscribed type.

Return type type

is_mapping_type

glotaran.model.util.**is_mapping_type**(*t: type*) → bool
Check if the type is mapping.

Parameters **t** (*type*) – The type to check.

Returns Whether the type is a mapping.

Return type bool

is_scalar_type

glotaran.model.util.**is_scalar_type**(*t: type*) → bool
Check if the type is scalar.

Scalar means the type is neither a sequence nor a mapping.

Parameters **t** (*type*) – The type to check.

Returns Whether the type is scalar.

Return type bool

is_sequence_type

glotaran.model.util.**is_sequence_type**(*t: type*) → bool
Check if the type is a sequence.

Parameters **t** (*type*) – The type to check.

Returns Whether the type is a sequence.

Return type bool

wrap_func_as_method

glotaran.model.util.**wrap_func_as_method**(*cls: Any*, *name: str = None*, *annotations: dict[str, type] = None*, *doc: str = None*) → Callable[[DecoratedFunc], DecoratedFunc]
A decorator to wrap a function as class method.

Notes

Only for internal use.

Parameters

- **cls** – The class in which the function will be wrapped.
- **name** – The name of method. If *None*, the original function's name is used.
- **annotations** – The annotations of the method. If *None*, the original function's annotations are used.
- **doc** – The documentation of the method. If *None*, the original function's documentation is used.

Exceptions

Exception Summary

ModelError	Raised when a model contains errors.

ModelError

exception glotaran.model.util.**ModelError**(*error: str*)
Raised when a model contains errors.

weight

The Weight property class.

Classes

Summary

Weight	The *Weight* class describes a value by which a dataset will scaled.

Weight

class glotaran.model.weight.**Weight**
Bases: object

The *Weight* class describes a value by which a dataset will scaled.

global_interval and *model_interval* are optional. The whole range of the dataset will be used if not set.

Attributes Summary

datasets	ModelProperty is an extension of the property decorator.
global_interval	ModelProperty is an extension of the property decorator.
model_interval	ModelProperty is an extension of the property decorator.
value	ModelProperty is an extension of the property decorator.

datasets

Weight.**datasets**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

global_interval

Weight.**global_interval**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

model_interval

Weight.**model_interval**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

value

Weight.**value**
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

Methods Summary

as_dict
fill
from_dict
markdown
validate

as_dict

Weight.**as_dict**() → dict

fill

Weight.**fill**(*model: Model*, *parameters: ParameterGroup*) → cls

from_dict

classmethod Weight.**from_dict**(*values: dict*) → cls

markdown

Weight.**markdown**(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

validate

Weight.**validate**(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

Methods Documentation

as_dict() → dict

property datasets: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

fill(*model: Model*, *parameters: ParameterGroup*) → cls

classmethod from_dict(*values: dict*) → cls

property global_interval: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

markdown(*all_parameters: ParameterGroup = None*, *initial_parameters: ParameterGroup = None*) → MarkdownStr

property model_interval: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

validate(*model: Model*, *parameters: ParameterGroup | None = None*) → list[str]

property value: model_property.glotaran_property_type
ModelProperty is an extension of the property decorator.

It adds convenience functions for meta programming model items.

15.1.8 parameter

The glotaran parameter package.

Modules

glotaran.parameter.parameter	The parameter class.
glotaran.parameter.parameter_group	The parameter group class.
glotaran.parameter.parameter_history	The glotaran parameter history package.

parameter

The parameter class.

Classes

Summary

Keys	Keys for parameter options.
Parameter	A parameter for optimization.

Keys

class glotaran.parameter.parameter.**Keys**
Bases: object

Keys for parameter options.

Attributes Summary

EXPR
MAX
MIN
NON_NEG
STD_ERR
VARY

EXPR

Keys.**EXPR** = **'expr'**

MAX

Keys.**MAX** = **'max'**

MIN

Keys.**MIN** = **'min'**

NON_NEG

Keys.**NON_NEG** = **'non-negative'**

STD_ERR

Keys.**STD_ERR** = **'standard-error'**

VARY

Keys.**VARY** = **'vary'**

Methods Summary

—

Methods Documentation

EXPR = 'expr'

MAX = 'max'

MIN = 'min'

NON_NEG = 'non-negative'

STD_ERR = 'standard-error'

VARY = 'vary'

Parameter

class glotaran.parameter.parameter.**Parameter**(*label: str = None*, *full_label: str = None*, *expression: str | None = None*, *maximum: float = inf*, *minimum: float = - inf*, *non_negative: bool = False*, *standard_error: float = nan*, *value: float = nan*, *vary: bool = True*)

Bases: numpy.typing._array_like._SupportsArray

A parameter for optimization.

Optimization Parameter supporting numpy array operations.

Parameters

- **label** (*str*) – The label of the parameter., by default None
- **full_label** (*str*) – The label of the parameter with its path in a parameter group prepended. , by default None
- **expression** (*str | None*) – Expression to calculate the parameters value from, e.g. if used in relation to another parameter. , by default None
- **maximum** (*float*) – Upper boundary for the parameter to be varied to., by default np.inf
- **minimum** (*float*) – Lower boundary for the parameter to be varied to., by default -np.inf
- **non_negative** (*bool*) – Whether the parameter should always be bigger than zero., by default False
- **standard_error** (*float*) – The standard error of the parameter. , by default np.nan
- **value** (*float*) – Value of the parameter, by default np.nan
- **vary** (*bool*) – Whether the parameter should be changed during optimization or not. , by default True

Attributes Summary

expression	Expression to calculate the parameters value from.
full_label	Label of the parameter with its path in a parameter group prepended.
label	Label of the parameter.
maximum	Upper bound of the parameter.
minimum	Lower bound of the parameter.
non_negative	Indicate if the parameter is non-negative.
standard_error	Standard error of the optimized parameter.
transformed_expression	Expression of the parameter transformed for evaluation within a *ParameterGroup*.
value	Value of the parameter.
vary	Indicate if the parameter should be optimized.

expression

Parameter.**expression**

Expression to calculate the parameters value from.

This can used to set a relation to another parameter.

Returns The expression.

Return type str | None

full_label

Parameter.**full_label**

Label of the parameter with its path in a parameter group prepended.

Returns The full label.

Return type str

label

Parameter.**label**

Label of the parameter.

Returns The label.

Return type str

maximum

Parameter.**maximum**
Upper bound of the parameter.

Returns The upper bound of the parameter.

Return type float

minimum

Parameter.**minimum**
Lower bound of the parameter.

Returns The lower bound of the parameter.

Return type float

non_negative

Parameter.**non_negative**
Indicate if the parameter is non-negative.

If true, the parameter will be transformed with $p' = \log p$ and $p = \exp p'$.

Notes

Always *False* if *expression* is not *None*.

Returns Whether the parameter is non-negative.

Return type bool

standard_error

Parameter.**standard_error**
Standard error of the optimized parameter.

Returns The standard error of the parameter.

Return type float

transformed_expression

Parameter.**transformed_expression**
Expression of the parameter transformed for evaluation within a *ParameterGroup*.

Returns The transformed expression.

Return type str | None

value

Parameter.**value**
Value of the parameter.

Returns The value of the parameter.

Return type float

vary

Parameter.**vary**
Indicate if the parameter should be optimized.

Notes

Always *False* if *expression* is not *None*.

Returns Whether the parameter should be optimized.

Return type bool

Methods Summary

as_dict	Create a dictionary containing the parameter properties.
from_dict	Create a *Parameter* from a dictionary.
from_list_or_value	Create a parameter from a list or numeric value.
get_value_and_bounds_for_optimization	Get the parameter value and bounds with expression and non-negative constraints applied.
markdown	Get a markdown representation of the parameter.
set_from_group	Set all values of the parameter to the values of the corresponding parameter in the group.
set_value_from_optimization	Set the value from an optimization result and reverses non-negative transformation.
valid_label	Check if a label is a valid label for *Parameter*.

as_dict

Parameter.**as_dict**(*as_optimized: bool = True*) → dict[str, Any]
Create a dictionary containing the parameter properties.

from_dict

classmethod Parameter.**from_dict**(*parameter_dict: dict[str, Any]*) → Parameter

Create a *Parameter* from a dictionary.

Expects a dictionary created by **:method:`Parameter.as_dict`**.

Parameters **parameter_dict** (*dict[str, Any]*) – The source dictionary.

Returns The created *Parameter*

Return type *Parameter*

from_list_or_value

classmethod Parameter.**from_list_or_value**(*value: int | float | list*, *default_options: dict = None*, *label: str = None*) → Parameter

Create a parameter from a list or numeric value.

Parameters

- **value** (*int | float | list*) – The list or numeric value.
- **default_options** (*dict*) – A dictionary of default options.
- **label** (*str*) – The label of the parameter.

Returns The created *Parameter*.

Return type *Parameter*

get_value_and_bounds_for_optimization

Parameter.**get_value_and_bounds_for_optimization**() → tuple[float, float, float]

Get the parameter value and bounds with expression and non-negative constraints applied.

Returns A tuple containing the value, the lower and the upper bound.

Return type tuple[float, float, float]

markdown

Parameter.**markdown**(*all_parameters: ParameterGroup | None = None*, *initial_parameters: ParameterGroup | None = None*) → MarkdownStr

Get a markdown representation of the parameter.

Parameters

- **all_parameters** (*ParameterGroup | None*) – A parameter group containing the whole parameter set (used for expression lookup).
- **initial_parameters** (*ParameterGroup | None*) – The initial parameter.

Returns The parameter as markdown string.

Return type *MarkdownStr*

set_from_group

Parameter.**set_from_group**(*group: ParameterGroup*)
Set all values of the parameter to the values of the corresponding parameter in the group.

Notes

For internal use.

Parameters **group** (ParameterGroup) – The glotaran.parameter.ParameterGroup.

set_value_from_optimization

Parameter.**set_value_from_optimization**(*value: float*)
Set the value from an optimization result and reverses non-negative transformation.

Parameters **value** (*float*) – Value from optimization.

valid_label

static Parameter.**valid_label**(*label: str*) → bool
Check if a label is a valid label for *Parameter*.

Parameters **label** (*str*) – The label to validate.

Returns Whether the label is valid.

Return type bool

Methods Documentation

as_dict(*as_optimized: bool = True*) → dict[str, Any]
Create a dictionary containing the parameter properties.

property expression: str | None
Expression to calculate the parameters value from.

This can used to set a relation to another parameter.

Returns The expression.

Return type str | None

classmethod from_dict(*parameter_dict: dict[str, Any]*) → *Parameter*
Create a *Parameter* from a dictionary.

Expects a dictionary created by **:method:`Parameter.as_dict`**.

Parameters **parameter_dict** (*dict[str, Any]*) – The source dictionary.

Returns The created *Parameter*

Return type *Parameter*

classmethod from_list_or_value(*value: int | float | list*, *default_options: dict = None*, *label: str = None*) → *Parameter*
Create a parameter from a list or numeric value.

Parameters

- **value** (*int | float | list*) – The list or numeric value.
- **default_options** (*dict*) – A dictionary of default options.
- **label** (*str*) – The label of the parameter.

Returns The created *Parameter*.

Return type *Parameter*

property full_label: str
Label of the parameter with its path in a parameter group prepended.

Returns The full label.

Return type str

get_value_and_bounds_for_optimization() → tuple[float, float, float]
Get the parameter value and bounds with expression and non-negative constraints applied.

Returns A tuple containing the value, the lower and the upper bound.

Return type tuple[float, float, float]

property label: str | None
Label of the parameter.

Returns The label.

Return type str

markdown(*all_parameters: ParameterGroup | None = None*, *initial_parameters: ParameterGroup | None = None*) → MarkdownStr
Get a markdown representation of the parameter.

Parameters

- **all_parameters** (*ParameterGroup | None*) – A parameter group containing the whole parameter set (used for expression lookup).
- **initial_parameters** (*ParameterGroup | None*) – The initial parameter.

Returns The parameter as markdown string.

Return type *MarkdownStr*

property maximum: float
Upper bound of the parameter.

Returns The upper bound of the parameter.

Return type float

property minimum: float
Lower bound of the parameter.

Returns The lower bound of the parameter.

Return type float

property non_negative: bool
Indicate if the parameter is non-negative.

If true, the parameter will be transformed with $p' = \log p$ and $p = \exp p'$.

Notes

Always *False* if *expression* is not *None*.

Returns Whether the parameter is non-negative.

Return type bool

set_from_group(*group: ParameterGroup*)
Set all values of the parameter to the values of the corresponding parameter in the group.

Notes

For internal use.

Parameters **group** (ParameterGroup) – The glotaran.parameter.ParameterGroup.

set_value_from_optimization(*value: float*)
Set the value from an optimization result and reverses non-negative transformation.

Parameters **value** (*float*) – Value from optimization.

property standard_error: float
Standard error of the optimized parameter.

Returns The standard error of the parameter.

Return type float

property transformed_expression: str | None
Expression of the parameter transformed for evaluation within a *ParameterGroup*.

Returns The transformed expression.

Return type str | None

static valid_label(*label: str*) → bool
Check if a label is a valid label for *Parameter*.

Parameters **label** (*str*) – The label to validate.

Returns Whether the label is valid.

Return type bool

property value: float
Value of the parameter.

Returns The value of the parameter.

Return type float

property vary: bool
Indicate if the parameter should be optimized.

Notes

Always *False* if *expression* is not *None*.

Returns Whether the parameter should be optimized.

Return type bool

parameter_group

The parameter group class.

Classes

Summary

ParameterGroup	Represents are group of parameters.

ParameterGroup

class glotaran.parameter.parameter_group.**ParameterGroup**(*label: Optional[str] = None*, *root_group: Optional[*glotaran.parameter.parameter_group.ParameterG *= None*)

Bases: dict

Represents are group of parameters.

Can contain other groups, creating a tree-like hierarchy.

Initialize a *ParameterGroup* instance with `label`.

Parameters

- **label** (*str*) – The label of the group.
- **root_group** (ParameterGroup) – The root group

Raises ValueError – Raised if the an invalid label is given.

Attributes Summary

label	Label of the group.
root_group	Root of the group.

label

ParameterGroup.**label**
Label of the group.

Returns The label of the group.

Return type str

root_group

ParameterGroup.**root_group**
Root of the group.

Returns The root group.

Return type *ParameterGroup*

Methods Summary

add_group	Add a *ParameterGroup* to the group.
add_parameter	Add a `Parameter` to the group.
all	Iterate over all parameter in the group and it's subgroups together with their labels.
clear	
copy	Create a copy of the *ParameterGroup*.
from_dataframe	Create a *ParameterGroup* from a `pandas.DataFrame`.
from_dict	Create a *ParameterGroup* from a dictionary.
from_list	Create a *ParameterGroup* from a list.
from_parameter_dict_list	Create a *ParameterGroup* from a list of parameter dictionaries.
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Get a `Parameter` by its label.
get_group_for_parameter_by_label	Get the group for a parameter by it's label.
get_label_value_and_bounds_arrays	Return a arrays of all parameter labels, values and bounds.
get_nr_roots	Return the number of roots of the group.
groups	Return a generator over all groups and their subgroups.
has	Check if a parameter with the given label is in the group or in a subgroup.
items	
keys	
loader	Create a *ParameterGroup* instance from the specs defined in a file.
markdown	Format the *ParameterGroup* as markdown string.

continues on next page

Table 173 – continued from previous page

pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
set_from_history	Update the *ParameterGroup* with values from a parameter history.
set_from_label_and_value_arrays	Update the parameter values from a list of labels and values.
setdefault	Insert key with a value of default if key is not in the dictionary.
to_csv	Save a *ParameterGroup* to a CSV file.
to_dataframe	Create a pandas data frame from the group.
to_parameter_dict_list	Create list of parameter dictionaries from the group.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
update_parameter_expression	Update all parameters which have an expression.
values	

add_group

ParameterGroup.**add_group**(*group:* glotaran.parameter.parameter_group.ParameterGroup)
Add a *ParameterGroup* to the group.

Parameters **group** (ParameterGroup) – The group to add.

Raises **TypeError** – Raised if the group is not an instance of *ParameterGroup*.

add_parameter

ParameterGroup.**add_parameter**(*parameter: Parameter | list[Parameter]*)
Add a Parameter to the group.

Parameters **parameter** (Parameter | *list*[Parameter]) – The parameter to add.

Raises **TypeError** – If parameter or any item of it is not an instance of Parameter.

all

ParameterGroup.**all**(*root: str | None = None*, *separator: str = '.'*) → Generator[tuple[str, Parameter], None, None]

Iterate over all parameter in the group and it's subgroups together with their labels.

Parameters

- **root** (*str*) – The label of the root group
- **separator** (*str*) – The separator for the parameter labels.

Yields *tuple[str, Parameter]* – A tuple containing the full label of the parameter and the parameter itself.

clear

ParameterGroup.**clear**() → None. Remove all items from D.

copy

ParameterGroup.**copy**() → *glotaran.parameter.parameter_group.ParameterGroup*

Create a copy of the *ParameterGroup*.

Returns A copy of the *ParameterGroup*.

Return type *ParameterGroup*

from_dataframe

classmethod ParameterGroup.**from_dataframe**(*df: pandas.core.frame.DataFrame*, *source: str = 'DataFrame'*) → *glotaran.parameter.parameter_group.ParameterGroup*

Create a *ParameterGroup* from a `pandas.DataFrame`.

Parameters

- **df** (*pd.DataFrame*) – The source data frame.
- **source** (*str*) – Optional name of the source file, used for error messages.

Returns The created parameter group.

Return type *ParameterGroup*

Raises **ValueError** – Raised if the columns 'label' or 'value' doesn't exist. Also raised if the columns 'minimum', 'maximum' or 'values' contain non numeric values or if the columns 'non-negative' or 'vary' are no boolean.

from_dict

classmethod ParameterGroup.**from_dict**(*parameter_dict: dict[str, dict[str, Any] | list[float | list[Any]]]*, *label: str = None*, *root_group: ParameterGroup = None*) → ParameterGroup

Create a *ParameterGroup* from a dictionary.

Parameters

- **parameter_dict** (*dict[str, dict | list]*) – A parameter dictionary containing parameters.
- **label** (*str*) – The label of the group.
- **root_group** (ParameterGroup) – The root group

Returns The created *ParameterGroup*

Return type *ParameterGroup*

from_list

classmethod ParameterGroup.**from_list**(*parameter_list: list[float | list[Any]]*, *label: str = None*, *root_group: ParameterGroup = None*) → ParameterGroup

Create a *ParameterGroup* from a list.

Parameters

- **parameter_list** (*list[float | list[Any]]*) – A parameter list containing parameters
- **label** (*str*) – The label of the group.
- **root_group** (ParameterGroup) – The root group

Returns The created *ParameterGroup*.

Return type *ParameterGroup*

from_parameter_dict_list

classmethod ParameterGroup.**from_parameter_dict_list**(*parameter_dict_list: list[dict[str, Any]]*) → ParameterGroup

Create a *ParameterGroup* from a list of parameter dictionaries.

Parameters **parameter_dict_list** (*list[dict[str, Any]]*) – A list of parameter dictionaries.

Returns The created *ParameterGroup*.

Return type *ParameterGroup*

fromkeys

ParameterGroup.**fromkeys**(*iterable*, *value=None*, */*)
Create a new dictionary with keys from iterable and values set to value.

get

ParameterGroup.**get**(*label: str*) → *glotaran.parameter.parameter.Parameter*
Get a Parameter by its label.

Parameters **label** (*str*) – The label of the parameter, with its path in a *ParameterGroup* prepended.

Returns The parameter.

Return type *Parameter*

Raises **ParameterNotFoundException** – Raised if no parameter with the given label exists.

get_group_for_parameter_by_label

ParameterGroup.**get_group_for_parameter_by_label**(*parameter_label: str*, *create_if_not_exist: bool = False*) → *glotaran.parameter.parameter_group.ParameterGroup*
Get the group for a parameter by it's label.

Parameters

- **parameter_label** (*str*) – The parameter label.
- **create_if_not_exist** (*bool*) – Create the parameter group if not existent.

Returns The group of the parameter.

Return type *ParameterGroup*

Raises **KeyError** – Raised if the group does not exist and *create_if_not_exist* is *False*.

get_label_value_and_bounds_arrays

ParameterGroup.**get_label_value_and_bounds_arrays**(*exclude_non_vary: bool = False*) → tuple[list[str], np.ndarray, np.ndarray, np.ndarray]
Return a arrays of all parameter labels, values and bounds.

Parameters **exclude_non_vary** (*bool*) – If true, parameters with *vary=False* are excluded.

Returns A tuple containing a list of parameter labels and an array of the values, lower and upper bounds.

Return type tuple[list[str], np.ndarray, np.ndarray, np.ndarray]

get_nr_roots

ParameterGroup.**get_nr_roots**() → int
Return the number of roots of the group.

Returns The number of roots.

Return type int

groups

ParameterGroup.**groups**() →
Generator[*glotaran.parameter.parameter_group.ParameterGroup*, None, None]
Return a generator over all groups and their subgroups.

Yields *ParameterGroup* – A subgroup of `ParameterGroup`.

has

ParameterGroup.**has**(*label: str*) → bool
Check if a parameter with the given label is in the group or in a subgroup.

Parameters **label** (*str*) – The label of the parameter, with its path in a `ParameterGroup` prepended.

Returns Whether a parameter with the given label exists in the group.

Return type bool

items

ParameterGroup.**items**() → a set-like object providing a view on D's items

keys

ParameterGroup.**keys**() → a set-like object providing a view on D's keys

loader

ParameterGroup.**loader**(*format_name: str = None*, ***kwargs*) → ParameterGroup
Create a `ParameterGroup` instance from the specs defined in a file.

Parameters

- **file_name** (*StrOrPath*) – File containing the parameter specs.
- **format_name** (*str*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_parameters` implementation of the project io plugin.

Returns *ParameterGroup* instance created from the file.

Return type *ParameterGroup*

markdown

ParameterGroup.**markdown**(*float_format: str = '.3e'*) → *glotaran.utils.ipython.MarkdownStr*
Format the *ParameterGroup* as markdown string.

This is done by recursing the nested *ParameterGroup* tree.

Parameters **float_format** (*str*) – Format string for floating point numbers, by default ".3e"

Returns The markdown representation as string.

Return type *MarkdownStr*

pop

ParameterGroup.**pop**(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem

ParameterGroup.**popitem**(*/*)
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

set_from_history

ParameterGroup.**set_from_history**(*history: ParameterHistory*, *index: int*)
Update the *ParameterGroup* with values from a parameter history.

Parameters

- **history** (*ParameterHistory*) – The parameter history.
- **index** (*int*) – The history index.

set_from_label_and_value_arrays

ParameterGroup.**set_from_label_and_value_arrays**(*labels: list[str]*, *values: np.ndarray*)
Update the parameter values from a list of labels and values.

Parameters

- **labels** (*list[str]*) – A list of parameter labels.
- **values** (*np.ndarray*) – An array of parameter values.

Raises **ValueError** – Raised if the size of the labels does not match the stize of values.

setdefault

ParameterGroup.**setdefault**(*key*, *default=None*, */*)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

to_csv

ParameterGroup.**to_csv**(*filename: str*, *delimiter: str = ','*) → None
Save a *ParameterGroup* to a CSV file.

> **Warning:** Deprecated use `glotaran.io.save_parameters(parameters, file_name=<parameters.csv>, format_name="csv")` instead.

Parameters

- **filename** (*str*) – File to write the parameter specs to.
- **delimiter** (*str*) – Character to separate columns., by default ","

to_dataframe

ParameterGroup.**to_dataframe**(*as_optimized: bool = True*) → pandas.core.frame.DataFrame
Create a pandas data frame from the group.

Parameters **as_optimized** (*bool*) – Whether to include properties which are the result of optimization.

Returns The created data frame.

Return type pd.DataFrame

to_parameter_dict_list

ParameterGroup.**to_parameter_dict_list**(*as_optimized: bool = True*) → list[dict[str, Any]]
Create list of parameter dictionaries from the group.

Parameters **as_optimized** (*bool*) – Whether to include properties which are the result of optimization.

Returns Alist of parameter dictionaries.

Return type list[dict[str, Any]]

update

ParameterGroup.**update**([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

update_parameter_expression

ParameterGroup.**update_parameter_expression**()
Update all parameters which have an expression.

Raises ValueError – Raised if an expression evaluates to a non-numeric value.

values

ParameterGroup.**values**() → an object providing a view on D's values

Methods Documentation

add_group(*group:* glotaran.parameter.parameter_group.ParameterGroup)
Add a *ParameterGroup* to the group.

Parameters **group** (ParameterGroup) – The group to add.

Raises TypeError – Raised if the group is not an instance of *ParameterGroup*.

add_parameter(*parameter: Parameter | list[Parameter]*)
Add a Parameter to the group.

Parameters **parameter** (Parameter | *list*[Parameter]) – The parameter to add.

Raises TypeError – If parameter or any item of it is not an instance of Parameter.

all(*root: str | None = None*, *separator: str = '.'*) → Generator[tuple[str, Parameter], None, None]
Iterate over all parameter in the group and it's subgroups together with their labels.

Parameters

- **root** (*str*) – The label of the root group
- **separator** (*str*) – The separator for the parameter labels.

Yields *tuple[str, Parameter]* – A tuple containing the full label of the parameter and the parameter itself.

clear() → None. Remove all items from D.

copy() → *glotaran.parameter.parameter_group.ParameterGroup*
Create a copy of the *ParameterGroup*.

Returns A copy of the *ParameterGroup*.

Return type *ParameterGroup*

classmethod from_dataframe(*df: pandas.core.frame.DataFrame*, *source: str = 'DataFrame'*) → *glotaran.parameter.parameter_group.ParameterGroup*

Create a *ParameterGroup* from a `pandas.DataFrame`.

Parameters

- **df** (*pd.DataFrame*) – The source data frame.
- **source** (*str*) – Optional name of the source file, used for error messages.

Returns The created parameter group.

Return type *ParameterGroup*

Raises **ValueError** – Raised if the columns 'label' or 'value' doesn't exist. Also raised if the columns 'minimum', 'maximum' or 'values' contain non numeric values or if the columns 'non-negative' or 'vary' are no boolean.

classmethod from_dict(*parameter_dict: dict[str, dict[str, Any] | list[float | list[Any]]]*, *label: str = None*, *root_group: ParameterGroup = None*) → *ParameterGroup*

Create a *ParameterGroup* from a dictionary.

Parameters

- **parameter_dict** (*dict[str, dict | list]*) – A parameter dictionary containing parameters.
- **label** (*str*) – The label of the group.
- **root_group** (ParameterGroup) – The root group

Returns The created *ParameterGroup*

Return type *ParameterGroup*

classmethod from_list(*parameter_list: list[float | list[Any]]*, *label: str = None*, *root_group: ParameterGroup = None*) → *ParameterGroup*

Create a *ParameterGroup* from a list.

Parameters

- **parameter_list** (*list[float | list[Any]]*) – A parameter list containing parameters
- **label** (*str*) – The label of the group.
- **root_group** (ParameterGroup) – The root group

Returns The created *ParameterGroup*.

Return type *ParameterGroup*

classmethod from_parameter_dict_list(*parameter_dict_list: list[dict[str, Any]]*) → *ParameterGroup*

Create a *ParameterGroup* from a list of parameter dictionaries.

Parameters **parameter_dict_list** (*list[dict[str, Any]]*) – A list of parameter dictionaries.

Returns The created *ParameterGroup*.

Return type *ParameterGroup*

fromkeys(*iterable*, *value=None*, */*)

Create a new dictionary with keys from iterable and values set to value.

get(*label: str*) → *glotaran.parameter.parameter.Parameter*
Get a Parameter by its label.

Parameters **label** (*str*) – The label of the parameter, with its path in a ParameterGroup prepended.

Returns The parameter.

Return type *Parameter*

Raises **ParameterNotFoundException** – Raised if no parameter with the given label exists.

get_group_for_parameter_by_label(*parameter_label: str*, *create_if_not_exist: bool = False*) → *glotaran.parameter.parameter_group.ParameterGroup*
Get the group for a parameter by it's label.

Parameters

- **parameter_label** (*str*) – The parameter label.
- **create_if_not_exist** (*bool*) – Create the parameter group if not existent.

Returns The group of the parameter.

Return type *ParameterGroup*

Raises **KeyError** – Raised if the group does not exist and *create_if_not_exist* is *False*.

get_label_value_and_bounds_arrays(*exclude_non_vary: bool = False*) → tuple[list[str], np.ndarray, np.ndarray, np.ndarray]
Return a arrays of all parameter labels, values and bounds.

Parameters **exclude_non_vary** (*bool*) – If true, parameters with *vary=False* are excluded.

Returns A tuple containing a list of parameter labels and an array of the values, lower and upper bounds.

Return type tuple[list[str], np.ndarray, np.ndarray, np.ndarray]

get_nr_roots() → int
Return the number of roots of the group.

Returns The number of roots.

Return type int

groups() → Generator[*glotaran.parameter.parameter_group.ParameterGroup*, None, None]
Return a generator over all groups and their subgroups.

Yields *ParameterGroup* – A subgroup of ParameterGroup.

has(*label: str*) → bool
Check if a parameter with the given label is in the group or in a subgroup.

Parameters **label** (*str*) – The label of the parameter, with its path in a ParameterGroup prepended.

Returns Whether a parameter with the given label exists in the group.

Return type bool

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

property label: str | None
Label of the group.

Returns The label of the group.

Return type str

loader(*format_name: str = None*, ***kwargs*) → *ParameterGroup*
Create a *ParameterGroup* instance from the specs defined in a file.

Parameters

- **file_name** (*StrOrPath*) – File containing the parameter specs.
- **format_name** (*str*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_parameters` implementation of the project io plugin.

Returns *ParameterGroup* instance created from the file.

Return type *ParameterGroup*

markdown(*float_format: str = '.3e'*) → *glotaran.utils.ipython.MarkdownStr*
Format the *ParameterGroup* as markdown string.

This is done by recursing the nested *ParameterGroup* tree.

Parameters **float_format** (*str*) – Format string for floating point numbers, by default ".3e"

Returns The markdown representation as string.

Return type *MarkdownStr*

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem(/)
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

property root_group: *ParameterGroup* | None
Root of the group.

Returns The root group.

Return type *ParameterGroup*

set_from_history(*history: ParameterHistory*, *index: int*)
Update the *ParameterGroup* with values from a parameter history.

Parameters

- **history** (*ParameterHistory*) – The parameter history.
- **index** (*int*) – The history index.

set_from_label_and_value_arrays(*labels: list[str]*, *values: np.ndarray*)
Update the parameter values from a list of labels and values.

Parameters

- **labels** (*list[str]*) – A list of parameter labels.
- **values** (*np.ndarray*) – An array of parameter values.

Raises **ValueError** – Raised if the size of the labels does not match the stize of values.

setdefault(*key*, *default=None*, */*)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

to_csv(*filename: str*, *delimiter: str = ','*) → None
Save a *ParameterGroup* to a CSV file.

Warning: Deprecated use `glotaran.io.save_parameters(parameters, file_name=<parameters.csv>, format_name="csv")` instead.

Parameters

- **filename** (*str*) – File to write the parameter specs to.
- **delimiter** (*str*) – Character to separate columns., by default ","

to_dataframe(*as_optimized: bool = True*) → pandas.core.frame.DataFrame
Create a pandas data frame from the group.

Parameters **as_optimized** (*bool*) – Whether to include properties which are the result of optimization.

Returns The created data frame.

Return type pd.DataFrame

to_parameter_dict_list(*as_optimized: bool = True*) → list[dict[str, Any]]
Create list of parameter dictionaries from the group.

Parameters **as_optimized** (*bool*) – Whether to include properties which are the result of optimization.

Returns Alist of parameter dictionaries.

Return type list[dict[str, Any]]

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

update_parameter_expression()
Update all parameters which have an expression.

Raises **ValueError** – Raised if an expression evaluates to a non-numeric value.

values() → an object providing a view on D's values

Exceptions

Exception Summary

ParameterNotFoundException	Raised when a Parameter is not found in the Group.

ParameterNotFoundException

exception glotaran.parameter.parameter_group.**ParameterNotFoundException**(*path*, *label*)

Raised when a Parameter is not found in the Group.

parameter_history

The glotaran parameter history package.

Classes

Summary

ParameterHistory	A class representing a history of parameters.

ParameterHistory

class glotaran.parameter.parameter_history.**ParameterHistory**

Bases: object

A class representing a history of parameters.

Attributes Summary

number_of_records	Return the number of records in the history.
parameter_labels	Return the labels of the parameters in the history.
parameters	Return the parameters in the history.

number_of_records

ParameterHistory.**number_of_records**
Return the number of records in the history.

Returns The number of records.

Return type int

parameter_labels

ParameterHistory.**parameter_labels**
Return the labels of the parameters in the history.

Returns A list of parameter labels.

Return type list[str]

parameters

ParameterHistory.**parameters**
Return the parameters in the history.

Returns A list of parameters in the history.

Return type list[np.ndarray]

Methods Summary

append	Append a ParameterGroup to the history.
from_csv	Create a history from a csv file.
from_dataframe	Create a history from a pandas data frame.
get_parameters	Get parameters for a history index.
loader	Create a history from a csv file.
to_csv	Write a ParameterGroup to a CSV file.
to_dataframe	Create a data frame from the history.

append

ParameterHistory.**append**(*parameter_group:* glotaran.parameter.parameter_group.ParameterGroup)
Append a ParameterGroup to the history.

Parameters **parameter_group** (ParameterGroup) – The group to append.

Raises **ValueError** – Raised if the parameter labels of the group differs from previous groups.

from_csv

classmethod ParameterHistory.**from_csv**(*path: str*) → *glotaran.parameter.parameter_history.ParameterHistory*

Create a history from a csv file.

Parameters **path** (*str*) – The path to the csv file.

Returns The created history.

Return type *ParameterHistory*

from_dataframe

classmethod ParameterHistory.**from_dataframe**(*history_df: pandas.core.frame.DataFrame*) → *glotaran.parameter.parameter_history.ParameterHistory*

Create a history from a pandas data frame.

Parameters **history_df** (*pd.DataFrame*) – The source data frame.

Returns The created history.

Return type *ParameterHistory*

get_parameters

ParameterHistory.**get_parameters**(*index: int*) → numpy.ndarray

Get parameters for a history index.

Parameters **index** (*int*) – The history index.

Returns The parameter values at the history index as array.

Return type np.ndarray

loader

classmethod ParameterHistory.**loader**(*path: str*) → *glotaran.parameter.parameter_history.ParameterHistory*

Create a history from a csv file.

Parameters **path** (*str*) – The path to the csv file.

Returns The created history.

Return type *ParameterHistory*

to_csv

ParameterHistory.**to_csv**(*file_name: str | PathLike[str]*, *delimiter: str = ','*)
Write a ParameterGroup to a CSV file.

Parameters

- **file_name** (*str*) – The path to the CSV file.
- **delimiter** (*str*) – The delimiter of the CSV file.

to_dataframe

ParameterHistory.**to_dataframe**() → pandas.core.frame.DataFrame
Create a data frame from the history.

Returns The created data frame.

Return type pd.DataFrame

Methods Documentation

append(*parameter_group:* glotaran.parameter.parameter_group.ParameterGroup)
Append a ParameterGroup to the history.

Parameters **parameter_group** (ParameterGroup) – The group to append.

Raises **ValueError** – Raised if the parameter labels of the group differs from previous groups.

classmethod **from_csv**(*path: str*) → *glotaran.parameter.parameter_history.ParameterHistory*
Create a history from a csv file.

Parameters **path** (*str*) – The path to the csv file.

Returns The created history.

Return type *ParameterHistory*

classmethod **from_dataframe**(*history_df: pandas.core.frame.DataFrame*) → *glotaran.parameter.parameter_history.ParameterHistory*
Create a history from a pandas data frame.

Parameters **history_df** (*pd.DataFrame*) – The source data frame.

Returns The created history.

Return type *ParameterHistory*

get_parameters(*index: int*) → numpy.ndarray
Get parameters for a history index.

Parameters **index** (*int*) – The history index.

Returns The parameter values at the history index as array.

Return type np.ndarray

classmethod **loader**(*path: str*) → *glotaran.parameter.parameter_history.ParameterHistory*
Create a history from a csv file.

Parameters **path** (*str*) – The path to the csv file.

Returns The created history.

Return type *ParameterHistory*

property number_of_records: int
Return the number of records in the history.

Returns The number of records.

Return type int

property parameter_labels: list[str]
Return the labels of the parameters in the history.

Returns A list of parameter labels.

Return type list[str]

property parameters: list[np.ndarray]
Return the parameters in the history.

Returns A list of parameters in the history.

Return type list[np.ndarray]

to_csv(*file_name: str | PathLike[str]*, *delimiter: str = ','*)
Write a `ParameterGroup` to a CSV file.

Parameters

- **file_name** (*str*) – The path to the CSV file.
- **delimiter** (*str*) – The delimiter of the CSV file.

to_dataframe() → pandas.core.frame.DataFrame
Create a data frame from the history.

Returns The created data frame.

Return type pd.DataFrame

15.1.9 plugin_system

Plugin system package containing all plugin related implementations.

Modules

glotaran.plugin_system.base_registry	Functionality to register, initialize and retrieve glotaran plugins.
glotaran.plugin_system.data_io_registration	Data Io registration convenience functions.
glotaran.plugin_system.io_plugin_utils	Utility functions for io plugin.
glotaran.plugin_system.megacomplex_registration	Megacomplex registration convenience functions.
glotaran.plugin_system.project_io_registration	Project Io registration convenience functions.

base_registry

Functionality to register, initialize and retrieve glotaran plugins.

Since this module is imported at the root `__init__.py` file all other glotaran imports should be used for typechecking only in the 'if TYPE_CHECKING' block. This is to prevent issues with circular imports.

Functions

Summary

add_instantiated_plugin_to_registry	Add instances of plugin_class to the given registry.
add_plugin_to_registry	Add a plugin with name `plugin_register_key` to the given registry.
full_plugin_name	Full name of a plugin instance/class similar to the `repr`.
get_method_from_plugin	Retrieve a method callabe from an class or instance plugin.
get_plugin_from_registry	Retrieve a plugin with name `plugin_register_key` is registered in a given registry.
is_registered_plugin	Check if a plugin with name `plugin_register_key` is registered in the given registry.
load_plugins	Initialize plugins registered under the entrypoint 'glotaran.plugins'.
methods_differ_from_baseclass	Check if a plugins methods implementation differ from its baseclass.
methods_differ_from_baseclass_table	Create table of which plugins methods differ from their baseclass.
registered_plugins	Names of the plugins in the given registry.
set_plugin	Set a plugins short name to a specific plugin referred by its full name.
show_method_help	Show help on a method as if it was called directly on it.

add_instantiated_plugin_to_registry

glotaran.plugin_system.base_registry.**add_instantiated_plugin_to_registry**(*plugin_register_keys: str | list[str], plugin_class: type[_PluginInstantiableType], plugin_registry: MutableMapping[str, _PluginInstantiableType], plugin_set_func_name: str*) → None

Add instances of plugin_class to the given registry.

Parameters

- **plugin_register_keys** (*str | list[str]*) – Name/-s of the plugin under which it is registered.
- **plugin_class** (*type[_PluginInstantiableType]*) – Pluginclass which should be instantiated with `plugin_register_keys` and added to the registry.
- **plugin_registry** (*MutableMapping[str, _PluginInstantiableType]*) – Registry the plugin should be added to.
- **plugin_set_func_name** (*str*) – Name of the function used to pin a plugin.

See also:

`add_plugin_to_register`

add_plugin_to_registry

glotaran.plugin_system.base_registry.**add_plugin_to_registry**(*plugin_register_key: str, plugin: _PluginType, plugin_registry: MutableMapping[str, _PluginType], plugin_set_func_name: str, instance_identifier: str = ''*) → None

Add a plugin with name `plugin_register_key` to the given registry.

In addition it also adds the plugin with it full import path name as key, which allows for a better reproducibility in case there are conflicting plugins.

Parameters

- **plugin_register_key** (*str*) – Name of the plugin under which it is registered.
- **plugin** (*_PluginType*) – Plugin to be added to the registry.
- **plugin_registry** (*MutableMapping[str, _PluginType]*) – Registry the plugin should be added to.
- **plugin_set_func_name** (*str*) – Name of the function used to pin a plugin.
- **instance_identifier** (*str*) – Used to differentiate between plugin instances (e.g. different format for IO plugins)

Raises ValueError – If `plugin_register_key` has the character '.' in it.

See also:

`add_instantiated_plugin_to_register`, *full_plugin_name*

full_plugin_name

glotaran.plugin_system.base_registry.**full_plugin_name**(*plugin: object | type[object]*) → str

Full name of a plugin instance/class similar to the `repr`.

Parameters **plugin** (*object | type[object]*) – plugin instance/class

Examples

```
>>> from glotaran.builtin.io.sdt.sdt_file_reader import SdtDataIo
>>> full_plugin_name(SdtDataIo)
"glotaran.builtin.io.sdt.sdt_file_reader.SdtDataIo"
>>> full_plugin_name(SdtDataIo("sdt"))
"glotaran.builtin.io.sdt.sdt_file_reader.SdtDataIo"
```

Returns Full name of the plugin.

Return type str

get_method_from_plugin

glotaran.plugin_system.base_registry.**get_method_from_plugin**(*plugin: object | type[object]*, *method_name: str*) → Callable[..., Any]

Retrieve a method callabe from an class or instance plugin.

Parameters

- **plugin** (*object | type[object],*) – Plugin instance or class.
- **method_name** (*str*) – Method name, e.g. load_megacomplex.

Returns Method callable.

Return type Callable[..., Any]

Raises

- **ValueError** – If plugin has an attribute with that name but it isn't callable.
- **ValueError** – If plugin misses the attribute.

get_plugin_from_registry

glotaran.plugin_system.base_registry.**get_plugin_from_registry**(*plugin_register_key: str*, *plugin_registry: MutableMapping[str, _PluginType]*, *not_found_error_message: str*) → _PluginType

Retrieve a plugin with name `plugin_register_key` is registered in a given registry.

Parameters

- **plugin_register_key** (*str*) – Name of the plugin under which it is registered.
- **plugin_registry** (*MutableMapping[str, _PluginType]*) – Registry to search in.
- **not_found_error_message** (*str*) – Error message to be shown if the plugin wasn't found.

Returns Plugin from the plugin Registry.

Return type _PluginType

Raises **ValueError** – If there was no plugin registered under the name `plugin_register_key`.

is_registered_plugin

glotaran.plugin_system.base_registry.**is_registered_plugin**(*plugin_register_key: str*, *plugin_registry: MutableMapping[str, _PluginType]*) → bool

Check if a plugin with name `plugin_register_key` is registered in the given registry.

Parameters

- **plugin_register_key** (*str*) – Name of the plugin under which it is registered.
- **plugin_registry** (*MutableMapping[str, _PluginType]*) – Registry to search in.

Returns Whether or not a plugin is in the registry.

Return type bool

load_plugins

glotaran.plugin_system.base_registry.**load_plugins**()

Initialize plugins registered under the entrypoint 'glotaran.plugins'.

For an entry_point to be considered a glotaran plugin it just needs to start with 'glotaran.plugins', which allows for an easy extendability.

Currently used builtin entrypoints are:

- `glotaran.plugins.data_io`
- `glotaran.plugins.megacomplex`
- `glotaran.plugins.project_io`

methods_differ_from_baseclass

glotaran.plugin_system.base_registry.**methods_differ_from_baseclass**(*method_names: str | Sequence[str]*, *plugin: GenericPluginInstance | type[GenericPluginInstance]*, *base_class: type[GenericPluginInstance]*) → list[bool]

Check if a plugins methods implementation differ from its baseclass.

Based on the assumption that `base_class` didn't implement the methods (e.g. `DataIoInterface` or `ProjectIoInterface`), this can be used to to create a 'supported methods' list.

Parameters

- **method_names** (*str | list[str]*) – Name|s of the method|s
- **plugin** (*GenericPluginInstance | type[GenericPluginInstance]*) – Plugin class or instance.
- **base_class** (*type[GenericPluginInstance]*) – Base class the plugin inherited from.

Returns List containing whether or not a plugins method differs from the baseclasses.

Return type list[bool]

methods_differ_from_baseclass_table

glotaran.plugin_system.base_registry.**methods_differ_from_baseclass_table**(*method_names: str | Sequence[str], plugin_registry_keys: str | Sequence[str], get_plugin_function: Callable[[str], GenericPluginInstance | type[GenericPluginInstance]], base_class: type[GenericPluginInstance], plugin_names: bool = False*) → list[list[str | bool]]

Create table of which plugins methods differ from their baseclass.

This uses the assumption that all plugins have the same `base_class`.

The main purpose of this function is to show the user which plugin implements which methods differently than its baseclass.

Based on the assumption that `base_class` didn't implement the methods (e.g. `DataIoInterface` or `ProjectIoInterface`), this can be used to to create a 'supported methods' table.

Parameters

- **method_names** (*str | list[str]*) – Name|s of the method|s.
- **plugin_registry_keys** (*str | list[str]*) – Keys the plugins are registered under (e.g. return value of the implementation of func:*registered_plugins*)
- **get_plugin_function** (*Callable[[str], GenericPluginInstance | type[GenericPluginInstance]]*) – Function to get plugin from plugin registry.
- **base_class** (*type[GenericPluginInstance]*) – Base class the plugin inherited from.
- **plugin_names** (*bool*) – Whether or not to add the names of the plugins to the lists.

Returns Table like structure with the first value of each row being the `plugin_registry_key` and the others whether or not a plugins method differs from the baseclasses.

Return type list[list[str | bool]]

See also:

methods_differ_from_baseclass

registered_plugins

glotaran.plugin_system.base_registry.**registered_plugins**(*plugin_registry: MutableMapping[str, _PluginType]*, *full_names: bool = False*) → list[str]

Names of the plugins in the given registry.

Parameters

- **plugin_registry** (*MutableMapping[str, _PluginType]*) – Registry to search in.
- **full_names** (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns List of plugin names in plugin_registry.

Return type list[str]

set_plugin

glotaran.plugin_system.base_registry.**set_plugin**(*plugin_register_key: str*, *full_plugin_name: str*, *plugin_registry: MutableMapping[str, _PluginType]*, *plugin_register_key_name: str = 'format_name'*) → None

Set a plugins short name to a specific plugin referred by its full name.

This can be used to ensure that a specific plugin is used in case there are conflicting plugins installed.

Parameters

- **plugin_register_key** (*str*) – Name of the plugin under which it is registered.
- **full_plugin_name** (*str*) – Full name (import path) of the registered plugin.
- **plugin_registry** (*MutableMapping[str, _PluginType]*) – Registry the plugin should be set in to.
- **plugin_register_key_name** (*str*) – Name of the arg passed `plugin_register_key` in the function that implements `set_plugin`.

Raises

- **ValueError** – If `plugin_register_key` has the character '.' in it.
- **ValueError** – If there isn't a registered plugin with the key `full_plugin_name`.

See also:

add_plugin_to_registry, *full_plugin_name*

show_method_help

glotaran.plugin_system.base_registry.**show_method_help**(*plugin: object | type[object]*, *method_name: str*) → None

Show help on a method as if it was called directly on it.

Parameters

- **plugin** (*object | type[object],*) – Plugin instance or class.
- **method_name** (*str*) – Method name, e.g. load_megacomplex.

Exceptions

Exception Summary

PluginOverwriteWarning	Warning used if a plugin tries to overwrite and existing plugin.

PluginOverwriteWarning

exception glotaran.plugin_system.base_registry.**PluginOverwriteWarning**(**args: Any*, *old_key: str*, *old_plugin: object | type[object]*, *new_plugin: object | type[object]*, *plugin_set_func_name: str*)

Warning used if a plugin tries to overwrite and existing plugin.

Use old and new plugin and keys to give verbose warning message.

Parameters

- **old_key** (*str*) – Old registry key.
- **old_plugin** (*object | type[object]*) – Old plugin ('registry[old_key]').
- **new_plugin** (*object | type[object]*) – New Plugin ('registry[new_key]').
- **plugin_set_func_name** (*str*) – Name of the function used to pin a plugin.
- ***args** (*Any*) – Additional args passed to the super constructor.

data_io_registration

Data Io registration convenience functions.

Note: The [call-arg] type error would be raised since the base methods doesn't have a `**kwargs` argument, but we rather ignore this error here, than adding `**kwargs` to the base method and causing an [override] type error in the plugins implementation.

Functions

Summary

data_io_plugin_table	Return registered data io plugins and which functions they support as markdown table.
get_data_io	Retrieve a data io plugin from the data_io registry.
get_dataloader	Retrieve implementation of the `read_dataset` functionality for the format 'format_name'.
get_datasaver	Retrieve implementation of the `save_dataset` functionality for the format 'format_name'.
is_known_data_format	Check if a data format is in the data_io registry.
known_data_formats	Names of the registered data io plugins.
load_dataset	Read data from a file to xarray.Dataset or xarray.DataArray.
register_data_io	Register data io plugins to one or more formats.
save_dataset	Save data from xarray.Dataset or xarray.DataArray to a file.
set_data_plugin	Set the plugin used for a specific data format.
show_data_io_method_help	Show help for the implementation of data io plugin methods.

data_io_plugin_table

glotaran.plugin_system.data_io_registration.**data_io_plugin_table**(*, *plugin_names: bool = False*, *full_names: bool = False*) → *glotaran.utils.ipython.MarkdownStr*

Return registered data io plugins and which functions they support as markdown table.

This is especially useful when you work with new plugins.

Parameters

- **plugin_names** (*bool*) – Whether or not to add the names of the plugins to the table.
- **full_names** (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns Markdown table of data io plugins.

Return type *MarkdownStr*

get_data_io

glotaran.plugin_system.data_io_registration.**get_data_io**(*format_name: str*) → *glotaran.io.interface.DataIoInterface*

Retrieve a data io plugin from the data_io registry.

Parameters **format_name** (*str*) – Name of the data io plugin under which it is registered.

Returns Data io plugin instance.

Return type *DataIoInterface*

get_dataloader

glotaran.plugin_system.data_io_registration.**get_dataloader**(*format_name: str*) → DataLoader

Retrieve implementation of the `read_dataset` functionality for the format 'format_name'.

This allows to get the proper help and autocomplete for the function, which is especially valuable if the function provides additional options.

Parameters **format_name** (*str*) – Format the dataloader should be able to read.

Returns Function to load data of format `format_name` as xarray.Dataset or xarray.DataArray.

Return type DataLoader

get_datasaver

glotaran.plugin_system.data_io_registration.**get_datasaver**(*format_name: str*) → DataSaver

Retrieve implementation of the `save_dataset` functionality for the format 'format_name'.

This allows to get the proper help and autocomplete for the function, which is especially valuable if the function provides additional options.

Parameters **format_name** (*str*) – Format the datawriter should be able to write.

Returns Function to write xarray.Dataset to the format `format_name` .

Return type DataSaver

is_known_data_format

glotaran.plugin_system.data_io_registration.**is_known_data_format**(*format_name: str*) → bool

Check if a data format is in the data_io registry.

Parameters **format_name** (*str*) – Name of the data io plugin under which it is registered.

Returns Whether or not the data format is a registered data io plugins.

Return type bool

known_data_formats

glotaran.plugin_system.data_io_registration.**known_data_formats**(*full_names: bool = False*) → list[str]

Names of the registered data io plugins.

Parameters **full_names** (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns List of registered data io plugins.

Return type list[str]

load_dataset

glotaran.plugin_system.data_io_registration.**load_dataset**(*file_name: StrOrPath*, *format_name: str = None*, ***kwargs: Any*) → xr.Dataset

Read data from a file to xarray.Dataset or xarray.DataArray.

Parameters

- **file_name** (*StrOrPath*) – File containing the data.
- **format_name** (*str*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `read_dataset` implementation of the data io plugin. If you aren't sure about those use `get_dataloader` to get the implementation with the proper help and autocomplete.

Returns Data loaded from the file.

Return type xr.Dataset

register_data_io

glotaran.plugin_system.data_io_registration.**register_data_io**(*format_names: str | list[str]*) → Callable[[type[DataIoInterface]], type[DataIoInterface]]

Register data io plugins to one or more formats.

Decorate a data io plugin class with `@register_data_io(format_name|[*format_names])` to add it to the registry.

Parameters **format_names** (*str | list[str]*) – Name of the data io plugin under which it is registered.

Returns Inner decorator function.

Return type Callable[[type[*DataIoInterface*]], type[*DataIoInterface*]]

Examples

```
>>> @register_data_io("my_format_1")
... class MyDataIo1(DataIoInterface):
...     pass
```

```
>>> @register_data_io(["my_format_1", "my_format_1_alias"])
... class MyDataIo2(DataIoInterface):
...     pass
```

save_dataset

glotaran.plugin_system.data_io_registration.**save_dataset**(*dataset: xr.Dataset | xr.DataArray*, *file_name: StrOrPath*, *format_name: str = None*, ***, *data_filters: list[str] | None = None*, *allow_overwrite: bool = False*, *update_source_path: bool = True*, ***kwargs: Any*) → None

Save data from xarray.Dataset or xarray.DataArray to a file.

Parameters

- **dataset** (*xr.Dataset | xr.DataArray*) – Data to be written to file.
- **file_name** (*StrOrPath*) – File to write the data to.
- **format_name** (*str*) – Format the file should be in, if not provided it will be inferred from the file extension.
- **data_filters** (*list[str] | None*) – Optional list of items in the dataset to be saved.
- **allow_overwrite** (*bool*) – Whether or not to allow overwriting existing files, by default False
- **update_source_path** (*bool*) – Whether or not to update the `source_path` attribute to `file_name` when saving. by default True
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `write_dataset` implementation of the data io plugin. If you aren't sure about those use `get_datawriter` to get the implementation with the proper help and autocomplete.

set_data_plugin

glotaran.plugin_system.data_io_registration.**set_data_plugin**(*format_name: str*, *full_plugin_name: str*) → None

Set the plugin used for a specific data format.

This function is useful when you want to resolve conflicts of installed plugins or overwrite the plugin used for a specific format.

Effected functions:

- *load_dataset()*
- *save_dataset()*

Parameters

- **format_name** (*str*) – Format name used to refer to the plugin when used for `save` and `load` functions.
- **full_plugin_name** (*str*) – Full name (import path) of the registered plugin.

show_data_io_method_help

glotaran.plugin_system.data_io_registration.**show_data_io_method_help**(*format_name: str*, *method_name: Literal['load_dataset', 'save_dataset']*) → None

Show help for the implementation of data io plugin methods.

Parameters

- **format_name** (*str*) – Format the method should support.
- **method_name** (*{'load_dataset', 'save_dataset'}*) – Method name

io_plugin_utils

Utility functions for io plugin.

Functions

Summary

bool_str_repr	Replace boolean value with string repr.
bool_table_repr	Replace boolean value with string repr for all table values.
inferr_file_format	Inferr format of a file if it exists.

continues on next page

Table 182 – continued from previous page

not_implemented_to_value_error	Decorate a function to raise ValueError instead of NotImplementedError.
protect_from_overwrite	Raise FileExistsError if files already exists and allow_overwrite isn't True.

bool_str_repr

glotaran.plugin_system.io_plugin_utils.**bool_str_repr**(*value: Any*, *true_repr: str = '*'*, *false_repr: str = '/'*) → Any

Replace boolean value with string repr.

This function is a helper for table representation (e.g. with tabulate) of boolean values.

Parameters

- **value** (*Any*) – Arbitrary value
- **true_repr** (*str*) – Desired repr for `True`, by default "*"
- **false_repr** (*str*) – Desired repr for `False`, by default "/"

Returns Original value or desired repr for bool

Return type Any

Examples

```
>>> table_data = [["foo", True, False], ["bar", False, True]]
>>> print(tabulate(map(lambda x: map(bool_table_repr, x), table_data)))
---  -  -
foo  *  /
bar  /  *
---  -  -
```

bool_table_repr

glotaran.plugin_system.io_plugin_utils.**bool_table_repr**(*table_data: Iterable[Iterable[Any]]*, *true_repr: str = '*'*, *false_repr: str = '/'*) → Iterator[Iterator[Any]]

Replace boolean value with string repr for all table values.

This function is an implementation of *bool_str_repr()* for a 2D table, for easy usage with tabulate.

Parameters

- **table_data** (*Iterable[Iterable[Any]]*) – Data of the table e.g. a list of lists.
- **true_repr** (*str*) – Desired repr for `True`, by default "*"
- **false_repr** (*str*) – Desired repr for `False`, by default "/"

Returns `table_data` with original values or desired repr for bool

Return type Iterator[Iterator[Any]]

See also:

bool_str_repr

Examples

```
>>> table_data = [["foo", True, False], ["bar", False, True]]
>>> print(tabulate(bool_table_repr(table_data))
---  -  -
foo  *  /
bar  /  *
---  -  -
```

inferr_file_format

glotaran.plugin_system.io_plugin_utils.**inferr_file_format**(*file_path: StrOrPath*, *, *needs_to_exist: bool = True*, *allow_folder=False*) → str

Inferr format of a file if it exists.

Parameters

- **file_path** (*StrOrPath*) – Path/str to the file.
- **needs_to_exist** (*bool*) – Whether or not a file need to exists for an successful format inferring. While write functions don't need the file to exists, load functions do.
- **allow_folder** (*bool*) – Whether or not to allow the format to be `folder`. This is only used in `save_result`.

Returns File extension without the leading dot.

Return type str

Raises

- **ValueError** – If file doesn't exists.
- **ValueError** – If file has no extension.

not_implemented_to_value_error

glotaran.plugin_system.io_plugin_utils.**not_implemented_to_value_error**(*func: glotaran.plugin_system.io_plugin_uti* → glotaran.plugin_system.io_plugin_uti

Decorate a function to raise ValueError instead of NotImplementedError.

This decorator is supposed to be used on functions which call functions that might raise a NotImplementedError, but raise ValueError instead with the same error text.

Parameters **func** (*DecoratedFunc*) – Function to be decorated.

Returns Wrapped function.

Return type DecoratedFunc

protect_from_overwrite

glotaran.plugin_system.io_plugin_utils.**protect_from_overwrite**(*path: str | os.PathLike[str]*, ***, *allow_overwrite: bool = False*) → None

Raise FileExistsError if files already exists and allow_overwrite isn't True.

Parameters

- **path** (*str*) – Path to a file or folder.
- **allow_overwrite** (*bool*) – Whether or not to allow overwriting existing files, by default False

Raises

- **FileExistsError** – If path points to an existing file.
- **FileExistsError** – If path points to an existing folder which is not empty.

megacomplex_registration

Megacomplex registration convenience functions.

Functions

Summary

get_megacomplex	Retrieve a megacomplex from the megacomplex registry.
is_known_megacomplex	Check if a megacomplex is in the megacomplex registry.
known_megacomplex_names	Names of the registered megacomplexs.
megacomplex_plugin_table	Return registered megacomplex plugins as markdown table.
register_megacomplex	Add a megacomplex to the megacomplex registry.
set_megacomplex_plugin	Set the plugin used for a specific megacomplex name.

get_megacomplex

glotaran.plugin_system.megacomplex_registration.**get_megacomplex**(*megacomplex_type: str*) → type[Megacomplex]

Retrieve a megacomplex from the megacomplex registry.

Parameters **megacomplex_type** (*str*) – Name of the megacomplex under which it is registered.

Returns Megacomplex class

Return type type[Megacomplex]

is_known_megacomplex

glotaran.plugin_system.megacomplex_registration.**is_known_megacomplex**(*megacomplex_type: str*) → bool

Check if a megacomplex is in the megacomplex registry.

Parameters **megacomplex_type** (*str*) – Name of the megacomplex under which it is registered.

Returns Whether or not the megacomplex is registered.

Return type bool

known_megacomplex_names

glotaran.plugin_system.megacomplex_registration.**known_megacomplex_names**(*full_names: bool = False*) → list[str]

Names of the registered megacomplexs.

Parameters **full_names** (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns List of registered megacomplexs.

Return type list[str]

megacomplex_plugin_table

glotaran.plugin_system.megacomplex_registration.**megacomplex_plugin_table**(*, *plugin_names: bool = False*, *full_names: bool = False*) → *glotaran.utils.ipython.MarkdownS*

Return registered megacomplex plugins as markdown table.

This is especially useful when you work with new plugins.

Parameters

- **plugin_names** (*bool*) – Whether or not to add the names of the plugins to the table.
- **full_names** (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns Markdown table of megacomplexnames.

Return type *MarkdownStr*

register_megacomplex

glotaran.plugin_system.megacomplex_registration.**register_megacomplex**(*megacomplex_type: str*, *megacomplex: type[Megacomplex]*) → None

Add a megacomplex to the megacomplex registry.

Parameters

- **megacomplex_type** (*str*) – Name of the megacomplex under which it is registered.
- **megacomplex** (*type[Megacomplex]*) – megacomplex class to be registered.

set_megacomplex_plugin

glotaran.plugin_system.megacomplex_registration.**set_megacomplex_plugin**(*megacomplex_name: str*, *full_plugin_name: str*) → None

Set the plugin used for a specific megacomplex name.

This function is useful when you want to resolve conflicts of installed plugins or overwrite the plugin used for a specific megacomplex name.

Effected functions:

- `optimize()`

Parameters

- **megacomplex_name** (*str*) – Name of the megacomplex to use the plugin for.
- **full_plugin_name** (*str*) – Full name (import path) of the registered plugin.

project_io_registration

Project Io registration convenience functions.

Note: The [call-arg] type error would be raised since the base methods doesn't have a `**kwargs` argument, but we rather ignore this error here, than adding `**kwargs` to the base method and causing an [override] type error in the plugins implementation.

Functions

Summary

get_project_io	Retrieve a data io plugin from the project_io registry.
get_project_io_method	Retrieve implementation of project io functionality for the format 'format_name'.
is_known_project_format	Check if a data format is in the project_io registry.
known_project_formats	Names of the registered project io plugins.
load_model	Create a Model instance from the specs defined in a file.
load_parameters	Create a `ParameterGroup` instance from the specs defined in a file.
load_result	Create a `Result` instance from the specs defined in a file.
load_scheme	Create a `Scheme` instance from the specs defined in a file.
project_io_plugin_table	Return registered project io plugins and which functions they support as markdown table.
register_project_io	Register project io plugins to one or more formats.
save_model	Save a `Model` instance to a spec file.
save_parameters	Save a `ParameterGroup` instance to a spec file.
save_result	Write a `Result` instance to a spec file.
save_scheme	Save a `Scheme` instance to a spec file.
set_project_plugin	Set the plugin used for a specific project format.
show_project_io_method_help	Show help for the implementation of project io plugin methods.

get_project_io

glotaran.plugin_system.project_io_registration.**get_project_io**(*format_name: str*) → *glotaran.io.interface.ProjectIoInterface*

Retrieve a data io plugin from the project_io registry.

Parameters **format_name** (*str*) – Name of the data io plugin under which it is registered.

Returns Project io plugin instance.

Return type *ProjectIoInterface*

get_project_io_method

glotaran.plugin_system.project_io_registration.**get_project_io_method**(*format_name: str*, *method_name: ProjectIoMethods*) → Callable[..., Any]

Retrieve implementation of project io functionality for the format 'format_name'.

This allows to get the proper help and autocomplete for the function, which is especially valuable if the function provides additional options.

Parameters

- **format_name** (*str*) – Format the dataloader should be able to read.
- **method_name** (*{'load_model', 'write_model', 'load_parameters', 'write_parameters', 'load_scheme', 'write_scheme', 'load_result', 'write_result'}*) – Method name, e.g. load_model.

Returns The function which is called in the background by the convenience functions.

Return type Callable[..., Any]

is_known_project_format

glotaran.plugin_system.project_io_registration.**is_known_project_format**(*format_name: str*) → bool

Check if a data format is in the project_io registry.

Parameters **format_name** (*str*) – Name of the project io plugin under which it is registered.

Returns Whether or not the data format is a registered project io plugin.

Return type bool

known_project_formats

glotaran.plugin_system.project_io_registration.**known_project_formats**(*full_names: bool = False*) → list[str]

Names of the registered project io plugins.

Parameters **full_names** (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns List of registered project io plugins.

Return type list[str]

load_model

glotaran.plugin_system.project_io_registration.**load_model**(*file_name: StrOrPath*, *format_name: str = None*, ***kwargs: Any*) → Model

Create a Model instance from the specs defined in a file.

Parameters

- **file_name** (*StrOrPath*) – File containing the model specs.
- **format_name** (*str*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_model` implementation of the project io plugin.

Returns Model instance created from the file.

Return type *Model*

load_parameters

glotaran.plugin_system.project_io_registration.**load_parameters**(*file_name: StrOrPath*, *format_name: str = None*, ***kwargs*) → ParameterGroup

Create a `ParameterGroup` instance from the specs defined in a file.

Parameters

- **file_name** (*StrOrPath*) – File containing the parameter specs.
- **format_name** (*str*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_parameters` implementation of the project io plugin.

Returns `ParameterGroup` instance created from the file.

Return type *ParameterGroup*

load_result

glotaran.plugin_system.project_io_registration.**load_result**(*result_path: StrOrPath*, *format_name: str = None*, ***kwargs: Any*) → Result

Create a `Result` instance from the specs defined in a file.

Parameters

- **result_path** (*StrOrPath*) – Path containing the result data.
- **format_name** (*str*) – Format the result is in, if not provided and it is a file it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_result` implementation of the project io plugin.

Returns `Result` instance created from the saved format.

Return type *Result*

load_scheme

glotaran.plugin_system.project_io_registration.**load_scheme**(*file_name: StrOrPath*, *format_name: str = None*, ***kwargs: Any*) → Scheme

Create a `Scheme` instance from the specs defined in a file.

Parameters

- **file_name** (*StrOrPath*) – File containing the parameter specs.
- **format_name** (*str*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_scheme` implementation of the project io plugin.

Returns `Scheme` instance created from the file.

Return type *Scheme*

project_io_plugin_table

glotaran.plugin_system.project_io_registration.**project_io_plugin_table**(*, *plugin_names: bool = False*, *full_names: bool = False*) → *glotaran.utils.ipython.MarkdownStr*

Return registered project io plugins and which functions they support as markdown table.

This is especially useful when you work with new plugins.

Parameters

- **plugin_names** (*bool*) – Whether or not to add the names of the plugins to the table.
- **full_names** (*bool*) – Whether to display the full names the plugins are registered under as well.

Returns Markdown table of project io plugins.

Return type *MarkdownStr*

register_project_io

glotaran.plugin_system.project_io_registration.**register_project_io**(*format_names: str | list[str]*) → Callable[[type[ProjectIoInterface]], type[ProjectIoInterface]]

Register project io plugins to one or more formats.

Decorate a project io plugin class with `@register_project_io(format_name|[*format_names])` to add it to the registry.

Parameters **format_names** (*str | list[str]*) – Name of the project io plugin under which it is registered.

Returns Inner decorator function.

Return type Callable[[type[*ProjectIoInterface*]], type[*ProjectIoInterface*]]

Examples

```
>>> @register_project_io("my_format_1")
... class MyProjectIo1(ProjectIoInterface):
...     pass
```

```
>>> @register_project_io(["my_format_1", "my_format_1_alias"])
... class MyProjectIo2(ProjectIoInterface):
...     pass
```

save_model

glotaran.plugin_system.project_io_registration.**save_model**(*model: Model, file_name: StrOrPath, format_name: str = None, *, allow_overwrite: bool = False, update_source_path: bool = True, **kwargs: Any*) → None

Save a `Model` instance to a spec file.

Parameters

- **model** (`Model`) – `Model` instance to save to specs file.
- **file_name** (*StrOrPath*) – File to write the model specs to.
- **format_name** (*str*) – Format the file should be in, if not provided it will be inferred from the file extension.
- **allow_overwrite** (*bool*) – Whether or not to allow overwriting existing files, by default False
- **update_source_path** (*bool*) – Whether or not to update the `source_path` attribute to `file_name` when saving. by default True
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `save_model` implementation of the project io plugin.

save_parameters

glotaran.plugin_system.project_io_registration.**save_parameters**(*parameters: ParameterGroup*, *file_name: StrOrPath*, *format_name: str = None*, ***, *allow_overwrite: bool = False*, *update_source_path: bool = True*, ***kwargs: Any*) → None

Save a ParameterGroup instance to a spec file.

Parameters

- **parameters** (ParameterGroup) – ParameterGroup instance to save to specs file.
- **file_name** (*StrOrPath*) – File to write the parameter specs to.
- **format_name** (*str*) – Format the file should be in, if not provided it will be inferred from the file extension.
- **allow_overwrite** (*bool*) – Whether or not to allow overwriting existing files, by default False
- **update_source_path** (*bool*) – Whether or not to update the source_path attribute to file_name when saving. by default True
- ****kwargs** (*Any*) – Additional keyword arguments passes to the save_parameters implementation of the project io plugin.

save_result

glotaran.plugin_system.project_io_registration.**save_result**(*result: Result*, *result_path: StrOrPath*, *format_name: str = None*, ***, *allow_overwrite: bool = False*, *update_source_path: bool = True*, ***kwargs: Any*) → list[str] | None

Write a Result instance to a spec file.

Parameters

- **result** (Result) – Result instance to write.
- **result_path** (*StrOrPath*) – Path to write the result data to.
- **format_name** (*str*) – Format the result should be saved in, if not provided and it is a file it will be inferred from the file extension.
- **allow_overwrite** (*bool*) – Whether or not to allow overwriting existing files, by default False

- **update_source_path** (*bool*) – Whether or not to update the `source_path` attribute to `result_path` when saving. by default True
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `save_result` implementation of the project io plugin.

Returns List of file paths which were saved.

Return type list[str] | None

save_scheme

glotaran.plugin_system.project_io_registration.**save_scheme**(*scheme: Scheme*, *file_name: StrOrPath*, *format_name: str = None*, *, allow_overwrite: bool = False*, *update_source_path: bool = True*, ***kwargs: Any*) → None

Save a `Scheme` instance to a spec file.

Parameters

- **scheme** (Scheme) – `Scheme` instance to save to specs file.
- **file_name** (*StrOrPath*) – File to write the scheme specs to.
- **format_name** (*str*) – Format the file should be in, if not provided it will be inferred from the file extension.
- **allow_overwrite** (*bool*) – Whether or not to allow overwriting existing files, by default False
- **update_source_path** (*bool*) – Whether or not to update the `source_path` attribute to `file_name` when saving. by default True
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `save_scheme` implementation of the project io plugin.

set_project_plugin

glotaran.plugin_system.project_io_registration.**set_project_plugin**(*format_name: str*, *full_plugin_name: str*) → None

Set the plugin used for a specific project format.

This function is useful when you want to resolve conflicts of installed plugins or overwrite the plugin used for a specific format.

Effected functions:

- *load_model()*
- *save_model()*
- *load_parameters()*
- *save_parameters()*

- *load_scheme()*
- *save_scheme()*
- *load_result()*
- *save_result()*

Parameters

- **format_name** (*str*) – Format name used to refer to the plugin when used for `save` and `load` functions.
- **full_plugin_name** (*str*) – Full name (import path) of the registered plugin.

show_project_io_method_help

glotaran.plugin_system.project_io_registration.**show_project_io_method_help**(*format_name: str*, *method_name: ProjectIoMethods*) → None

Show help for the implementation of project io plugin methods.

Parameters

- **format_name** (*str*) – Format the method should support.
- **method_name** (*{'load_model', 'write_model', 'load_parameters', 'write_parameters', 'load_scheme', 'write_scheme', 'load_result', 'write_result'}*) – Method name.

Classes

Summary

SavingOptions	A collection of options for result saving.

SavingOptions

class glotaran.plugin_system.project_io_registration.**SavingOptions**(*data_filter: list[str] | None = None*, *data_format: Literal['nc'] = 'nc'*, *parameter_format: Literal['csv'] = 'csv'*, *report: bool = True*)

Bases: object

A collection of options for result saving.

Attributes Summary

data_filter
data_format
parameter_format
report

data_filter

SavingOptions.**data_filter: list[str] | None = None**

data_format

SavingOptions.**data_format: Literal['nc'] = 'nc'**

parameter_format

SavingOptions.**parameter_format: Literal['csv'] = 'csv'**

report

SavingOptions.**report: bool = True**

Methods Summary

—

Methods Documentation

data_filter: list[str] | None = None

data_format: Literal['nc'] = 'nc'

parameter_format: Literal['csv'] = 'csv'

report: bool = True

15.1.10 project

The glotaran project package.

Modules

glotaran.project.dataclass_helpers	Contains helper methods for dataclasses.
glotaran.project.result	The result class for global analysis.
glotaran.project.scheme	The module for :class:Scheme.

dataclass_helpers

Contains helper methods for dataclasses.

Functions

Summary

asdict	Create a dictionary containing all fields of the dataclass.
exclude_from_dict_field	Create a dataclass field with which will be excluded from asdict.
file_loadable_field	Create a dataclass field which can be and object of type targetClass or file path.

continues on next page

Table 189 – continued from previous page

file_loader_factory	Create `file_loader` functions to load `targetClass` from file.
fromdict	Create a dataclass instance from a dict and loads all file represented fields.
init_file_loadable_fields	Load objects into class when dataclass is initialized with paths.

asdict

glotaran.project.dataclass_helpers.**asdict**(*dataclass: object*, *folder: Path = None*) → dict[str, Any]

Create a dictionary containing all fields of the dataclass.

Parameters

- **dataclass** (*object*) – A dataclass instance.
- **folder** (*Path*) – Parent folder of `FileLoadable` fields. by default None

Returns The dataclass represented as a dictionary.

Return type dict[str, Any]

exclude_from_dict_field

glotaran.project.dataclass_helpers.**exclude_from_dict_field**(*default: DefaultType = <dataclasses._MISSING_TYPE object>*) → DefaultType

Create a dataclass field with which will be excluded from `asdict`.

Parameters **default** (*DefaultType*) – The default value of the field.

Returns The created field.

Return type DefaultType

file_loadable_field

glotaran.project.dataclass_helpers.**file_loadable_field**(*targetClass: type[FileLoadable]*, ***, *is_wrapper_class=False*) → FileLoadable

Create a dataclass field which can be and object of type `targetClass` or file path.

Parameters

- **targetClass** (*type[FileLoadable]*) – Class the resulting value should be an instance of.
- **is_wrapper_class** (*bool*) – Whether or not `targetClass` is a wrapper class, so the isinstance check will be ignored and instead the responsibility for supported types lies at the implementation of the loader.

Notes

This also requires to add `init_file_loadable_fields` in the `__post_init__` method.

Returns Instance of `targetClass`.

Return type FileLoadable

See also:

init_file_loadable_fields

file_loader_factory

glotaran.project.dataclass_helpers.**file_loader_factory**(*targetClass: type[FileLoadable]*, ***, *is_wrapper_class: bool = False*) → Callable[[FileLoadable | str | Path], FileLoadable]

Create `file_loader` functions to load `targetClass` from file.

Parameters

- **targetClass** (*type[FileLoadable]*) – Class the loader function should return an instance of.
- **is_wrapper_class** (*bool*) – Whether or not `targetClass` is a wrapper class, so the isinstance check will be ignored and instead the responsibility for supported types lies at the implementation of the loader.

Returns **file_loader** – Function to load `FileLoadable` from source file or return instance if already loaded.

Return type Callable[[FileLoadable | str | Path], FileLoadable]

fromdict

glotaran.project.dataclass_helpers.**fromdict**(*dataclass_type: type*, *dataclass_dict: dict[str, Any]*, *folder: Path = None*) → object

Create a dataclass instance from a dict and loads all file represented fields.

Parameters

- **dataclass_type** (*type*) – A dataclass type.
- **dataclass_dict** (*dict[str, Any]*) – A dict for instancing the the dataclass.
- **folder** (*Path*) – The root folder for file paths. If `None` file paths are consider absolute.

Returns Created instance of dataclass_type.

Return type object

init_file_loadable_fields

glotaran.project.dataclass_helpers.**init_file_loadable_fields**(*dataclass_instance: object*)

Load objects into class when dataclass is initialized with paths.

If the class has file_loadable fields, this needs be called in the `__post_init__` method of that class.

Parameters **dataclass_instance** (*object*) – Instance of the dataclass being initialized. When used inside of `__post_init__` for the class itself use `self`.

See also:

file_loadable_field

result

The result class for global analysis.

Classes

Summary

Result	The result of a global analysis.

Result

class glotaran.project.result.**Result**(*number_of_function_evaluations: int*, *success: bool*, *termination_reason: str*, *glotaran_version: str*, *free_parameter_labels: list[str]*, *scheme: Scheme*, *initial_parameters: ParameterGroup*, *optimized_parameters: ParameterGroup*, *parameter_history: ParameterHistory*, *data: Mapping[str, xr.Dataset]*, *additional_penalty: np.ndarray | None = None*, *cost: ArrayLike | None = None*, *chi_square: float | None = None*, *covariance_matrix: ArrayLike | None = None*, *degrees_of_freedom: int | None = None*, *jacobian: ArrayLike | list | None = None*, *number_of_data_points: int | None = None*, *number_of_jacobian_evaluations: int | None = None*, *number_of_variables: int | None = None*, *optimality: float | None = None*, *reduced_chi_square: float | None = None*, *root_mean_square_error: float | None = None*)

Bases: `object`

The result of a global analysis.

Attributes Summary

additional_penalty	A vector with the value for each additional penalty, or None
chi_square	The chi-square of the optimization.
cost	The final cost.
covariance_matrix	Covariance matrix.
degrees_of_freedom	Degrees of freedom in optimization $N - N_{vars}$.
jacobian	Modified Jacobian matrix at the solution
model	Return the model used to fit result.
number_of_data_points	Number of data points N.
number_of_jacobian_evaluations	The number of jacobian evaluations.
number_of_variables	Number of variables in optimization N_{vars}
optimality	
reduced_chi_square	The reduced chi-square of the optimization.
root_mean_square_error	The root mean square error the optimization.
source_path	
number_of_function_evaluations	The number of function evaluations.
success	Indicates if the optimization was successful.
termination_reason	The reason (message when) the optimizer terminated
glotaran_version	The glotaran version used to create the result.
free_parameter_labels	List of labels of the free parameters used in optimization.
scheme	
initial_parameters	
optimized_parameters	
parameter_history	The parameter history.
data	The resulting data as a dictionary of xarray.Dataset.

additional_penalty

Result.**additional_penalty: np.ndarray | None = None**

A vector with the value for each additional penalty, or None

chi_square

Result.**chi_square: float | None = None**
The chi-square of the optimization.

$\chi^2 = \sum_i^N [Residual_i]^2$.

cost

Result.**cost: ArrayLike | None = None**
The final cost.

covariance_matrix

Result.**covariance_matrix: ArrayLike | None = None**
Covariance matrix.

The rows and columns are corresponding to *free_parameter_labels*.

degrees_of_freedom

Result.**degrees_of_freedom: int | None = None**
Degrees of freedom in optimization $N - N_{vars}$.

jacobian

Result.**jacobian: ArrayLike | list | None = None**
Modified Jacobian matrix at the solution

See also: scipy.optimize.least_squares()

model

Result.**model**
Return the model used to fit result.

Returns The model instance.

Return type *Model*

number_of_data_points

Result.**number_of_data_points: int | None = None**
Number of data points N.

number_of_jacobian_evaluations

Result.**number_of_jacobian_evaluations**: int | None = None
The number of jacobian evaluations.

number_of_variables

Result.**number_of_variables**: int | None = None
Number of variables in optimization N_{vars}

optimality

Result.**optimality**: float | None = None

reduced_chi_square

Result.**reduced_chi_square**: float | None = None
The reduced chi-square of the optimization.

$\chi^2_{red} = \chi^2/(N - N_{vars})$.

root_mean_square_error

Result.**root_mean_square_error**: float | None = None
The root mean square error the optimization.

$rms = \sqrt{\chi^2_{red}}$

source_path

Result.**source_path**: StrOrPath = 'result.yml'

number_of_function_evaluations

Result.**number_of_function_evaluations**: int
The number of function evaluations.

success

Result.**success**: bool
Indicates if the optimization was successful.

termination_reason

Result.**termination_reason**: str
The reason (message when) the optimizer terminated

glotaran_version

Result.**glotaran_version**: str
The glotaran version used to create the result.

free_parameter_labels

Result.**free_parameter_labels**: list[str]
List of labels of the free parameters used in optimization.

scheme

Result.**scheme**: Scheme

initial_parameters

Result.**initial_parameters**: ParameterGroup

optimized_parameters

Result.**optimized_parameters**: ParameterGroup

parameter_history

Result.**parameter_history**: ParameterHistory
The parameter history.

data

Result.**data**: Mapping[str, xr.Dataset]
The resulting data as a dictionary of xarray.Dataset.

Notes

The actual content of the data depends on the actual model and can be found in the documentation for the model.

Methods Summary

get_dataset	Return the result dataset for the given dataset label.
get_scheme	Return a new scheme from the Result object with optimized parameters.
loader	Create a *Result* instance from the specs defined in a file.
markdown	Format the model as a markdown text.
recreate	Recrate a result from the initial parameters.
save	Save the result to given folder.
verify	Verify a result.

get_dataset

`Result.`**`get_dataset`**`(`*dataset_label: str*`)` → xarray.core.dataset.Dataset

Return the result dataset for the given dataset label.

> **Warning:** Deprecated use `glotaran.project.result.Result.data[dataset_label]` instead.

Parameters **`dataset_label`** (*str*) – The label of the dataset.

Returns The dataset.

Return type xr.Dataset

get_scheme

`Result.`**`get_scheme`**`()` → *glotaran.project.scheme.Scheme*

Return a new scheme from the Result object with optimized parameters.

Returns A new scheme with the parameters set to the optimized values. For the dataset weights the (precomputed) weights from the original scheme are used.

Return type *Scheme*

loader

Result.**loader**(*format_name: str = None*, ***kwargs: Any*) → Result
Create a `Result` instance from the specs defined in a file.

Parameters

- **result_path** (*StrOrPath*) – Path containing the result data.
- **format_name** (*str*) – Format the result is in, if not provided and it is a file it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_result` implementation of the project io plugin.

Returns `Result` instance created from the saved format.

Return type *Result*

markdown

Result.**markdown**(*with_model: bool = True*, *base_heading_level: int = 1*) → *glotaran.utils.ipython.MarkdownStr*
Format the model as a markdown text.

Parameters

- **with_model** (*bool*) – If *True*, the model will be printed with initial and optimized parameters filled in.
- **base_heading_level** (*int*) – The level of the base heading.

Returns **MarkdownStr** – The scheme as markdown string.

Return type str

recreate

Result.**recreate**() → *glotaran.project.result.Result*
Recrate a result from the initial parameters.

Returns The recreated result.

Return type *Result*

save

Result.**save**(*path: str*) → list[str]
Save the result to given folder.

Parameters **path** (*str*) – The path to the folder in which to save the result.

Returns Paths to all the saved files.

Return type list[str]

verify

Result.**verify**() → bool
Verify a result.

Returns Weather the recreated result is equal to this result.

Return type bool

Methods Documentation

additional_penalty: np.ndarray | None = None
A vector with the value for each additional penalty, or None

chi_square: float | None = None
The chi-square of the optimization.

$\chi^2 = \sum_i^N [Residual_i]^2$.

cost: ArrayLike | None = None
The final cost.

covariance_matrix: ArrayLike | None = None
Covariance matrix.

The rows and columns are corresponding to *free_parameter_labels*.

data: Mapping[str, xr.Dataset]
The resulting data as a dictionary of xarray.Dataset.

Notes

The actual content of the data depends on the actual model and can be found in the documentation for the model.

degrees_of_freedom: int | None = None
Degrees of freedom in optimization $N - N_{vars}$.

free_parameter_labels: list[str]
List of labels of the free parameters used in optimization.

get_dataset(*dataset_label: str*) → xarray.core.dataset.Dataset
Return the result dataset for the given dataset label.

> **Warning:** Deprecated use `glotaran.project.result.Result.data[dataset_label]` instead.

Parameters **dataset_label** (*str*) – The label of the dataset.

Returns The dataset.

Return type xr.Dataset

get_scheme() → *glotaran.project.scheme.Scheme*
Return a new scheme from the Result object with optimized parameters.

Returns A new scheme with the parameters set to the optimized values. For the dataset weights the (precomputed) weights from the original scheme are used.

Return type *Scheme*

glotaran_version: str
The glotaran version used to create the result.

initial_parameters: ParameterGroup

jacobian: ArrayLike | list | None = None
Modified Jacobian matrix at the solution

See also: `scipy.optimize.least_squares()`

loader(*format_name: str = None*, ***kwargs: Any*) → *Result*
Create a `Result` instance from the specs defined in a file.

Parameters

- **result_path** (*StrOrPath*) – Path containing the result data.
- **format_name** (*str*) – Format the result is in, if not provided and it is a file it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_result` implementation of the project io plugin.

Returns `Result` instance created from the saved format.

Return type *Result*

markdown(*with_model: bool = True*, *base_heading_level: int = 1*) → *glotaran.utils.ipython.MarkdownStr*
Format the model as a markdown text.

Parameters

- **with_model** (*bool*) – If *True*, the model will be printed with initial and optimized parameters filled in.
- **base_heading_level** (*int*) – The level of the base heading.

Returns **MarkdownStr** – The scheme as markdown string.

Return type str

property model: *glotaran.model.model.Model*
Return the model used to fit result.

Returns The model instance.

Return type *Model*

number_of_data_points: int | None = None
Number of data points N.

number_of_function_evaluations: int
The number of function evaluations.

number_of_jacobian_evaluations: int | None = None
The number of jacobian evaluations.

number_of_variables: int | None = None
Number of variables in optimization N_{vars}

optimality: float | None = None

optimized_parameters: ParameterGroup

parameter_history: ParameterHistory
The parameter history.

recreate() → *glotaran.project.result.Result*
Recrate a result from the initial parameters.

Returns The recreated result.

Return type *Result*

reduced_chi_square: float | None = None
The reduced chi-square of the optimization.

$\chi^2_{red} = \chi^2/(N - N_{vars})$.

root_mean_square_error: float | None = None
The root mean square error the optimization.

$rms = \sqrt{\chi^2_{red}}$

save(*path: str*) → list[str]
Save the result to given folder.

Parameters **path** (*str*) – The path to the folder in which to save the result.

Returns Paths to all the saved files.

Return type list[str]

scheme: Scheme

source_path: StrOrPath = 'result.yml'

success: bool
Indicates if the optimization was successful.

termination_reason: str
The reason (message when) the optimizer terminated

verify() → bool
Verify a result.

Returns Weather the recreated result is equal to this result.

Return type bool

scheme

The module for :class:Scheme.

Classes

Summary

Scheme	A scheme is a collection of a model, parameters and a dataset.

Scheme

class glotaran.project.scheme.**Scheme**(*model: Model*, *parameters: ParameterGroup*, *data: Mapping[str, xr.Dataset]*, *clp_link_tolerance: float = 0.0*, *maximum_number_function_evaluations: int | None = None*, *non_negative_least_squares: bool | None = None*, *group_tolerance: float | None = None*, *group: bool | None = None*, *add_svd: bool = True*, *ftol: float = 1e-08*, *gtol: float = 1e-08*, *xtol: float = 1e-08*, *optimization_method: Literal['TrustRegionReflection', 'Dogbox', 'Levenberg-Marquardt'] = 'TrustRegionReflection'*, *result_path: str | None = None*)

Bases: object

A scheme is a collection of a model, parameters and a dataset.

A scheme also holds options for optimization.

Attributes Summary

add_svd	
clp_link_tolerance	
ftol	
global_dimensions	Return the dataset model's global dimension.
group	
group_tolerance	
gtol	
maximum_number_function_evaluations	
model_dimensions	Return the dataset model's model dimension.
non_negative_least_squares	
optimization_method	
result_path	
source_path	
xtol	
model	
parameters	
data	

add_svd

Scheme.**add_svd**: bool = True

clp_link_tolerance

Scheme.**clp_link_tolerance**: float = 0.0

ftol

Scheme.**ftol**: float = 1e-08

global_dimensions

Scheme.**global_dimensions**
Return the dataset model's global dimension.

Returns A dictionary with the dataset labels as key and the global dimension of the dataset as value.

Return type dict[str, str]

group

Scheme.**group**: bool | None = None

group_tolerance

Scheme.**group_tolerance**: float | None = None

gtol

Scheme.**gtol**: float = 1e-08

maximum_number_function_evaluations

Scheme.**maximum_number_function_evaluations**: int | None = None

model_dimensions

Scheme.**model_dimensions**
Return the dataset model's model dimension.

Returns A dictionary with the dataset labels as key and the model dimension of the dataset as value.

Return type dict[str, str]

non_negative_least_squares

Scheme.**non_negative_least_squares: bool | None = None**

optimization_method

Scheme.**optimization_method: Literal['TrustRegionReflection', 'Dogbox', 'Levenberg-Marquardt'] = 'TrustRegionReflection'**

result_path

Scheme.**result_path: str | None = None**

source_path

Scheme.**source_path: StrOrPath = 'scheme.yml'**

xtol

Scheme.**xtol: float = 1e-08**

model

Scheme.**model: Model**

parameters

Scheme.**parameters: ParameterGroup**

data

Scheme.**data: Mapping[str, xr.Dataset]**

Methods Summary

from_yaml_file	Create *Scheme* from specs in yaml file.
loader	Create a *Scheme* instance from the specs defined in a file.
markdown	Format the *Scheme* as markdown string.
problem_list	Return a list with all problems in the model and missing parameters.
valid	Check if there are no problems with the model or the parameters.
validate	Return a string listing all problems in the model and missing parameters.

from_yaml_file

static Scheme.**from_yaml_file**(*filename: str*) → *glotaran.project.scheme.Scheme*

Create *Scheme* from specs in yaml file.

> **Warning:** Deprecated use `glotaran.io.load_scheme(filename)` instead.

Parameters **filename** (*str*) – Path to the spec file.

Returns Analysis schmeme

Return type *Scheme*

loader

Scheme.**loader**(*format_name: str = None*, ***kwargs: Any*) → Scheme

Create a *Scheme* instance from the specs defined in a file.

Parameters

- **file_name** (*StrOrPath*) – File containing the parameter specs.
- **format_name** (*str*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_scheme` implementation of the project io plugin.

Returns *Scheme* instance created from the file.

Return type *Scheme*

markdown

Scheme.**markdown**()
Format the *Scheme* as markdown string.

Returns The scheme as markdown string.

Return type *MarkdownStr*

problem_list

Scheme.**problem_list**() → list[str]
Return a list with all problems in the model and missing parameters.

Returns A list of all problems found in the scheme's model.

Return type list[str]

valid

Scheme.**valid**() → bool
Check if there are no problems with the model or the parameters.

Returns Whether the scheme is valid.

Return type bool

validate

Scheme.**validate**() → str
Return a string listing all problems in the model and missing parameters.

Returns A user-friendly string containing all the problems of a model if any. Defaults to 'Your model is valid.' if no problems are found.

Return type str

Methods Documentation

add_svd: bool = True

clp_link_tolerance: float = 0.0

data: Mapping[str, xr.Dataset]

static from_yaml_file(*filename: str*) → *glotaran.project.scheme.Scheme*
Create *Scheme* from specs in yaml file.

> **Warning:** Deprecated use `glotaran.io.load_scheme(filename)` instead.

Parameters **filename** (*str*) – Path to the spec file.

Returns Analysis schmeme

Return type *Scheme*

ftol: float = 1e-08

property global_dimensions: dict[str, str]
Return the dataset model's global dimension.

Returns A dictionary with the dataset labels as key and the global dimension of the dataset as value.

Return type dict[str, str]

group: bool | None = None

group_tolerance: float | None = None

gtol: float = 1e-08

loader(*format_name: str = None*, ***kwargs: Any*) → *Scheme*
Create a *Scheme* instance from the specs defined in a file.

Parameters

- **file_name** (*StrOrPath*) – File containing the parameter specs.
- **format_name** (*str*) – Format the file is in, if not provided it will be inferred from the file extension.
- ****kwargs** (*Any*) – Additional keyword arguments passes to the `load_scheme` implementation of the project io plugin.

Returns *Scheme* instance created from the file.

Return type *Scheme*

markdown()
Format the *Scheme* as markdown string.

Returns The scheme as markdown string.

Return type *MarkdownStr*

maximum_number_function_evaluations: int | None = None

model: Model

property model_dimensions: dict[str, str]
Return the dataset model's model dimension.

Returns A dictionary with the dataset labels as key and the model dimension of the dataset as value.

Return type dict[str, str]

non_negative_least_squares: bool | None = None

optimization_method: Literal['TrustRegionReflection', 'Dogbox', 'Levenberg-Marquardt'] = 'TrustRegionReflection'

parameters: ParameterGroup

problem_list() → list[str]
Return a list with all problems in the model and missing parameters.

Returns A list of all problems found in the scheme's model.

Return type list[str]

result_path: str | None = None

source_path: StrOrPath = 'scheme.yml'

valid() → bool
Check if there are no problems with the model or the parameters.

Returns Whether the scheme is valid.

Return type bool

validate() → str
Return a string listing all problems in the model and missing parameters.

Returns A user-friendly string containing all the problems of a model if any. Defaults to 'Your model is valid.' if no problems are found.

Return type str

xtol: float = 1e-08

15.1.11 testing

Testing framework package for glotaran itself and plugins.

Modules

glotaran.testing.model_generators	Model generators used to generate simple models from a set of inputs.
glotaran.testing.plugin_system	Mock functionality for the plugin system.

model_generators

Model generators used to generate simple models from a set of inputs.

Classes

Summary

SimpleModelGenerator	A minimal boilerplate model and parameters generator.

SimpleModelGenerator

class glotaran.testing.model_generators.**SimpleModelGenerator**(*rates: list[float] = <factory>, k_matrix: Literal[('parallel', 'sequential')] | dict[tuple[str, str], str] = 'parallel', compartments: list[str] | None = None, irf: dict[str, float] = <factory>, initial_concentration: list[float] = <factory>, dispersion_coefficients: list[float] = <factory>, dispersion_center: float | None = None, default_megacomplex: str = 'decay'*)

Bases: `object`

A minimal boilerplate model and parameters generator.

Generates a model (together with the parameters specification) based on parameter input values assigned to the generator's attributes

Attributes Summary

compartments	A list of compartment names
default_megacomplex	The default_megacomplex identifier
dispersion_center	A value representing the dispersion center
k_matrix	"A *dict* with a k_matrix specification or *Literal["parallel", "sequential"]*
model	Return the generated model.
model_and_parameters	Return generated model and parameters.
model_dict	Return a dict representation of the generated model.
parameters	Return the generated parameters of type `glotaran.parameter.ParameterGroup`.
parameters_dict	Return a dict representation of the generated parameters.
valid	Check if the generator state is valid.
rates	A list of values representing decay rates
irf	A dict of items specifying an irf
initial_concentration	A list values representing the initial concentration
dispersion_coefficients	A list of values representing the dispersion coefficients

compartments

SimpleModelGenerator.**compartments: list[str] | None = None**
A list of compartment names

default_megacomplex

SimpleModelGenerator.**default_megacomplex: str = 'decay'**
The default_megacomplex identifier

dispersion_center

SimpleModelGenerator.**dispersion_center: float | None = None**
A value representing the dispersion center

k_matrix

SimpleModelGenerator.**k_matrix: Literal['parallel', 'sequential'] | dict[tuple[str, str], str] = 'parallel'**
"A *dict* with a k_matrix specification or *Literal["parallel", "sequential"]*

model

SimpleModelGenerator.**model**
Return the generated model.

Returns The generated model of type `glotaran.model.Model`.

Return type *Model*

model_and_parameters

SimpleModelGenerator.**model_and_parameters**
Return generated model and parameters.

Returns A model of type `glotaran.model.Model` and and parameters of type `glotaran.parameter.ParameterGroup`.

Return type tuple[*Model*, *ParameterGroup*]

model_dict

SimpleModelGenerator.**model_dict**
Return a dict representation of the generated model.

Returns A dict representation of the generated model.

Return type dict

parameters

SimpleModelGenerator.**parameters**
Return the generated parameters of type glotaran.parameter.ParameterGroup.

Returns The generated parameters of type of type glotaran.parameter.ParameterGroup.

Return type *ParameterGroup*

parameters_dict

SimpleModelGenerator.**parameters_dict**
Return a dict representation of the generated parameters.

Returns A dict representing the generated parameters.

Return type dict

valid

SimpleModelGenerator.**valid**
Check if the generator state is valid.

Returns Generator state obtained by calling the generated model's *valid* function with the generated parameters as input.

Return type bool

rates

SimpleModelGenerator.**rates: list[float]**
A list of values representing decay rates

irf

SimpleModelGenerator.**irf: dict[str, float]**
A dict of items specifying an irf

initial_concentration

SimpleModelGenerator.**initial_concentration: list[float]**
A list values representing the initial concentration

dispersion_coefficients

SimpleModelGenerator.**dispersion_coefficients: list[float]**
A list of values representing the dispersion coefficients

Methods Summary

markdown	Return a markdown string representation of the generated model and parameters.
validate	Call *validate* on the generated model and return its output.

markdown

SimpleModelGenerator.**markdown**() → MarkdownStr
Return a markdown string representation of the generated model and parameters.

Returns A markdown string

Return type *MarkdownStr*

validate

SimpleModelGenerator.**validate**() → str
Call *validate* on the generated model and return its output.

Returns A string listing problems in the generated model and parameters if any.

Return type str

Methods Documentation

compartments: list[str] | None = None
A list of compartment names

default_megacomplex: str = 'decay'
The default_megacomplex identifier

dispersion_center: float | None = None
A value representing the dispersion center

dispersion_coefficients: list[float]
A list of values representing the dispersion coefficients

initial_concentration: list[float]
A list values representing the initial concentration

irf: dict[str, float]
A dict of items specifying an irf

k_matrix: Literal['parallel', 'sequential'] | dict[tuple[str, str], str] = 'parallel'
"A *dict* with a k_matrix specification or *Literal["parallel", "sequential"]*

markdown() → MarkdownStr
Return a markdown string representation of the generated model and parameters.

Returns A markdown string

Return type *MarkdownStr*

property model: *glotaran.model.model.Model*
Return the generated model.

Returns The generated model of type `glotaran.model.Model`.

Return type *Model*

property model_and_parameters: tuple[Model, ParameterGroup]
Return generated model and parameters.

Returns A model of type `glotaran.model.Model` and and parameters of type `glotaran.parameter.ParameterGroup`.

Return type tuple[*Model*, *ParameterGroup*]

property model_dict: dict
Return a dict representation of the generated model.

Returns A dict representation of the generated model.

Return type dict

property parameters: *glotaran.parameter.parameter_group.ParameterGroup*
Return the generated parameters of type `glotaran.parameter.ParameterGroup`.

Returns The generated parameters of type of type `glotaran.parameter.ParameterGroup`.

Return type *ParameterGroup*

property parameters_dict: dict
Return a dict representation of the generated parameters.

Returns A dict representing the generated parameters.

Return type dict

rates: list[float]
A list of values representing decay rates

property valid: bool
Check if the generator state is valid.

Returns Generator state obtained by calling the generated model's *valid* function with the generated parameters as input.

Return type bool

validate() → str
Call *validate* on the generated model and return its output.

Returns A string listing problems in the generated model and parameters if any.

Return type str

plugin_system

Mock functionality for the plugin system.

Functions

Summary

monkeypatch_plugin_registry	Contextmanager to monkeypatch multiple plugin registries at once.
monkeypatch_plugin_registry_data_io	Monkeypatch the `DataIoInterface` registry.
monkeypatch_plugin_registry_megacomplex	Monkeypatch the `Megacomplex` registry.
monkeypatch_plugin_registry_project_io	Monkeypatch the `ProjectIoInterface` registry.

monkeypatch_plugin_registry

glotaran.testing.plugin_system.**monkeypatch_plugin_registry**(*, *test_megacomplex: MutableMapping[str, type[Megacomplex]] | None = None*, *test_data_io: MutableMapping[str, DataIoInterface] | None = None*, *test_project_io: MutableMapping[str, ProjectIoInterface] | None = None*, *create_new_registry: bool = False*) → Generator[None, None, None]

Contextmanager to monkeypatch multiple plugin registries at once.

Parameters

- **test_megacomplex** (*MutableMapping[str, type[Megacomplex]], optional*) – Registry to to update or replace the `Megacomplex` registry with. , by default None
- **test_data_io** (*MutableMapping[str, DataIoInterface], optional*) – Registry to to update or replace the `DataIoInterface` registry with. , by default None
- **test_project_io** (*MutableMapping[str, ProjectIoInterface], optional*) – Registry to to update or replace the `ProjectIoInterface` registry with. , by default None
- **create_new_registry** (*bool*) – Whether to update the actual registry or create a new one from the arguments. , by default False

Yields *Generator[None, None, None]* – Just keeps all context manager alive

See also:

monkeypatch_plugin_registry_megacomplex, *monkeypatch_plugin_registry_data_io*, *monkeypatch_plugin_registry_project_io*

monkeypatch_plugin_registry_data_io

glotaran.testing.plugin_system.**monkeypatch_plugin_registry_data_io**(*test_data_io: MutableMapping[str, DataIoInterface] | None = None*, *create_new_registry: bool = False*) → Generator[None, None, None]

Monkeypatch the `DataIoInterface` registry.

Parameters

- **test_data_io** (*MutableMapping[str, DataIoInterface], optional*) – Registry to to update or replace the `DataIoInterface` registry with. , by default None
- **create_new_registry** (*bool*) – Whether to update the actual registry or create a new one from `test_data_io` , by default False

Yields *Generator[None, None, None]* – Just to keep the context alive.

monkeypatch_plugin_registry_megacomplex

glotaran.testing.plugin_system.**monkeypatch_plugin_registry_megacomplex**(*test_megacomplex: MutableMapping[str, type[Megacomplex]] | None = None*, *create_new_registry: bool = False*) → Generator[None, None, None]

Monkeypatch the `Megacomplex` registry.

Parameters

- **test_megacomplex** (*MutableMapping[str, type[Megacomplex]], optional*) – Registry to to update or replace the `Megacomplex` registry with. , by default None
- **create_new_registry** (*bool*) – Whether to update the actual registry or create a new one from `test_megacomplex` , by default False

Yields *Generator[None, None, None]* – Just to keep the context alive.

monkeypatch_plugin_registry_project_io

glotaran.testing.plugin_system.**monkeypatch_plugin_registry_project_io**(*test_project_io: MutableMapping[str, ProjectIoInterface] | None = None*, *create_new_registry: bool = False*) → Generator[None, None, None]

Monkeypatch the `ProjectIoInterface` registry.

Parameters

- **test_project_io** (*MutableMapping[str, ProjectIoInterface], optional*) – Registry to to update or replace the `ProjectIoInterface` registry with. , by default None
- **create_new_registry** (*bool*) – Whether to update the actual registry or create a new one from `test_data_io` , by default False

Yields *Generator[None, None, None]* – Just to keep the context alive.

15.1.12 typing

Glotaran specific typing module.

Modules

glotaran.typing.protocols	Protocol like type definitions.
glotaran.typing.types	Glotaran types module containing commonly used types.

protocols

Protocol like type definitions.

Classes

Summary

FileLoadableProtocol

FileLoadableProtocol

class glotaran.typing.protocols.**FileLoadableProtocol**(**args*, ***kwargs*)
Bases: Protocol

Attributes Summary

loader
source_path

loader

FileLoadableProtocol.**loader: Callable[[StrOrPath | Sequence[StrOrPath] | Mapping[str, StrOrPath]], FileLoadableProtocol]**

source_path

FileLoadableProtocol.**source_path: StrOrPath | Sequence[StrOrPath] | Mapping[str, StrOrPath]**

Methods Summary

—

Methods Documentation

loader: Callable[[StrOrPath | Sequence[StrOrPath] | Mapping[str, StrOrPath]], *FileLoadableProtocol*]

source_path: StrOrPath | Sequence[StrOrPath] | Mapping[str, StrOrPath]

types

Glotaran types module containing commonly used types.

15.1.13 utils

Glotaran utility function/class package.

Modules

glotaran.utils.io	Glotaran IO utility module.
glotaran.utils.ipython	Glotaran module with utilities for `ipython` integration (e.g.
glotaran.utils.regex	Glotaran module with regular expression patterns and functions.
glotaran.utils.sanitize	Glotaran module with utilities for sanitation of parsed content.

io

Glotaran IO utility module.

Functions

Summary

load_datasets	Load multiple datasets into a mapping (convenience function).
relative_posix_path	Ensure that `source_path` is a posix path, relative to `base_path` if defined.

load_datasets

glotaran.utils.io.**load_datasets**(*dataset_mappable: Union[str, pathlib.Path, xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray, Sequence[Union[str, pathlib.Path, xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray]], Mapping[str, Union[str, pathlib.Path, xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray]]]*) → *glotaran.utils.io.DatasetMapping*

Load multiple datasets into a mapping (convenience function).

This is used for `file_loadable_field` of a dataset mapping e.g. in `Scheme`

Parameters **dataset_mappable** (*DatasetMappable*) – Single dataset/file path to a dataset or sequence or mapping of it.

Returns Mapping of dataset with string keys, where datasets hare ensured to have the `source_path` attr.

Return type *DatasetMapping*

relative_posix_path

glotaran.utils.io.**relative_posix_path**(*source_path: StrOrPath*, *base_path: StrOrPath | None = None*) → str

Ensure that `source_path` is a posix path, relative to `base_path` if defined.

Parameters

- **source_path** (*StrOrPath*) – Path which should be converted to a relative posix path.
- **base_path** (*StrOrPath, optional*) – Base path the resulting path string should be relative to., by default None

Returns `source_path` as posix path relative to `base_path` if defined.

Return type str

Classes

Summary

DatasetMapping	Wrapper class for a mapping of datasets which can be used for a `file_loadable_field`.

DatasetMapping

class glotaran.utils.io.**DatasetMapping**(*init_map: Mapping[str, xr.Dataset] = None*)
Bases: collections.abc.MutableMapping

Wrapper class for a mapping of datasets which can be used for a `file_loadable_field`.

Initialize an instance of *DatasetMapping*.

Parameters **init_dict** (*dict[str, xr.Dataset], optional*) – Mapping to initially populate the instance., by default None

Attributes Summary

source_path	Map the `source_path` attribute of each dataset to a standalone mapping.

source_path

DatasetMapping.**source_path**
Map the `source_path` attribute of each dataset to a standalone mapping.

Note: When the `source_path` attribute of the dataset gets updated (e.g. by calling `save_dataset` with the default `update_source_path=True`) this value will be updated as well.

Returns Mapping of the dataset source paths.

Return type Mapping[str, str]

Methods Summary

clear	
get	
items	
keys	
loader	Loader function utilized by `file_loadable_field`.
pop	If key is not found, d is returned if given, otherwise KeyError is raised.
popitem	as a 2-tuple; but raise KeyError if D is empty.
setdefault	

continues on next page

Table 209 – continued from previous page

update	If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v
values	

clear

DatasetMapping.**clear**() → None. Remove all items from D.

get

DatasetMapping.**get**(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items

DatasetMapping.**items**() → a set-like object providing a view on D's items

keys

DatasetMapping.**keys**() → a set-like object providing a view on D's keys

loader

classmethod DatasetMapping.**loader**(*dataset_mappable: DatasetMappable*) → DatasetMapping

Loader function utilized by `file_loadable_field`.

Parameters **dataset_mappable** (*DatasetMappable*) – Mapping of datasets to initialize *DatasetMapping*.

Returns Populated instance of *DatasetMapping*.

Return type *DatasetMapping*

pop

DatasetMapping.**pop**(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem

DatasetMapping.**popitem**() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault

DatasetMapping.**setdefault**(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update

DatasetMapping.**update**([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values

DatasetMapping.**values**() → an object providing a view on D's values

Methods Documentation

clear() → None. Remove all items from D.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

classmethod loader(*dataset_mappable: DatasetMappable*) → *DatasetMapping*
Loader function utilized by `file_loadable_field`.

Parameters **dataset_mappable** (*DatasetMappable*) – Mapping of datasets to initialize *DatasetMapping*.

Returns Populated instance of *DatasetMapping*.

Return type *DatasetMapping*

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

property source_path
Map the `source_path` attribute of each dataset to a standalone mapping.

Note: When the `source_path` attribute of the dataset gets updated (e.g. by calling `save_dataset` with the default `update_source_path=True`) this value will be updated as well.

Returns Mapping of the dataset source paths.

Return type Mapping[str, str]

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

ipython

Glotaran module with utilities for `ipython` integration (e.g. `notebooks`).

Functions

Summary

display_file	Display a file with syntax highlighting `syntax`.

display_file

glotaran.utils.ipython.**display_file**(*path: str | PathLike[str]*, *, *syntax: str = None*) → MarkdownStr

Display a file with syntax highlighting `syntax`.

Parameters

- **path** (*str | PathLike[str]*) – Paths to the file
- **syntax** (*str*) – Syntax highlighting which should be applied, by default None

Returns File content with syntax highlighting to render in ipython.

Return type *MarkdownStr*

Classes

Summary

MarkdownStr	String wrapper class for rich display integration of markdown in ipython.

MarkdownStr

class glotaran.utils.ipython.**MarkdownStr**(*wrapped_str: str*, *, *syntax: Optional[str] = None*)

Bases: collections.UserString

String wrapper class for rich display integration of markdown in ipython.

Initialize string class that is automatically displayed as markdown by `ipython`.

Parameters

- **wrapped_str** (*str*) – String to be wrapped.
- **syntax** (*str*) – Syntax highlighting which should be applied, by default None

Note: Possible syntax highlighting values can e.g. be found here: https://support.codebasehq.com/articles/tips-tricks/syntax-highlighting-in-markdown

Methods Summary

capitalize	
casefold	
center	
count	
encode	
endswith	
expandtabs	
find	
format	
format_map	
index	Raises ValueError if the value is not present.
isalnum	

continues on next page

Table 212 – continued from previous page

isalpha	
isascii	
isdecimal	
isdigit	
isidentifier	
islower	
isnumeric	
isprintable	
isspace	
istitle	
isupper	
join	
ljust	
lower	
lstrip	
maketrans	Return a translation table usable for str.translate().
partition	
replace	
rfind	
rindex	
rjust	
rpartition	
rsplit	
rstrip	
split	

continues on next page

Table 212 – continued from previous page

splitlines
startswith
strip
swapcase
title
translate
upper
zfill

capitalize

MarkdownStr.**capitalize**()

casefold

MarkdownStr.**casefold**()

center

MarkdownStr.**center**(*width*, **args*)

count

MarkdownStr.**count**(*value*) → integer -- return number of occurrences of value

encode

MarkdownStr.**encode**(*encoding='utf-8'*, *errors='strict'*)

endswith

MarkdownStr.**endswith**(*suffix*, *start=0*, *end=9223372036854775807*)

expandtabs

MarkdownStr.**expandtabs**(*tabsize=8*)

find

MarkdownStr.**find**(*sub*, *start=0*, *end=9223372036854775807*)

format

MarkdownStr.**format**(**args*, ***kwds*)

format_map

MarkdownStr.**format_map**(*mapping*)

index

MarkdownStr.**index**(*value*[, *start*[, *stop*]]) → integer -- return first index of value.
Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

isalnum

MarkdownStr.**isalnum**()

isalpha

MarkdownStr.**isalpha**()

isascii

MarkdownStr.**isascii**()

isdecimal

MarkdownStr.**isdecimal**()

isdigit

MarkdownStr.**isdigit**()

isidentifier

MarkdownStr.**isidentifier**()

islower

MarkdownStr.**islower**()

isnumeric

MarkdownStr.**isnumeric**()

isprintable

MarkdownStr.**isprintable**()

isspace

MarkdownStr.**isspace**()

istitle

MarkdownStr.**istitle**()

isupper

MarkdownStr.**isupper**()

join

MarkdownStr.**join**(*seq*)

ljust

MarkdownStr.**ljust**(*width*, **args*)

lower

MarkdownStr.**lower**()

lstrip

MarkdownStr.**lstrip**(*chars=None*)

maketrans

MarkdownStr.**maketrans**(*x*, *y=<unrepresentable>*, *z=<unrepresentable>*, */*)

Return a translation table usable for str.translate().

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition

MarkdownStr.**partition**(*sep*)

replace

MarkdownStr.**replace**(*old*, *new*, *maxsplit=- 1*)

rfind

MarkdownStr.**rfind**(*sub*, *start=0*, *end=9223372036854775807*)

rindex

MarkdownStr.**rindex**(*sub*, *start=0*, *end=9223372036854775807*)

rjust

MarkdownStr.**rjust**(*width*, **args*)

rpartition

MarkdownStr.**rpartition**(*sep*)

rsplit

MarkdownStr.**rsplit**(*sep=None*, *maxsplit=- 1*)

rstrip

MarkdownStr.**rstrip**(*chars=None*)

split

MarkdownStr.**split**(*sep=None*, *maxsplit=- 1*)

splitlines

MarkdownStr.**splitlines**(*keepends=False*)

startswith

MarkdownStr.**startswith**(*prefix*, *start=0*, *end=9223372036854775807*)

strip

MarkdownStr.**strip**(*chars=None*)

swapcase

MarkdownStr.**swapcase**()

title

MarkdownStr.**title**()

translate

MarkdownStr.**translate**(**args*)

upper

MarkdownStr.**upper**()

zfill

MarkdownStr.**zfill**(*width*)

Methods Documentation

capitalize()

casefold()

center(*width*, **args*)

count(*value*) → integer -- return number of occurrences of value

encode(*encoding='utf-8'*, *errors='strict'*)

endswith(*suffix*, *start=0*, *end=9223372036854775807*)

expandtabs(*tabsize=8*)

find(*sub*, *start=0*, *end=9223372036854775807*)

format(**args*, ***kwds*)

format_map(*mapping*)

index(*value*[, *start*[, *stop*]]) → integer -- return first index of value.
Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

isalnum()

isalpha()

isascii()

isdecimal()

isdigit()

isidentifier()

islower()

isnumeric()

isprintable()

isspace()

istitle()

isupper()

join(*seq*)

ljust(*width*, **args*)

lower()

lstrip(*chars=None*)

maketrans(*x*, *y=<unrepresentable>*, *z=<unrepresentable>*, */*)
Return a translation table usable for str.translate().

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(*sep*)

replace(*old*, *new*, *maxsplit=- 1*)

rfind(*sub*, *start=0*, *end=9223372036854775807*)

rindex(*sub*, *start=0*, *end=9223372036854775807*)

rjust(*width*, **args*)

rpartition(*sep*)

rsplit(*sep=None*, *maxsplit=- 1*)

rstrip(*chars=None*)

split(*sep=None*, *maxsplit=- 1*)

splitlines(*keepends=False*)

startswith(*prefix*, *start=0*, *end=9223372036854775807*)

strip(*chars=None*)

swapcase()

title()

translate(**args*)

upper()

zfill(*width*)

regex

Glotaran module with regular expression patterns and functions.

Classes

Summary

RegexPattern	An 'Enum' of (compiled) regular expression patterns (rp).

RegexPattern

class glotaran.utils.regex.**RegexPattern**

Bases: object

An 'Enum' of (compiled) regular expression patterns (rp).

Attributes Summary

elements_in_string_of_list
group
list_with_tuples
number

continues on next page

Table 214 – continued from previous page

number_scientific
tuple_number
tuple_word
word

elements_in_string_of_list

RegexPattern.**elements_in_string_of_list: re.Pattern** = **re.compile('(\\(.+?\\)|[-+.\\d]+)')**

group

RegexPattern.**group: re.Pattern = re.compile('(\\(.+?\\))')**

list_with_tuples

RegexPattern.**list_with_tuples: re.Pattern** = **re.compile('(\\[.+\\(.+\\).+\\])')**

number

RegexPattern.**number: re.Pattern = re.compile('[\\d.+-]+')**

number_scientific

RegexPattern.**number_scientific: re.Pattern** = **re.compile('[-+]?[0-9]*\\.?[0-9]+([eE][-+]?[0-9]+)')**

tuple_number

RegexPattern.**tuple_number: re.Pattern** = **re.compile('(\\([\\s\\d.+-]+?[,\\s\\d.+-]*?\\))')**

tuple_word

```
RegexPattern.tuple_word: re.Pattern =
re.compile('(\\([.\\s\\w\\d]+?[,.\\s\\w\\d]*?\\))')
```

word

```
RegexPattern.word: re.Pattern = re.compile('[\\w]+')
```

Methods Summary

—

Methods Documentation

```
elements_in_string_of_list: re.Pattern =
re.compile('(\\(.+?\\)|[-+.\\d]+)')

group: re.Pattern = re.compile('(\\(.+?\\))')

list_with_tuples: re.Pattern = re.compile('(\\[.+\\(.+\\).+\\])')

number: re.Pattern = re.compile('[\\d.+-]+')

number_scientific: re.Pattern =
re.compile('[-+]?[0-9]*\\.?[0-9]+([eE][-+]?[0-9]+)')

tuple_number: re.Pattern =
re.compile('(\\([\\s\\d.+-]+?[,\\s\\d.+-]*?\\))')

tuple_word: re.Pattern =
re.compile('(\\([.\\s\\w\\d]+?[,.\\s\\w\\d]*?\\))')

word: re.Pattern = re.compile('[\\w]+')
```

sanitize

Glotaran module with utilities for sanitation of parsed content.

Functions

Summary

convert_scientific_to_float	Convert value to float if it matches scientific notation string.
list_string_to_tuple	Convert a list of strings (representing tuples) to a list of tuples.
sanitize_dict_keys	Sanitize the stringified tuple dict keys in a yaml parsed dict.
sanitize_dict_values	Sanitizes a dict with broken tuples inside modifying it in-place.

continues on next page

Table 216 – continued from previous page

sanitize_list_with_broken_tuples	Sanitize a list with 'broken' tuples.
sanitize_parameter_list	Replace in a list strings matching scientific notation with floats.
sanitize_yaml	Sanitize a yaml-returned dict for key or (list) values containing tuples.
sanity_scientific_notation_conversion	Convert scientific notation string values to floats.
string_to_tuple	Convert a string to a tuple if it matches a tuple pattern.

convert_scientific_to_float

glotaran.utils.sanitize.**convert_scientific_to_float**(*value: str*) → float | str

Convert value to float if it matches scientific notation string.

Parameters **value** (*str*) – value to convert from string to float if it matches scientific notation

Returns return float if value was scientific notation string, else turn original value

Return type float | string

list_string_to_tuple

glotaran.utils.sanitize.**list_string_to_tuple**(*a_list: list[str]*) → list[tuple[float, ...] | tuple[str, ...] | float | str]

Convert a list of strings (representing tuples) to a list of tuples.

Parameters **a_list** (*List[str]*) – A list of strings, some of them representing (numbered) tuples

Returns A list of the (numbered) tuples represted by the incoming a_list

Return type List[Union[float, str]]

sanitize_dict_keys

glotaran.utils.sanitize.**sanitize_dict_keys**(*d: dict*) → dict

Sanitize the stringified tuple dict keys in a yaml parsed dict.

Keys representing a tuple, e.g. '(s1, s2)' are converted to a tuple of strings e.g. ('s1', 's2')

Parameters **d** (*dict*) – A dict containing tuple-like string keys

Returns A dict with tuple-like string keys converted to tuple keys

Return type dict

sanitize_dict_values

glotaran.utils.sanitize.**sanitize_dict_values**(*d: dict[str, Any] | list[Any]*)

Sanitizes a dict with broken tuples inside modifying it in-place.

Broken tuples are tuples that are turned into strings by the yaml parser. This functions calls *sanitize_list_with_broken_tuples* to glue the broken strings together and then calls list_to_tuple to turn the list with tuple strings back to number tuples.

Parameters **d** (*dict*) – A (complex) dict containing (possibly nested) values of broken tuple strings.

sanitize_list_with_broken_tuples

glotaran.utils.sanitize.**sanitize_list_with_broken_tuples**(*mangled_list: list[str | float]*) → list[str]

Sanitize a list with 'broken' tuples.

A list of broken tuples as returned by yaml when parsing tuples. e.g parsing the list of tuples [(3,100), (4,200)] results in a list of str ['(3', '100)', '(4', '200)'] which can be restored to a list with the tuples restored as strings ['(3, 100)', '(4, 200)']

Parameters **mangled_list** (*List[Union[str, float]]*) – A list with strings representing tuples broken up by round brackets.

Returns A list containing the restores tuples (in string form) which can be converted back to numbered tuples using *list_string_to_tuple*

Return type List[str]

sanitize_parameter_list

glotaran.utils.sanitize.**sanitize_parameter_list**(*parameter_list: list[str | float]*) → list[str | float]

Replace in a list strings matching scientific notation with floats.

Parameters **parameter_list** (*list*) – A list of parameters where some elements may be strings like 1E7

Returns A list where strings matching a scientific number have been converted to float

Return type list

sanitize_yaml

glotaran.utils.sanitize.**sanitize_yaml**(*d: dict*, *do_keys: bool = True*, *do_values: bool = False*) → dict

Sanitize a yaml-returned dict for key or (list) values containing tuples.

Parameters

- **d** (*dict*) – a dict resulting from parsing a pyglotaran model spec yml file
- **do_keys** (*bool*) – toggle sanitization of dict keys, by default True
- **do_values** (*bool*) – toggle sanitization of dict values, by default False

Returns a sanitized dict with (broken) string tuples restored as proper tuples

Return type dict

sanity_scientific_notation_conversion

glotaran.utils.sanitize.**sanity_scientific_notation_conversion**(*d: dict[str, Any] | list[Any]*)

Convert scientific notation string values to floats.

Parameters **d** (*dict[str, Any] | list[Any]*) – Iterable which should be checked for scientific notation values.

string_to_tuple

glotaran.utils.sanitize.**string_to_tuple**(*tuple_str: str*, *from_list=False*) → tuple[float, ...] | tuple[str, ...] | float | str

Convert a string to a tuple if it matches a tuple pattern.

Parameters

- **tuple_str** (*str*) – A string representing some tuple to convert the numbers inside the string tuple are mapped to float
- **from_list** (*bool, optional*) – only if true will a single number string be converted to float, otherwise returned as-is since it may represent a label, by default False

Returns Returns the tuple intended by the string

Return type tuple[float], tuple[str], float, str

CHAPTER

SIXTEEN

PLUGIN DEVELOPMENT

If you don't find the plugin that fits your needs you can always write your own. This sections will explain you how and what you need to know.

In time we will also provide you with a cookiecutter template, to kickstart your new plugin for publishing as a package on PyPi.

The following section was generated from `docs/source/notebooks/plugin_system/plugin_howto_write_a_io_plugin.ipynb`

16.1 How to Write your own Io plugin

There are all kinds of different data formats, so it is quite likely that your experimental setup uses a format which isn't yet supported by a `glotaran` plugin and want to write your own `DataIo` plugin to support this format.

Since `json` is very common format (admittedly not for data, but in general) and python has builtin support for it we will use it as an example.

First let's have a look which `DataIo` plugins are already installed and which functions they support.

[1]:
```
from glotaran.io import data_io_plugin_table
```

[2]:
```
data_io_plugin_table()
```

[2]:

Format name	load_dataset	save_dataset
ascii	*	*
nc	*	*
sdt	*	/

Looks like there isn't a `json` plugin installed yet, but maybe someone else did already write one, so have a look at the `` `3rd party plugins `` list in the user docsumentation <https://pyglotaran.readthedocs.io/en/latest/user_documentation/using_plugins.html>`__ before you start writing your own plugin.

For the sake of the example, we will write our `json` plugin even if there already exists one by the time you read this.

First you need to import all needed libraries and functions.

- `from __future__ import annotations`: needed to write python 3.10 typing syntax (`|`), even with a lower python version
- `json`,`xarray`: Needed for reading and writing itself
- `DataIoInterface`: needed to subclass from, this way you get the proper type and especially signature checking
- `register_data_io`: registers the DataIo plugin under the given `format_name`s

[3]:
```
from __future__ import annotations

import json

import xarray as xr

from glotaran.io.interface import DataIoInterface
from glotaran.plugin_system.data_io_registration import register_data_io
```

`DataIoInterface` has two methods we could implement `load_dataset` and `save_dataset`, which are used by the identically named functions in `glotaran.io`.

We will just implement both for our example to be complete. the quickest way to get started is to just copy over the code from `DataIoInterface` which already has the right signatures and some boilerplate docstrings, for the method arguments.

If the default arguments aren't enough for your plugin and you need your methods to have additional option, you can just add those. Note the `*` between `file_name` and `my_extra_option`, this tell python that `my_extra_option` is an keyword only argument and `mypy <https://github.com/python/mypy>`__ won't raise an `[override]` type error for changing the signature of the method. To help others who might use your plugin and your future self, it is good practice to documents what each parameter does in the methods docstring, which will be accessed by the help function.

Finally add the `@register_data_io` with the `format_name`'s you want to register the plugin to, in our case `json` and `my_json`.

Pro tip: You don't need to implement the whole functionality inside of the method itself,

[4]:
```
@register_data_io(["json", "my_json"])
class JsonDataIo(DataIoInterface):
    """My new shiny glotaran plugin for json data io"""

    def load_dataset(
        self, file_name: str, *, my_extra_option: str = None
    ) -> xr.Dataset | xr.DataArray:
        """Read json data to xarray.Dataset


        Parameters
        ----------
        file_name : str
            File containing the data.
        my_extra_option: str
            This argument is only for demonstration
        """
        if my_extra_option is not None:
            print(f"Using my extra option loading json: {my_extra_option}")

        with open(file_name) as json_file:
            data_dict = json.load(json_file)
        return xr.Dataset.from_dict(data_dict)

    def save_dataset(
        self, dataset: xr.Dataset | xr.DataArray, file_name: str, *, my_extra_option=None
    ):
        """Write xarray.Dataset to a json file
```

(continues on next page)

(continued from previous page)

```
        Parameters
        ----------
        dataset : xr.Dataset
            Dataset to be saved to file.
        file_name : str
            File to write the result data to.
        my_extra_option: str
            This argument is only for demonstration
        """
        if my_extra_option is not None:
            print(f"Using my extra option for writing json: {my_extra_option}")

        data_dict = dataset.to_dict()
        with open(file_name, "w") as json_file:
            json.dump(data_dict, json_file)
```

Let's verify that our new plugin was registered successfully under the `format_names` `json` and `my_json`.

[5]:
```
data_io_plugin_table()
```

[5]:

Format name	load_dataset	save_dataset
ascii	*	*
json	*	*
my_json	*	*
nc	*	*
sdt	*	/

Now let's use the example data from the quickstart to test the reading and writing capabilities of our plugin.

[6]:
```
from glotaran.examples.sequential import dataset
from glotaran.io import load_dataset
from glotaran.io import save_dataset
```

[7]:
```
dataset
```

[7]:
```
<xarray.Dataset>
Dimensions:   (time: 2100, spectral: 72)
Coordinates:
  * time      (time) float64 -1.0 -0.99 -0.98 -0.97 ... 19.96 19.97 19.98 19.99
  * spectral  (spectral) float64 600.0 601.4 602.8 604.2 ... 696.6 698.0 699.4
Data variables:
    data      (time, spectral) float64 -0.0175 -0.006334 0.0048 ... 1.7 1.532
Attributes:
    source_path:  dataset_1.nc
```

To get a feeling for our data, let's plot some traces.

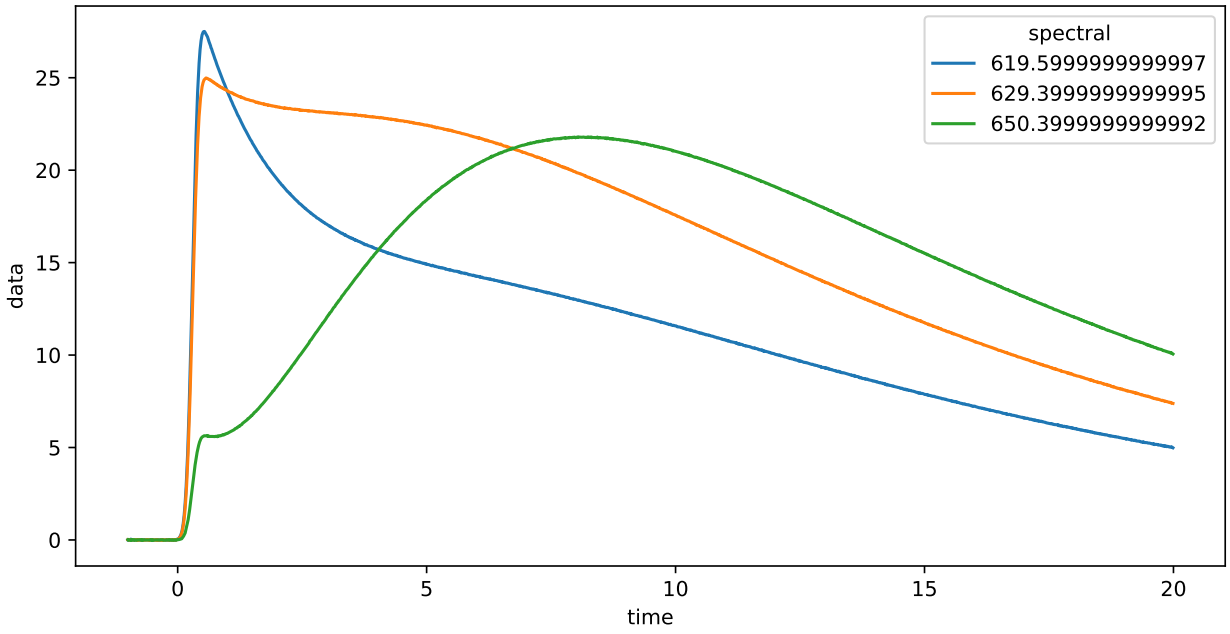
[8]:
```
plot_data = dataset.data.sel(spectral=[620, 630, 650], method="nearest")
plot_data.plot.line(x="time", aspect=2, size=5)
```

[8]:
```
[<matplotlib.lines.Line2D at 0x7fd67353fe20>,
 <matplotlib.lines.Line2D at 0x7fd67353fdc0>,
```

(continues on next page)

(continued from previous page)

```
<matplotlib.lines.Line2D at 0x7fd67353fee0>]
```



Since we want to see a difference of our saved and loaded data, we divide the amplitudes by 2 for no reason.

[9]:
```
dataset["data"] = dataset.data / 2
```

Now that we changed the data, let's write them to a file.

But in which order were the arguments again? And are there any additional option?

Good thing we documented our new plugin, so we can just lookup the help.

[10]:
```
from glotaran.io import show_data_io_method_help

show_data_io_method_help("json", "save_dataset")
```

```
Help on method save_dataset in module __main__:

save_dataset(dataset: 'xr.Dataset | xr.DataArray', file_name: 'str', *, my_extra_
↪option=None) method of __main__.JsonDataIo instance
    Write xarray.Dataset to a json file

    Parameters
    ----------
    dataset : xr.Dataset
        Dataset to be saved to file.
    file_name : str
        File to write the result data to.
    my_extra_option: str
        This argument is only for demonstration
```

Note that the **function** `save_dataset` has additional arguments:

- `format_name`: overwrites the inferred plugin selection
- `allow_overwrite`: Allows to overwrite existing files **(USE WITH CAUTION!!!)**

[11]:
```
help(save_dataset)
```

```
Help on function save_dataset in module glotaran.plugin_system.data_io_registration:

save_dataset(dataset: 'xr.Dataset | xr.DataArray', file_name: 'StrOrPath', format_name:
↪'str' = None, *, data_filters: 'list[str] | None' = None, allow_overwrite: 'bool' =␣
↪False, update_source_path: 'bool' = True, **kwargs: 'Any') -> 'None'
    Save data from :xarraydoc:`Dataset` or :xarraydoc:`DataArray` to a file.

    Parameters
    ----------
    dataset : xr.Dataset | xr.DataArray
        Data to be written to file.
    file_name : StrOrPath
        File to write the data to.
    format_name : str
        Format the file should be in, if not provided it will be inferred from the file␣
↪extension.
    data_filters : list[str] | None
        Optional list of items in the dataset to be saved.
    allow_overwrite : bool
        Whether or not to allow overwriting existing files, by default False
    update_source_path: bool
        Whether or not to update the ``source_path`` attribute to ``file_name`` when␣
↪saving.
        by default True
    **kwargs : Any
        Additional keyword arguments passes to the ``write_dataset`` implementation
        of the data io plugin. If you aren't sure about those use ``get_datawriter``
        to get the implementation with the proper help and autocomplete.
```

Since this is just an example and we don't overwrite important data we will use `allow_overwrite=True`. Also it makes writing this documentation easier, not having to manually delete the test file each time you run the cell.

[12]:
```
save_dataset(
    dataset, "half_intensity.json", allow_overwrite=True, my_extra_option="just as an␣
↪example"
)
```

```
Using my extra option for writing json: just as an example
```

Now let's test our data loading functionality.

[13]:
```
reloaded_data = load_dataset("half_intensity.json", my_extra_option="just as an example")
reloaded_data
```

```
Using my extra option loading json: just as an example
```

[13]:
```
<xarray.Dataset>
Dimensions:   (time: 2100, spectral: 72)
Coordinates:
```

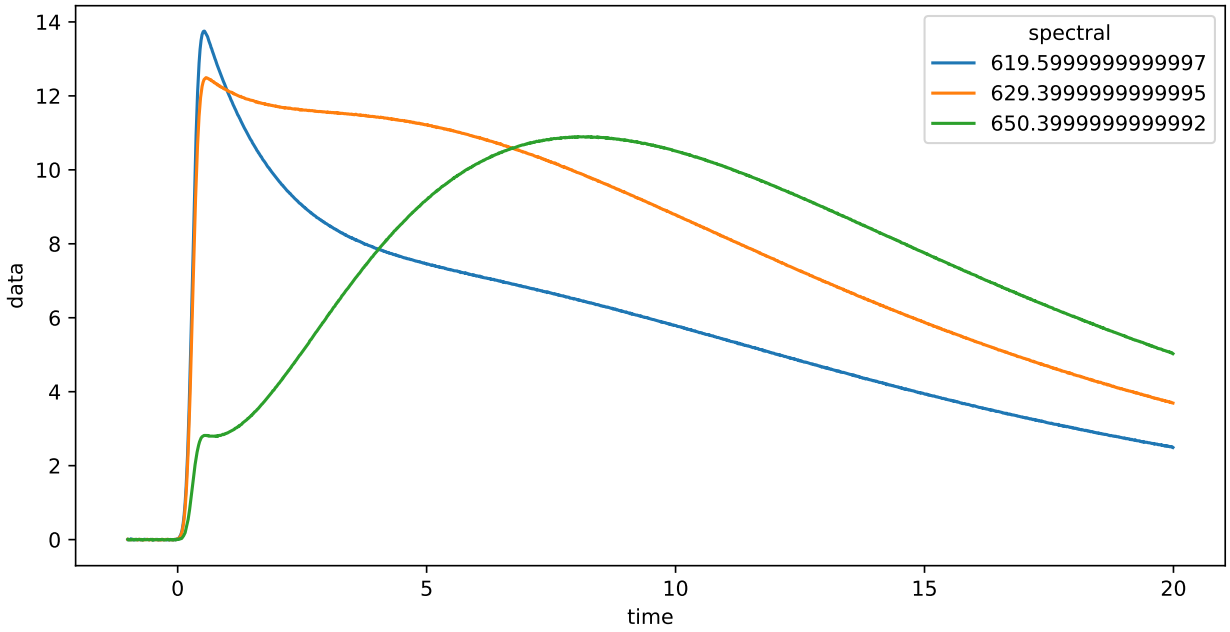
(continues on next page)

(continued from previous page)

```
  * time      (time) float64 -1.0 -0.99 -0.98 -0.97 ... 19.96 19.97 19.98 19.99
  * spectral  (spectral) float64 600.0 601.4 602.8 604.2 ... 696.6 698.0 699.4
Data variables:
    data      (time, spectral) float64 -0.008749 -0.003167 ... 0.8499 0.7658
Attributes:
    loader:       <function load_dataset at 0x7fd6879a4f70>
    source_path:  half_intensity.json
```

[14]:
```
reloaded_plot_data = reloaded_data.data.sel(spectral=[620, 630, 650], method="nearest")
reloaded_plot_data.plot.line(x="time", aspect=2, size=5)
```

[14]:
```
[<matplotlib.lines.Line2D at 0x7fd67c9600a0>,
 <matplotlib.lines.Line2D at 0x7fd672d73fa0>,
 <matplotlib.lines.Line2D at 0x7fd672d31130>]
```



Since this looks like the above plot, but with half the amplitudes, so writing and reading our data worked as we hoped it would.

Writing a `ProjectIo` plugin words analogous:

	`DataIo` plugin	`ProjectIo` plugin
Register function	`glotaran.plugin_system.data_io_registration.register_data_io`	`glotaran.plugin_system.project_io_registration.register_project_io`
Baseclass	`glotaran.io.interface.DataIoInterface`	`glotaran.io.interface.DataIoInterface`
Possible methods	`load_dataset`, `save_dataset`	`load_model`, `save_model`, `load_parameters`, `save_parameters`, `load_scheme`, `save_scheme`, `load_result`, `save_result`

Of course you don't have to implement all methods (sometimes that doesn't even make sense), but only the ones you need.

Last but not least:

Chances are that if you need a plugin someone else does too, so it would awesome if you would publish it open source, so the wheel isn't reinvented over and over again.

CHAPTER

SEVENTEEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[1] https://glotaran.github.io/legacy/file_formats

[1] https://glotaran.github.io/legacy/file_formats

PYTHON MODULE INDEX

g

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z