

---

# **pyglotaran Documentation**

***Release v0.5.1***

**Joern Weissenborn, Joris Snellenburg, Ivo van Stokkum**

**2021-12-31**



## CONTENTS:

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Installation</b>	<b>3</b>
<b>3</b>	<b>Quickstart/Cheat-Sheet</b>	<b>5</b>
<b>4</b>	<b>Changelog</b>	<b>17</b>
<b>5</b>	<b>Authors</b>	<b>25</b>
<b>6</b>	<b>Overview</b>	<b>27</b>
<b>7</b>	<b>Data IO</b>	<b>29</b>
<b>8</b>	<b>Plotting</b>	<b>31</b>
<b>9</b>	<b>Modelling</b>	<b>33</b>
<b>10</b>	<b>Parameter</b>	<b>35</b>
<b>11</b>	<b>Optimizing</b>	<b>37</b>
<b>12</b>	<b>Plugins</b>	<b>39</b>
<b>13</b>	<b>Command-line Interface</b>	<b>41</b>
<b>14</b>	<b>Contributing</b>	<b>45</b>
<b>15</b>	<b>API Documentation</b>	<b>53</b>
<b>16</b>	<b>Plugin development</b>	<b>365</b>
<b>17</b>	<b>Indices and tables</b>	<b>373</b>
	<b>Bibliography</b>	<b>375</b>
	<b>Python Module Index</b>	<b>377</b>
	<b>Index</b>	<b>379</b>



## 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 (\*.yaml) 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.



## 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 re-run the command above.

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



## 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 0.002764 -0.005696 ... 1.716 1.54
Attributes:
  source_path:  dataset_1.nc
```

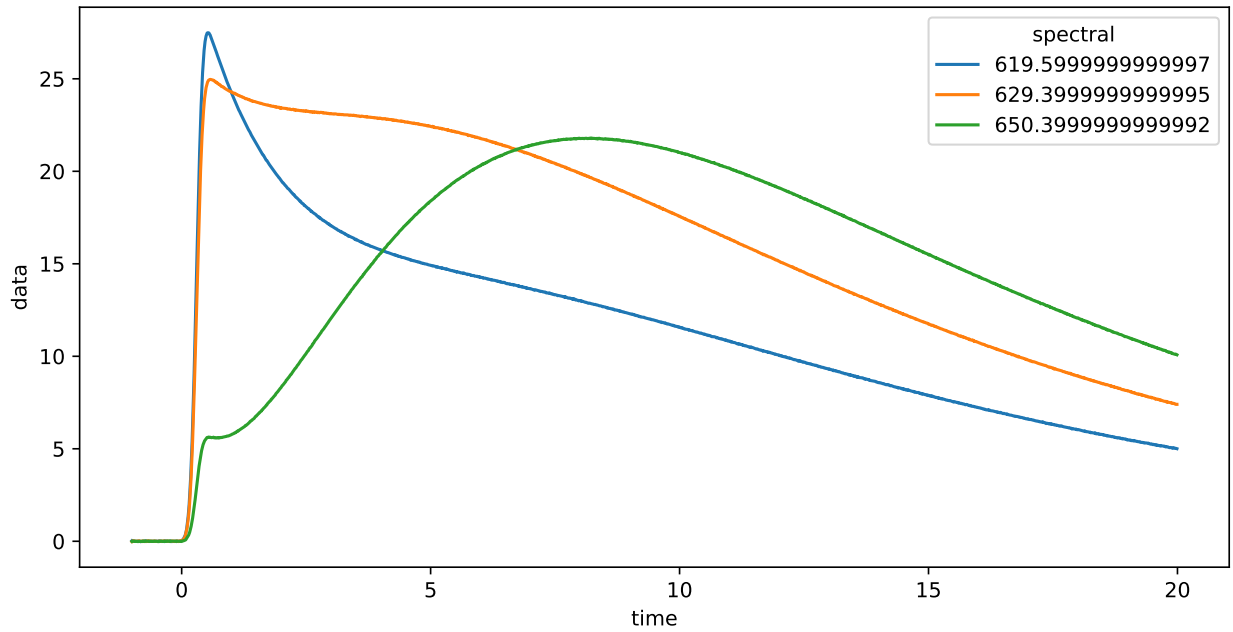
Like all data in pyglotaran, the dataset is a `xarray.Dataset`. You can find more information about the `xarray` library the [xarray homepage](#).

The loaded dataset is a simulated sequential model.

### 3.1 Plotting raw data

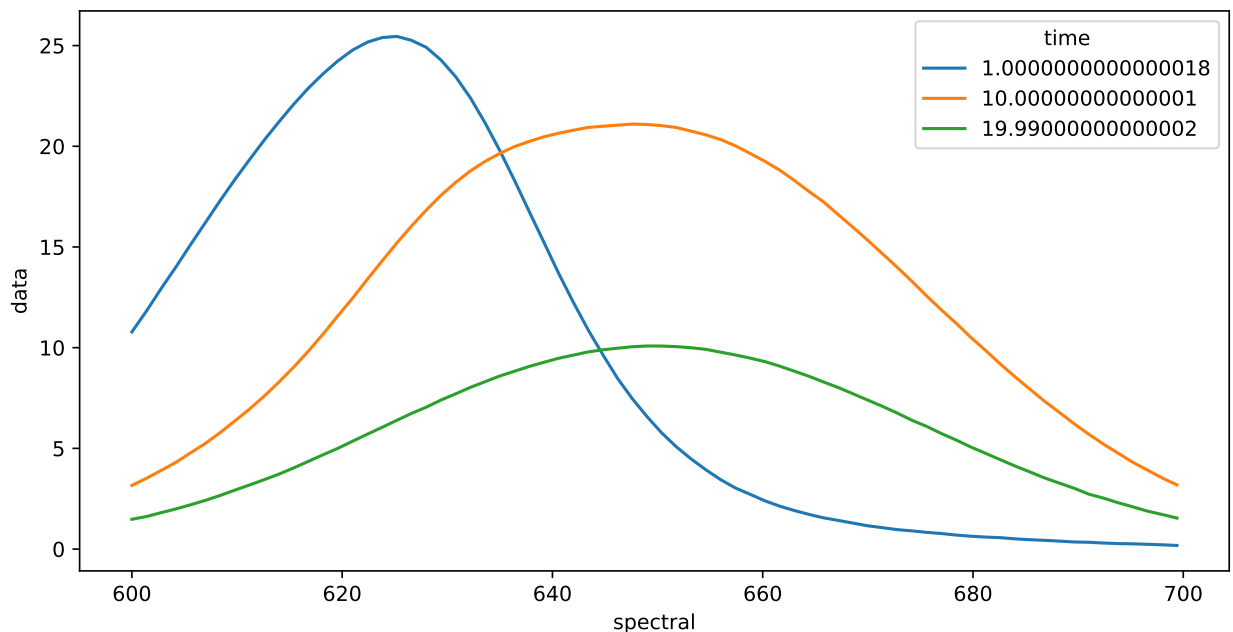
Now we let's plot some time traces.

```
[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

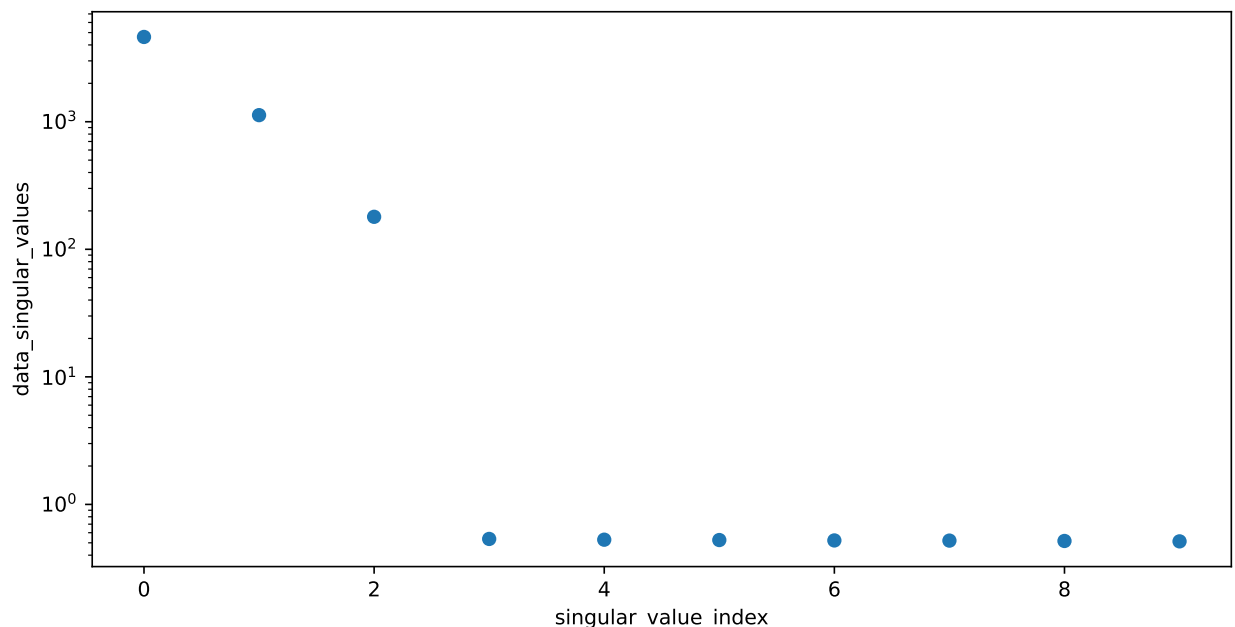
To get an idea about how to model your data, you should inspect the singular value decomposition. Pyglotaran has a function to calculate it (among other things).

```
[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 0.002764 ... 1.54
  data_left_singular_vectors (time, left_singular_value_index) float64 -1...
  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]: 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]: 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:**

<i>Label</i>	<i>Value</i>	<i>Standard Error</i>	<i>Mini- mum</i>	<i>Maxi- mum</i>	<i>Vary</i>	<i>Non- Negative</i>	<i>Expres- sion</i>
1	1.000e+00	nan	-inf	inf	False	False	None
0	0.000e+00	nan	-inf	inf	False	False	None

- **irf:**

<i>Label</i>	<i>Value</i>	<i>Standard Error</i>	<i>Mini- mum</i>	<i>Maxi- mum</i>	<i>Vary</i>	<i>Non- Negative</i>	<i>Expres- sion</i>
center	3.000e-01	nan	-inf	inf	True	False	None
width	1.000e-01	nan	-inf	inf	True	False	None

- **kinetic:**

<i>Label</i>	<i>Value</i>	<i>Standard Error</i>	<i>Mini- mum</i>	<i>Maxi- mum</i>	<i>Vary</i>	<i>Non- Negative</i>	<i>Expres- sion</i>
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	7.5359e+00			1.42e+02
1	2	7.5357e+00	2.20e-04	3.93e-05	5.38e-02
2	3	7.5357e+00	2.46e-11	4.38e-09	4.30e-06

Both `ftol` and `xtol` termination conditions are satisfied.

Function evaluations 3, initial cost 7.5359e+00, final cost 7.5357e+00, first-order ↵ optimality 4.30e-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	9.97e-05
Root Mean Square Error (RMSE)	9.98e-03
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±7.25e-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±4.77e-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±5.00e-06, initial: 3.00e-01)
  - *Width:* irf.width(1.00e-01±6.69e-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	5.003e-06	-inf	inf	True	False	None
width	1.000e-01	6.693e-06	-inf	inf	True	False	None

• kinetic:

Label	Value	Standard Error	Minimum	Maximum	Vary	Non-Negative	Expression
1	5.000e-01	7.248e-05	-inf	inf	True	False	None
2	3.000e-01	4.187e-05	-inf	inf	True	False	None
3	9.999e-02	4.772e-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]: &lt;xarray.Dataset&gt;

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...
  lifetime                 (component) float64 -2.0 ... -10.0
* 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 0.0027...
  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 6.093...
  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:   0.009983903887846732
  weighted_root_mean_square_error: 0.009983903887846732
  dataset_scale:            1

```

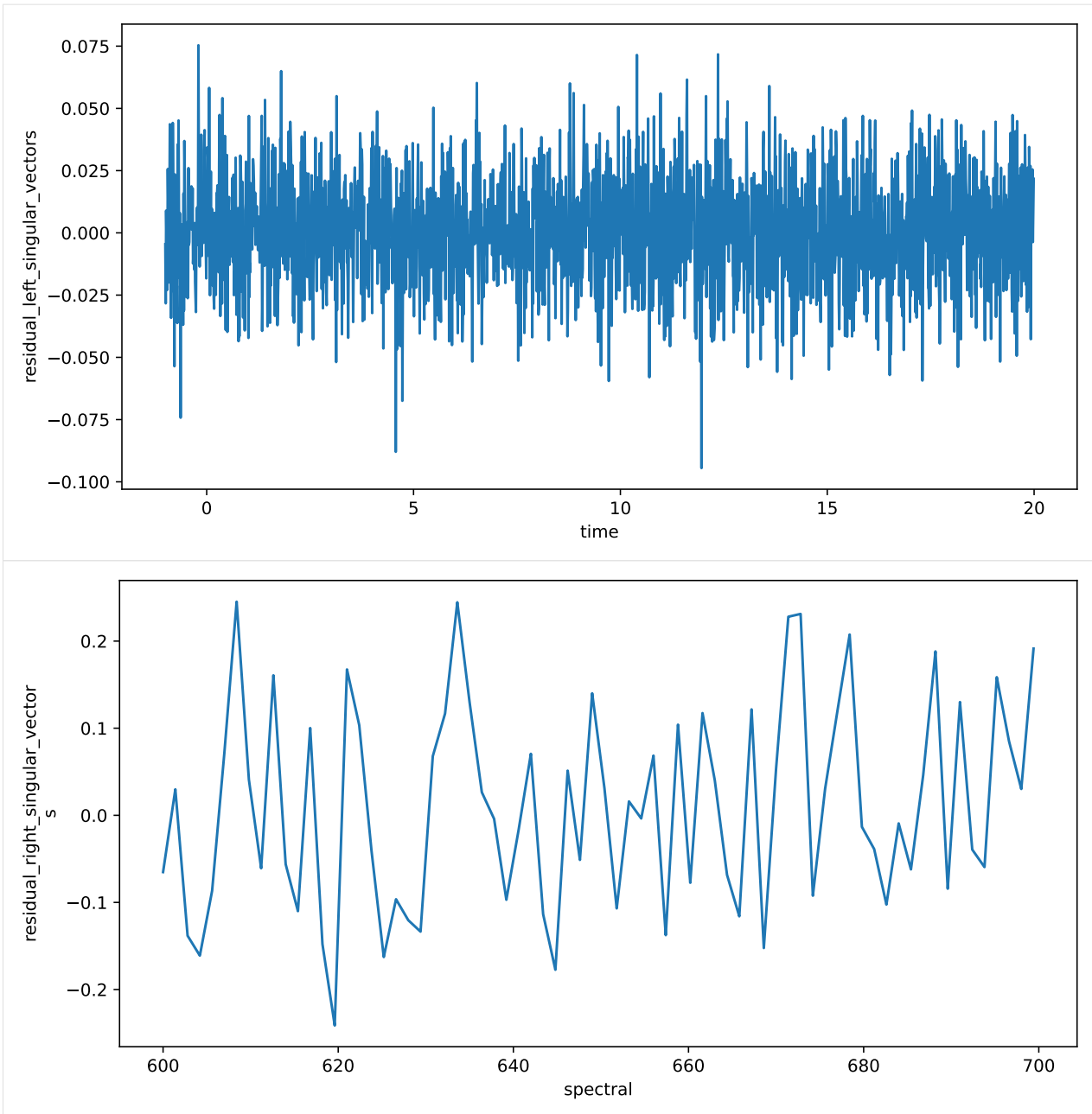
## 3.6 Visualize the Result

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")
```

## CHANGELOG

### 4.1 0.5.1 (2021-12-31)

#### 4.1.1 Bug fixes

- Bugfix Use normalized initial\_concentrations in result creation for decay megacomplex (#927)
- Fix save\_result crashes on Windows if input data are on a different drive than result (#931)

#### 4.1.2 Maintenance

- Forward port Improve result comparison workflow and v0.4 changelog (#938)
- Forward port of #936 test\_result\_consistency

### 4.2 0.5.0 (2021-12-01)

#### 4.2.1 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)

#### 4.2.2 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)

### 4.2.3 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)

### 4.2.4 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)

### 4.2.5 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')`

## 4.2.6 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)

## 4.3 0.4.2 (2021-12-31)

### 4.3.1 Bug fixes

- Backport of bugfix #927 discovered in PR #860 related to initial\_concentration normalization when saving results (#935).

### 4.3.2 Maintenance

- Updated ‘gold standard’ result comparison reference (old -> new)
- Refine test\_result\_consistency (#936).

## 4.4 0.4.1 (2021-09-07)

### 4.4.1 Features

- Integration test result validation (#760)

### 4.4.2 Bug fixes

- Fix unintended saving of sub-optimal parameters (0ece818, backport from #747)
- Improve ordering in k\_matrix involved\_compartments function (#791)

## 4.5 0.4.0 (2021-06-25)

### 4.5.1 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)

### 4.5.2 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)



### 4.5.3 Plugins

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

### 4.5.4 Documentation

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

### 4.5.5 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, ...)`

## 4.6 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)

## 4.7 0.3.2 (2021-02-28)

- Re-release of version 0.3.1 due to packaging issue

## 4.8 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)

## 4.9 0.3.0 (2021-02-11)

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

## 4.10 0.2.0 (2020-12-02)

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

## 4.11 0.1.0 (2020-07-14)

- Package was renamed to `pyglotaran` on PyPi

## 4.12 0.0.8 (2018-08-07)

- Changed `nan_policy` to `omit`

## 4.13 0.0.7 (2018-08-07)

- Added support for multiple shapes per compartment.

## 4.14 0.0.6 (2018-08-07)

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



## AUTHORS

### 5.1 Development Lead

- Joern Weissenborn <[joern.weissenborn@gmail.com](mailto:joern.weissenborn@gmail.com)>
- Joris Snellenburg <[j.snellenburg@gmail.com](mailto:j.snellenburg@gmail.com)>

### 5.2 Contributors

- Sebastian Weigand <[s.weigand.phy@gmail.com](mailto:s.weigand.phy@gmail.com)>

### 5.3 Special Thanks

- Stefan Schuetz
- Sergey P. Laptanok

### 5.4 Supervision

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

### 5.5 Original publications

1. Joris J. Snellenburg, Sergey Laptanok, 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.bbabi.2004.04.011>



**OVERVIEW**





---

CHAPTER  
**SEVEN**

---

**DATA IO**



**PLOTTING**



**MODELLING**



PARAMETER





**OPTIMIZING**



## 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.

## 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





## 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/glottaran/pyglottaran/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

pyglottaran could always use more documentation, whether as part of the official pyglottaran 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/glottaran/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](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 `GlottaranApiDeprecationWarning` 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 your home folder under `pyglotaran_examples_results`. Those results then 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 it might be included in a citation.

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





## API DOCUMENTATION

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

---

<i>glotaran</i>	Glotaran package <code>__init__.py</code>
-----------------	---

---

### 15.1 glotaran

Glotaran package `__init__.py`

#### Modules

---

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

---

### 15.1.1 analysis

This package contains functions for model simulation and fitting.

#### Modules

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

<i>residual_nnls</i>	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

<i>OptimizationGroup</i>	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

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

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

---

<i>OptimizationGroupCalculator</i>	A Problem class
------------------------------------	-----------------

---

### OptimizationGroupCalculator

**class** `glotaran.analysis.optimization_group_calculator.OptimizationGroupCalculator`(*group: Optimization-Group*)

Bases: `object`

A Problem class

#### Methods Summary

---

*calculate\_full\_penalty*

---

*calculate\_matrices*

---

*calculate\_residual*

---

continues on next page

Table 10 – continued from previous page

<code>create_index_dependent_result_dataset</code>	Creates a result datasets for index dependent matrices.
<code>create_index_independent_result_dataset</code>	Creates a result datasets for index independent matrices.
<code>prepare_result_creation</code>	

---

**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

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

### DatasetIndexModel

```
class glotaran.analysis.optimization_group_calculator_linked.DatasetIndexModel(label:  
str,  
in-  
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

<i>axis</i>	Alias for field number 2
<i>indices</i>	Alias for field number 1
<i>label</i>	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

<code>count</code>	Return number of occurrences of value.
<code>index</code>	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

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

<i>count</i>	Return number of occurrences of value.
<i>index</i>	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`(*group*)

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**

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

**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

---

<i>OptimizationGroupCalculatorUnlinked</i>	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

<code>simulate</code>	Simulates a model.
<code>simulate_clp</code>	
<code>simulate_global_model</code>	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

<code>apply_constraints</code>	
<code>apply_relations</code>	
<code>apply_weight</code>	
<code>calculate_clp_penalties</code>	
<code>calculate_matrix</code>	
<code>combine_matrix</code>	
<code>find_closest_index</code>	
<code>find_overlap</code>	
<code>get_idx_from_interval</code>	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
<code>get_min_max_from_interval</code>	
<code>reduce_matrix</code>	
<code>retrieve_clps</code>	

### **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: A tuple of floats with begin and end of the interval :param 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

<i>clp_labels</i>	Alias for field number 0
<i>matrix</i>	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

<i>count</i>	Return number of occurrences of value.
<i>index</i>	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

---

<i>residual_variable_projection</i>	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

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

<i>load_dataset</i>	Reads an ascii file in wavelength- or time-explicit format.
<i>save_dataset</i>	Save data from <code>xarray.Dataset</code> to a file ( <b>NOT IMPLEMENTED</b> ).

## 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:  
                                                                           Op-  
                                                                           tional[str]  
                                                                           =  
                                                                           None,  
                                                                           dataset:  
                                                                           Op-  
                                                                           tional[xarray.core.dataarray.DataArray]  
                                                                           =  
                                                                           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(axes)`

**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(axes)`

`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.DataArray] = 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

<i>CsvProjectIo</i>	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

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

continues on next page

Table 41 – continued from previous page

<code>save_result</code>	Save a Result instance to a spec file ( <b>NOT IMPLEMENTED</b> ).
<code>save_scheme</code>	Save a Scheme instance to a spec file ( <b>NOT IMPLEMENTED</b> ).

### `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

---

<code>glotaran.builtin.io.folder.folder_plugin</code>	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

<i>FolderProjectIo</i>	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

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

### 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

<i>NetCDFDataIo</i>	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

<i>load_dataset</i>	Read data from a file to <i>xarray.Dataset</i> or <i>xarray.DataArray</i> ( <b>NOT IMPLEMENTED</b> ).
<i>save_dataset</i>	Save data from <i>xarray.Dataset</i> to a file ( <b>NOT IMPLEMENTED</b> ).

#### 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

---

<code>glotaran.builtin.io.sdt.sdt_file_reader</code>	Glotarans module to read files
--	--------------------------------

---

## sdt\_file\_reader

Glotarans module to read files

## Classes

### Summary

---

<code>SdtDataIo</code>	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

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

## 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

---

<i>YmlProjectIo</i>	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

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

### load\_model

`YmlProjectIo.load_model(file_name: str) → glotaran.model.model.Model`

`parse_yaml_file` reads the given file and parses its content as YAML.

**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 YAML.

**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

<i>dimension</i>	ModelProperty is an extension of the property decorator.
<i>label</i>	ModelProperty is an extension of the property decorator.
<i>name</i>	
<i>type</i>	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

<i>as_dict</i>
<i>calculate_matrix</i>
<i>fill</i>
<i>finalize_data</i>
<i>from_dict</i>

continues on next page

Table 58 – continued from previous page

<i>glotaran_dataset_model_items</i>
<i>glotaran_dataset_properties</i>
<i>glotaran_model_items</i>
<i>glotaran_unique</i>
<i>index_dependent</i>
<i>markdown</i>
<i>validate</i>

**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

<i>glotaran.builtin.megacomplexes.coherent_artifact.coherent_artifact_megacomplex</i>	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

<i>dimension</i>	ModelProperty is an extension of the property decorator.
<i>label</i>	ModelProperty is an extension of the property decorator.
<i>name</i>	
<i>order</i>	ModelProperty is an extension of the property decorator.
<i>type</i>	ModelProperty is an extension of the property decorator.
<i>width</i>	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

<i>finalize_data</i>
<i>from_dict</i>
<i>glotaran_dataset_model_items</i>
<i>glotaran_dataset_properties</i>
<i>glotaran_model_items</i>
<i>glotaran_unique</i>
<i>index_dependent</i>
<i>markdown</i>
<i>validate</i>

**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` 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

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

<i>glotaran_dataset_properties</i>
<i>glotaran_model_items</i>
<i>glotaran_unique</i>
<i>index_dependent</i>
<i>markdown</i>
<i>validate</i>

**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

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

decay\_megacomplex

This package contains the decay megacomplex item.

Classes

Summary

<code>DecayMegacomplex</code>	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

<i>dimension</i>	ModelProperty is an extension of the property decorator.
<i>involved_compartments</i>	
<i>k_matrix</i>	ModelProperty is an extension of the property decorator.
<i>label</i>	ModelProperty is an extension of the property decorator.
<i>name</i>	
<i>type</i>	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

<i>index_dependent</i>
<i>markdown</i>
<i>validate</i>
<b>as_dict</b>
DecayMegacomplex. <b>as_dict</b> () → dict
<b>calculate_matrix</b>
DecayMegacomplex. <b>calculate_matrix</b> (dataset_model: DatasetModel, indices: dict[str, int], **kwargs)
<b>fill</b>
DecayMegacomplex. <b>fill</b> (model: Model, parameters: ParameterGroup) → cls
<b>finalize_data</b>
DecayMegacomplex. <b>finalize_data</b> (dataset_model: glotaran.model.dataset_model.DatasetModel, dataset: xarray.core.dataset.Dataset, is_full_model: bool = False, as_global: bool = False)
<b>from_dict</b>
<b>classmethod</b> DecayMegacomplex. <b>from_dict</b> (values: dict) → cls
<b>full_k_matrix</b>
DecayMegacomplex. <b>full_k_matrix</b> (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

<i>InitialConcentration</i>	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

<i>compartments</i>	ModelProperty is an extension of the property decorator.
<i>exclude_from_normalize</i>	ModelProperty is an extension of the property decorator.
<i>label</i>	ModelProperty is an extension of the property decorator.
<i>parameters</i>	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

<i>Irf</i>	Represents an IRF.
<i>IrfGaussian</i>	
<i>IrfMeasured</i>	A measured IRF.
<i>IrfMultiGaussian</i>	Represents a gaussian IRF.
<i>IrfSpectralGaussian</i>	
<i>IrfSpectralMultiGaussian</i>	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

<i>backswEEP</i>	ModelProperty is an extension of the property decorator.
<i>backswEEP_period</i>	ModelProperty is an extension of the property decorator.
<i>center</i>	ModelProperty is an extension of the property decorator.
<i>label</i>	ModelProperty is an extension of the property decorator.
<i>normalize</i>	ModelProperty is an extension of the property decorator.
<i>scale</i>	ModelProperty is an extension of the property decorator.
<i>shift</i>	ModelProperty is an extension of the property decorator.

continues on next page

Table 77 – continued from previous page

<i>type</i>	ModelProperty is an extension of the property decorator.
<i>width</i>	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

---

<i>as_dict</i>	
<hr/>	
<i>calculate</i>	
<hr/>	
<i>fill</i>	
<hr/>	
<i>from_dict</i>	
<hr/>	
<i>is_index_dependent</i>	
<hr/>	
<i>markdown</i>	
<hr/>	
<i>parameter</i>	Returns the properties of the irf with shift applied.
<hr/>	
<i>validate</i>	

---

**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

<i>label</i>	ModelProperty is an extension of the property decorator.
<i>type</i>	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

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

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

### **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

<i>backsweep</i>	ModelProperty is an extension of the property decorator.
<i>backsweep_period</i>	ModelProperty is an extension of the property decorator.
<i>center</i>	ModelProperty is an extension of the property decorator.

continues on next page

Table 83 – continued from previous page

<i>center_dispersion_coefficients</i>	ModelProperty is an extension of the property decorator.
<i>dispersion_center</i>	ModelProperty is an extension of the property decorator.
<i>label</i>	ModelProperty is an extension of the property decorator.
<i>model_dispersion_with_wavenumber</i>	ModelProperty is an extension of the property decorator.
<i>normalize</i>	ModelProperty is an extension of the property decorator.
<i>scale</i>	ModelProperty is an extension of the property decorator.
<i>shift</i>	ModelProperty is an extension of the property decorator.
<i>type</i>	ModelProperty is an extension of the property decorator.
<i>width</i>	ModelProperty is an extension of the property decorator.
<i>width_dispersion_coefficients</i>	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**

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



**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

<i>backsweep</i>	ModelProperty is an extension of the property decorator.
<i>backsweep_period</i>	ModelProperty is an extension of the property decorator.
<i>center</i>	ModelProperty is an extension of the property decorator.
<i>center_dispersion_coefficients</i>	ModelProperty is an extension of the property decorator.
<i>dispersion_center</i>	ModelProperty is an extension of the property decorator.
<i>label</i>	ModelProperty is an extension of the property decorator.
<i>model_dispersion_with_wavenumber</i>	ModelProperty is an extension of the property decorator.
<i>normalize</i>	ModelProperty is an extension of the property decorator.
<i>scale</i>	ModelProperty is an extension of the property decorator.
<i>shift</i>	ModelProperty is an extension of the property decorator.
<i>type</i>	ModelProperty is an extension of the property decorator.
<i>width</i>	ModelProperty is an extension of the property decorator.
<i>width_dispersion_coefficients</i>	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**

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

**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

---

<i>KMatrix</i>	A K-Matrix represents a first order differential system.
----------------	--

---

### KMatrix

**class** glotaran.builtin.megacomplexes.decay.k\_matrix.**KMatrix**

Bases: *object*

A K-Matrix represents a first order differential system.

### Attributes Summary

---

<i>label</i>	ModelProperty is an extension of the property decorator.
<i>matrix</i>	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**

<i>a_matrix</i>	The resulting A matrix of the KMatrix.
<i>a_matrix_as_markdown</i>	Returns the A Matrix as markdown formatted table.
<i>a_matrix_non_unibranh</i>	The resulting A matrix of the KMatrix for a non-unibranched model.
<i>a_matrix_unibranh</i>	The resulting A matrix of the KMatrix for an unibranched model.
<i>as_dict</i>	
<i>combine</i>	Creates a combined matrix.
<i>eigen</i>	Returns the eigenvalues and eigenvectors of the k matrix.
<i>empty</i>	Creates an empty K-Matrix.
<i>fill</i>	
<i>from_dict</i>	
<i>full</i>	The full representation of the KMatrix as numpy array.
<i>involved_compartments</i>	A list of all compartments in the Matrix.
<i>is_unibranched</i>	Returns true in the KMatrix represents an unibranched model.
<i>markdown</i>	
<i>matrix_as_markdown</i>	Returns the KMatrix as markdown formatted table.
<i>rates</i>	The resulting rates of the matrix.
<i>reduced</i>	The reduced representation of the KMatrix as numpy array.
<i>validate</i>	

### `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_unibranh`

`KMatrix.a_matrix_non_unibranh(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_unibranh`

`KMatrix.a_matrix_unibranh(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\_unbranched

**KMatrix.is\_unbranched**(*initial\_concentration:*

*glotaran.builtin.megacomplexes.decay.initial\_concentration.InitialConcentration*)

→ bool

Returns true in the KMatrix represents an unbranched 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\_unibranched**(*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\_unibranched**(*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 over-written 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

<code>calculate_decay_matrix_gaussian_irf</code>	Calculates a decay matrix with a gaussian irf.
<code>calculate_decay_matrix_no_irf</code>	
<code>decay_matrix_implementation</code>	
<code>retrieve_decay_associated_data</code>	
<code>retrieve_irf</code>	
<code>retrieve_species_associated_data</code>	



### **calculate\_decay\_matrix\_gaussian\_irf**

```
glotaran.builtin.megacomplexes.decay.util.calculate_decay_matrix_gaussian_irf(matrix,  
                                                                              rates,  
                                                                              times,  
                                                                              center,  
                                                                              width,  
                                                                              scale,  
                                                                              back-sweep,  
                                                                              back-sweep_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.DatasetModel)
```

### retrieve\_decay\_associated\_data

```
glotaran.builtin.megacomplexes.decay.util.retrieve_decay_associated_data(megacomplex:  
                                                                           De-  
                                                                           cayMega-  
                                                                           complex,  
                                                                           dataset_model:  
                                                                           Dataset-  
                                                                           Model,  
                                                                           dataset:  
                                                                           xr.Dataset,  
                                                                           global_dimension:  
                                                                           str,  
                                                                           name:  
                                                                           str,  
                                                                           multi-  
                                                                           ple_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:  
                                                                              xar-  
                                                                              ray.core.dataset.Dataset,  
                                                                              species_dimension:  
                                                                              str,  
                                                                              global_dimension:  
                                                                              str,  
                                                                              name:  
                                                                              str,  
                                                                              is_full_model:  
                                                                              bool,  
                                                                              as_global:  
                                                                              bool)
```

spectral

Modules

<i>glotaran.builtin.megacomplexes.spectral.shape</i>	This package contains the spectral shape item.
<i>glotaran.builtin.megacomplexes.spectral.spectral_megacomplex</i>	

shape

This package contains the spectral shape item.

Classes

Summary

<i>SpectralShape</i>	Base class for spectral shapes
<i>SpectralShapeGaussian</i>	A Gaussian spectral shape
<i>SpectralShapeOne</i>	A constant spectral shape with value 1
<i>SpectralShapeSkewedGaussian</i>	A skewed Gaussian spectral shape
<i>SpectralShapeZero</i>	A constant spectral shape with value 0

SpectralShape

**class** `glotaran.builtin.megacomplexes.spectral.shape.SpectralShape`

Bases: `object`

Base class for spectral shapes

Methods Summary

<i>add_type</i>
<i>get_default_type</i>

**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**

<i>amplitude</i>	ModelProperty is an extension of the property decorator.
<i>label</i>	ModelProperty is an extension of the property decorator.
<i>location</i>	ModelProperty is an extension of the property decorator.
<i>type</i>	ModelProperty is an extension of the property decorator.
<i>width</i>	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*<sub>0</sub> : **location**
- *Δ* : **width**

In this formalism, *Δ* represents the full width at half maximum (FWHM). Compared to the more common definition  $\exp(-(x - \mu)^2 / (2\sigma^2))$  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*<sub>0</sub> : location
- *Δ* : width

In this formalism, Δ represents the full width at half maximum (FWHM). Compared to the more common definition  $\exp(-(x - \mu)^2 / (2\sigma^2))$  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

<i>label</i>	ModelProperty is an extension of the property decorator.
<i>type</i>	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

<i>as_dict</i>	
<i>calculate</i>	calculate calculates the shape.
<i>fill</i>	
<i>from_dict</i>	
<i>markdown</i>	
<i>validate</i>	



### 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

<code>amplitude</code>	ModelProperty is an extension of the property decorator.
<code>label</code>	ModelProperty is an extension of the property decorator.
<code>location</code>	ModelProperty is an extension of the property decorator.
<code>skewness</code>	ModelProperty is an extension of the property decorator.
<code>type</code>	ModelProperty is an extension of the property decorator.
<code>width</code>	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:

$$f(x, x_0, A, \Delta, b) = \begin{cases} 0 & \text{if } \theta \leq 0 \\ A \exp\left(-\frac{\log(2) \log(\theta(x, x_0, \Delta, b))^2}{b^2}\right) & \text{if } \theta > 0 \end{cases}$$

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
- $\Delta$ : width
- $b$ : skewness

Where  $\Delta$  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  $\lim_{b \rightarrow 0} \frac{\ln(1+bx)}{b} = x$ ), 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:

$$f(x, x_0, A, \Delta, b) = \begin{cases} 0 & \text{if } \theta \leq 0 \\ A \exp\left(-\frac{\log(2) \log(\theta(x, x_0, \Delta, b))^2}{b^2}\right) & \text{if } \theta > 0 \end{cases}$$

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`
- `x0`: `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, x0, A, Δ, b)` simplifies to a normal gaussian (since  $\lim_{b \rightarrow 0} \frac{\ln(1+bx)}{b} = x$ ), 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*)  $\rightarrow$  `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*)  $\rightarrow$  `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

<code>label</code>	ModelProperty is an extension of the property decorator.
<code>type</code>	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

<i>dimension</i>	ModelProperty is an extension of the property decorator.
<i>label</i>	ModelProperty is an extension of the property decorator.
<i>name</i>	
<i>shape</i>	ModelProperty is an extension of the property decorator.
<i>type</i>	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

---

<i>export</i>	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

---

<i>optimize_cmd</i>	Optimizes a model.
---------------------	--------------------

---

## optimize\_cmd

```
glotaran.cli.commands.optimize.optimize_cmd(dataformat: str, data: List[str], out: str,
                                             outformat: str, nfev: int, nmls: bool, yes: bool,
                                             parameters_file: str, model_file: str,
                                             scheme_file: str)
```

Optimizes a model. e.g.: `glotaran optimize -`

## pluginlist

### Functions

#### Summary

<code>plugin_list_cmd</code>	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

<code>print_cmd</code>	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

<code>load_dataset_file</code>	
--------------------------------	--

continues on next page

Table 111 – continued from previous page

<code>load_model_file</code>	
<code>load_parameter_file</code>	
<code>load_scheme_file</code>	
<code>project_io_list_supporting_plugins</code>	List all project-io plugin that implement <code>method_name</code> .
<code>select_data</code>	
<code>select_name</code>	
<code>signature_analysis</code>	
<code>write_data</code>	

### `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

<i>name</i>	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**

<i>convert</i>	Convert the value to the correct type.
<i>fail</i>	Helper method to fail with an invalid value message.
<i>get_metavar</i>	Returns the metavar default for this param if it provides one.
<i>get_missing_message</i>	Optionally might return extra information about a missing parameter.
<i>shell_complete</i>	Return a list of <code>CompletionItem</code> objects for the incomplete value.
<i>split_envvar_value</i>	Given a value from an environment variable this splits it up into small chunks depending on the defined envvar list splitter.
<i>to_info_dict</i>	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

<code>validate_cmd</code>	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

---

<code>glotaran.deprecation.deprecation_utils</code>	Helper functions to give deprecation warnings.
<code>glotaran.deprecation.modules</code>	Package containing deprecated implementations which were removed.

---

### deprecation\_utils

Helper functions to give deprecation warnings.

### Functions

#### Summary

---

<code>check_overdue</code>	Check if a deprecation is overdue for removal.
<code>check_qualnames_in_tests</code>	Test that qualnames import path exists when running tests.
<code>deprecate</code>	Decorate a function, method or class to deprecate it.
<code>deprecate_dict_entry</code>	Replace dict entry inplace and warn about usage change, if present in the dict.
<code>deprecate_module_attribute</code>	Import and return an attribute from the new location.
<code>deprecate_submodule</code>	Create a module at runtime which retrieves attributes from new module.
<code>glotaran_version</code>	Version of the distribution.
<code>module_attribute</code>	Import and return the attribute (e.g.
<code>parse_version</code>	Parse version string to tuple of three ints for comparison.
<code>raise_deprecation_error</code>	Raise <code>GlotaranDeprectedApiError</code> error, with formatted message.
<code>warn_deprecated</code>	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\_yaml\_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_yaml_str)'`
- **new\_qual\_name\_usage** (*str*) – New usage as fully qualified name e.g.: `'glotaran.io.load_model(model_yaml_str, format_name="yaml_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**(\*, *depre-*  
*cated\_qual\_name:*  
*str,*  
*new\_qual\_name:*  
*str,*  
*to\_be\_removed\_in\_version:*  
*str*) → Any

Import and return and attribute 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 causing 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 than 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\_yaml\_str)'
- **new\_qual\_name\_usage** (*str*) – New usage as fully qualified name e.g.: 'glotaran.io.load\_model(model\_yaml\_str, format\_name="yaml\_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\_yaml\_str)'
- **new\_qual\_name\_usage** (*str*) – New usage as fully qualified name e.g.: 'glotaran.io.load\_model(model\_yaml\_str, format\_name="yaml\_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 is 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 be 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

<code>GlottaranApiDeprecationWarning</code>	Warning to give users about API changes.
<code>GlottaranDeprectedApiError</code>	Exception raised when a deprecation has no replacement.
<code>OverDueDeprecation</code>	Error thrown when a deprecation should have been removed.

### GlotalanApiDeprecationWarning

**exception** `glotaran.deprecation.deprecation_utils.GlotalanApiDeprecationWarning`  
Warning to give users about API changes.

**See also:**

`deprecate`, `warn_deprecated`, `deprecate_module_attribute`, `deprecate_submodule`,  
`deprecate_dict_entry`

### GlotalanDeprectedApiError

**exception** `glotaran.deprecation.deprecation_utils.GlotalanDeprectedApiError`  
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

---

<code>glotaran.deprecation.modules.builtin_io_yaml</code>	Deprecation functions for the yaml parser.
<code>glotaran.deprecation.modules.glotaran_root</code>	Deprecated attributes from <code>glotaran.__init__</code> which are removed.

---



## builtin\_io\_yaml

Deprecation functions for the yaml parser.

### Functions

#### Summary

<code>model_spec_deprecations</code>	Check deprecations in the model specification spec dict.
<code>scheme_spec_deprecations</code>	Check deprecations in the scheme specification spec dict.

#### model\_spec\_deprecations

`glotaran.deprecation.modules.builtin_io_yaml.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_yaml.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

<code>read_model_from_yaml</code>	Parse yaml string to Model.
<code>read_model_from_yaml_file</code>	Parse model.yaml file to Model.
<code>read_parameters_from_csv_file</code>	Parse parameters_file to ParameterGroup.
<code>read_parameters_from_yaml</code>	Parse yaml string to ParameterGroup.
<code>read_parameters_from_yaml_file</code>	Parse parameters_file to ParameterGroup.

### read\_model\_from\_yaml

glotaran.deprecation.modules.glotaran\_root.read\_model\_from\_yaml(model\_yaml\_str: *str*)  
→ Model

Parse yaml string to Model.

**Warning:** Deprecated use `glotaran.io.load_model(model_yaml_str, format_name="yaml_str")` instead.

**Parameters** `model_yaml_str` (*str*) – Model spec description in yaml.

**Returns** Model described in `model_yaml_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\_yaml\_str: str*) → ParameterGroup

Parse yaml string to ParameterGroup.

**Warning:** Deprecated use `glotaran.io.load_parameters(parameters_yaml_str, format_name="yaml_str")` instead.

**Parameters** `parameters_yaml_str` (*str*) – PArameter spec description in yaml.

**Returns** ParameterGroup described in `parameters_yaml_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

---

<code>glotaran.io.interface</code>	Baseclasses to create Data/Project IO plugins from.
<code>glotaran.io.prepare_dataset</code>	

---

#### 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 demonstrate 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

---

<code>DataIoInterface</code>	Baseclass for Data IO plugins.
<code>ProjectIoInterface</code>	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

<code>load_dataset</code>	Read data from a file to <code>xarray.Dataset</code> or <code>xarray.DataArray</code> ( <b>NOT IMPLEMENTED</b> ).
<code>save_dataset</code>	Save data from <code>xarray.Dataset</code> to a file ( <b>NOT IMPLEMENTED</b> ).

### 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

<code>load_model</code>	Create a Model instance from the specs defined in a file ( <b>NOT IMPLEMENTED</b> ).
<code>load_parameters</code>	Create a ParameterGroup instance from the specs defined in a file ( <b>NOT IMPLEMENTED</b> ).
<code>load_result</code>	Create a Result instance from the specs defined in a file ( <b>NOT IMPLEMENTED</b> ).
<code>load_scheme</code>	Create a Scheme instance from the specs defined in a file ( <b>NOT IMPLEMENTED</b> ).
<code>save_model</code>	Save a Model instance to a spec file ( <b>NOT IMPLEMENTED</b> ).
<code>save_parameters</code>	Save a ParameterGroup instance to a spec file ( <b>NOT IMPLEMENTED</b> ).
<code>save_result</code>	Save a Result instance to a spec file ( <b>NOT IMPLEMENTED</b> ).
<code>save_scheme</code>	Save a Scheme instance to a spec file ( <b>NOT IMPLEMENTED</b> ).

### 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

<code>add_svd_to_dataset</code>	Add the SVD of a dataset inplace as Data variables to the dataset.
<code>prepare_time_trace_dataset</code>	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

Glottaran Model Package

This package contains the Glottaran's base model object, the model decorators and common model items.

### Modules

<code>glottaran.model.clp_penalties</code>	This package contains compartment constraint items.
<code>glottaran.model.constraint</code>	This package contains compartment constraint items.
<code>glottaran.model.dataset_group</code>	
<code>glottaran.model.dataset_model</code>	The DatasetModel class.
<code>glottaran.model.interval_property</code>	Helper functions.
<code>glottaran.model.item</code>	The model item decorator.
<code>glottaran.model.megacomplex(*[, dimension, ...])</code>	The <code>@megacomplex</code> decorator is intended to be used on subclasses of <code>glottaran.model.Megacomplex</code> .
<code>glottaran.model.model</code>	A base class for global analysis models.
<code>glottaran.model.property</code>	This module holds the model property class.
<code>glottaran.model.relation</code>	Glottaran Relation
<code>glottaran.model.util</code>	Helper functions.
<code>glottaran.model.weight</code>	The Weight property class.

### clp\_penalties

This package contains compartment constraint items.

### Functions

#### Summary

---

`apply_spectral_penalties`

---

---

`has_spectral_penalties`

---

#### apply\_spectral\_penalties

`glottaran.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**

<i>EqualAreaPenalty</i>	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**

<i>parameter</i>	ModelProperty is an extension of the property decorator.
<i>source</i>	ModelProperty is an extension of the property decorator.
<i>source_intervals</i>	ModelProperty is an extension of the property decorator.
<i>target</i>	ModelProperty is an extension of the property decorator.
<i>target_intervals</i>	ModelProperty is an extension of the property decorator.
<i>weight</i>	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

<i>applies</i>	Returns true if the index is in one of the intervals.
<i>as_dict</i>	
<i>fill</i>	
<i>from_dict</i>	
<i>markdown</i>	
<i>validate</i>	

### 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

<i>Constraint</i>	A constraint is applied on one clp on one or many intervals on the estimated axis type.
<i>OnlyConstraint</i>	A only constraint sets the calculated matrix row of a compartment to 0 outside the given intervals.
<i>ZeroConstraint</i>	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**

<i>interval</i>	ModelProperty is an extension of the property decorator.
<i>target</i>	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

<i>applies</i>	Returns true if <i>value</i> is in one of the intervals.
<i>as_dict</i>	
<i>fill</i>	
<i>from_dict</i>	
<i>markdown</i>	
<i>validate</i>	

### 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

<i>interval</i>	ModelProperty is an extension of the property decorator.
<i>target</i>	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

<i>applies</i>	Returns true if value is in one of the intervals.
<i>as_dict</i>	
<i>fill</i>	
<i>from_dict</i>	
<i>markdown</i>	
<i>validate</i>	

### 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

<i>link_clp</i>	Whether to link the clp parameter.
<i>residual_function</i>	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

<code>ensure_unique_megacomplexes</code>	Ensure that unique megacomplexes Are only used once per dataset.
<code>finalize_data</code>	
<code>get_coordinates</code>	Gets the dataset model's coordinates.
<code>get_data</code>	Gets the dataset model's data.
<code>get_global_axis</code>	Gets the dataset model's global axis.
<code>get_global_dimension</code>	Returns the dataset model's global dimension.
<code>get_model_axis</code>	Gets the dataset model's model axis.
<code>get_model_dimension</code>	Returns the dataset model's model dimension.
<code>get_weight</code>	Gets the dataset model's weight.
<code>has_global_model</code>	Indicates if the dataset model can model the global dimension.
<code>is_index_dependent</code>	Indicates if the dataset model is index dependent.
<code>iterate_global_megacomplexes</code>	Iterates of der dataset model's global megacomplexes.
<code>iterate_megacomplexes</code>	Iterates of der dataset model's megacomplexes.
<code>overwrite_global_dimension</code>	Overwrites the dataset model's global dimension.
<code>overwrite_index_dependent</code>	Overrides the index dependency of the dataset
<code>overwrite_model_dimension</code>	Overwrites the dataset model's model dimension.
<code>set_coordinates</code>	Sets the dataset model's coordinates.
<code>set_data</code>	Sets the dataset model's data.
<code>swap_dimensions</code>	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

---

<i>IntervalProperty</i>	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

---

<i>interval</i>	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

<i>applies</i>	Returns true if <i>value</i> is in one of the intervals.
<i>as_dict</i>	
<i>fill</i>	
<i>from_dict</i>	
<i>markdown</i>	
<i>validate</i>	

## 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

<code>model_item</code>	The <code>@model_item</code> decorator adds the given properties to the class.
<code>model_item_typed</code>	The <code>model_item_typed</code> 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.
<code>model_item_validator</code>	The <code>model_item_validator</code> 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

---

<i>Model</i>	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

<i>dataset_group_models</i>	
<i>default_megacomplex</i>	The default megacomplex used by this model.
<i>global_dimension</i>	Deprecated use <code>Scheme.global_dimensions['&lt;dataset_name&gt;']</code> instead
<i>global_megacomplex</i>	Alias for <code>glotaran.model.megacomplex</code> .
<i>megacomplex_types</i>	The megacomplex types used by this model.
<i>model_dimension</i>	Deprecated use <code>Scheme.model_dimensions['&lt;dataset_name&gt;']</code> instead
<i>model_items</i>	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

<code>as_dict</code>	
<code>from_dict</code>	Creates a model from a dictionary.
<code>get_dataset_groups</code>	
<code>get_parameters</code>	
<code>is_groupable</code>	
<code>loader</code>	Create a Model instance from the specs defined in a file.
<code>markdown</code>	Formats the model as Markdown string.
<code>need_index_dependent</code>	Returns true if e.g.
<code>problem_list</code>	Returns a list with all problems in the model and missing parameters if specified.
<code>valid</code>	Returns <i>True</i> if the number problems in the model is 0, else <i>False</i>
<code>validate</code>	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

---

<i>ModelProperty</i>	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

<i><code>fdel</code></i>	
<i><code>fget</code></i>	
<i><code>fset</code></i>	
<i><code>glotaran_allow_none</code></i>	Check if the property is allowed to be None.
<i><code>glotaran_is_mapping_property</code></i>	Check if the type is mapping.
<i><code>glotaran_is_parameter_property</code></i>	Check if the subtype is parameter.
<i><code>glotaran_is_scalar_property</code></i>	Check if the type is scalar.
<i><code>glotaran_is_sequence_property</code></i>	Check if the type is a sequence.
<i><code>glotaran_property_subtype</code></i>	Get the subscribed type.
<i><code>glotaran_property_type</code></i>	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

<code>deleter</code>	Descriptor to change the deleter on a property.
<code>getter</code>	Descriptor to change the getter on a property.
<code>glotaran_fill</code>	Fill a property with items from a model and parameters.
<code>glotaran_format_value</code>	Format a value to string.
<code>glotaran_replace_parameter_with_label</code>	Replace parameter values with their full label.
<code>glotaran_validate</code>	Validate a value against a model and optionally against parameters.
<code>glotaran_value_as_markdown</code>	Get a markdown representation of the property.
<code>setter</code>	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

Glotalan 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

<code>interval</code>	ModelProperty is an extension of the property decorator.
<code>parameter</code>	ModelProperty is an extension of the property decorator.
<code>source</code>	ModelProperty is an extension of the property decorator.
<code>target</code>	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

<i>applies</i>	Returns true if <i>value</i> is in one of the intervals.
<i>as_dict</i>	
<i>fill</i>	
<i>from_dict</i>	
<i>markdown</i>	
<i>validate</i>	

## 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

<code>get_subtype</code>	Gets the subscribed type of a generic type.
<code>is_mapping_type</code>	Check if the type is mapping.
<code>is_scalar_type</code>	Check if the type is scalar.
<code>is_sequence_type</code>	Check if the type is a sequence.
<code>wrap_func_as_method</code>	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

<code>ModelError</code>	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

<i>Weight</i>	The <i>Weight</i> 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

<i>datasets</i>	ModelProperty is an extension of the property decorator.
<i>global_interval</i>	ModelProperty is an extension of the property decorator.
<i>model_interval</i>	ModelProperty is an extension of the property decorator.
<i>value</i>	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

<code>glotaran.parameter.parameter</code>	The parameter class.
<code>glotaran.parameter.parameter_group</code>	The parameter group class.
<code>glotaran.parameter.parameter_history</code>	The glotaran parameter history package.

### parameter

The parameter class.

### Classes

#### Summary

<code>Keys</code>	Keys for parameter options.
<code>Parameter</code>	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

<i>expression</i>	Expression to calculate the parameters value from.
<i>full_label</i>	Label of the parameter with its path in a parameter group prepended.
<i>label</i>	Label of the parameter.
<i>maximum</i>	Upper bound of the parameter.
<i>minimum</i>	Lower bound of the parameter.
<i>non_negative</i>	Indicate if the parameter is non-negative.
<i>standard_error</i>	Standard error of the optimized parameter.
<i>transformed_expression</i>	Expression of the parameter transformed for evaluation within a <i>ParameterGroup</i> .
<i>value</i>	Value of the parameter.
<i>vary</i>	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**

<code>as_dict</code>	Create a dictionary containing the parameter properties.
<code>from_dict</code>	Create a <a href="#">Parameter</a> from a dictionary.
<code>from_list_or_value</code>	Create a parameter from a list or numeric value.
<code>get_value_and_bounds_for_optimization</code>	Get the parameter value and bounds with expression and non-negative constraints applied.
<code>markdown</code>	Get a markdown representation of the parameter.
<code>set_from_group</code>	Set all values of the parameter to the values of the corresponding parameter in the group.
<code>set_value_from_optimization</code>	Set the value from an optimization result and reverses non-negative transformation.
<code>valid_label</code>	Check if a label is a valid label for <a href="#">Parameter</a> .

**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

---

<i>ParameterGroup</i>	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

---

<i>label</i>	Label of the group.
<i>root_group</i>	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**

<code>add_group</code>	Add a <code>ParameterGroup</code> to the group.
<code>add_parameter</code>	Add a <code>Parameter</code> to the group.
<code>all</code>	Iterate over all parameter in the group and it's subgroups together with their labels.
<code>clear</code>	
<code>copy</code>	Create a copy of the <code>ParameterGroup</code> .
<code>from_dataframe</code>	Create a <code>ParameterGroup</code> from a <code>pandas.DataFrame</code> .
<code>from_dict</code>	Create a <code>ParameterGroup</code> from a dictionary.
<code>from_list</code>	Create a <code>ParameterGroup</code> from a list.
<code>from_parameter_dict_list</code>	Create a <code>ParameterGroup</code> from a list of parameter dictionaries.
<code>fromkeys</code>	Create a new dictionary with keys from iterable and values set to value.
<code>get</code>	Get a <code>Parameter</code> by its label.
<code>get_group_for_parameter_by_label</code>	Get the group for a parameter by it's label.
<code>get_label_value_and_bounds_arrays</code>	Return a arrays of all parameter labels, values and bounds.
<code>get_nr_roots</code>	Return the number of roots of the group.
<code>groups</code>	Return a generator over all groups and their subgroups.
<code>has</code>	Check if a parameter with the given label is in the group or in a subgroup.
<code>items</code>	
<code>keys</code>	
<code>loader</code>	Create a <code>ParameterGroup</code> instance from the specs defined in a file.
<code>markdown</code>	Format the <code>ParameterGroup</code> as markdown string.

continues on next page

Table 173 – continued from previous page

<code>pop</code>	If key is not found, d is returned if given, otherwise <code>KeyError</code> is raised
<code>popitem</code>	Remove and return a (key, value) pair as a 2-tuple.
<code>set_from_history</code>	Update the <code>ParameterGroup</code> with values from a parameter history.
<code>set_from_label_and_value_arrays</code>	Update the parameter values from a list of labels and values.
<code>setdefault</code>	Insert key with a value of default if key is not in the dictionary.
<code>to_csv</code>	Save a <code>ParameterGroup</code> to a CSV file.
<code>to_dataframe</code>	Create a pandas data frame from the group.
<code>to_parameter_dict_list</code>	Create list of parameter dictionaries from the group.
<code>update</code>	If E is present and has a <code>.keys()</code> method, then does: for k in E: D[k] = E[k] If E is present and lacks a <code>.keys()</code> 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]
<code>update_parameter_expression</code>	Update all parameters which have an expression.
<code>values</code>	

### `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 its 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 size 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:** Deprecate 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** A list 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 its 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 size 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

<code>ParameterNotFoundException</code>	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

<i>ParameterHistory</i>	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

<i>number_of_records</i>	Return the number of records in the history.
<i>parameter_labels</i>	Return the labels of the parameters in the history.
<i>parameters</i>	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**

<i>append</i>	Append a <code>ParameterGroup</code> to the history.
<i>from_csv</i>	Create a history from a csv file.
<i>from_dataframe</i>	Create a history from a pandas data frame.
<i>get_parameters</i>	Get parameters for a history index.
<i>loader</i>	Create a history from a csv file.
<i>to_csv</i>	Write a <code>ParameterGroup</code> to a CSV file.
<i>to_dataframe</i>	Create a data frame from the history.

**append****ParameterHistory.append**(*parameter\_group*:`pyglotaran.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

---

<i>glotaran.plugin_system.base_registry</i>	Functionality to register, initialize and retrieve glotaran plugins.
<i>glotaran.plugin_system.data_io_registration</i>	Data Io registration convenience functions.
<i>glotaran.plugin_system.io_plugin_utils</i>	Utility functions for io plugin.
<i>glotaran.plugin_system.megacomplex_registration</i>	Megacomplex registration convenience functions.
<i>glotaran.plugin_system.project_io_registration</i>	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

<i>add_instantiated_plugin_to_registry</i>	Add instances of <code>plugin_class</code> to the given registry.
<i>add_plugin_to_registry</i>	Add a plugin with name <code>plugin_register_key</code> to the given registry.
<i>full_plugin_name</i>	Full name of a plugin instance/class similar to the <code>repr</code> .
<i>get_method_from_plugin</i>	Retrieve a method callabe from an class or instance plugin.
<i>get_plugin_from_registry</i>	Retrieve a plugin with name <code>plugin_register_key</code> is registered in a given registry.
<i>is_registered_plugin</i>	Check if a plugin with name <code>plugin_register_key</code> is registered in the given registry.
<i>load_plugins</i>	Initialize plugins registered under the entrypoint ‘glotaran.plugins’.
<i>methods_differ_from_baseclass</i>	Check if a plugins methods implementation differ from its baseclass.
<i>methods_differ_from_baseclass_table</i>	Create table of which plugins methods differ from their baseclass.
<i>registered_plugins</i>	Names of the plugins in the given registry.
<i>set_plugin</i>	Set a plugins short name to a specific plugin referred by its full name.
<i>show_method_help</i>	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],
                                                                    plu-
                                                                    gin_class:
                                                                    type[_PluginInstantiableType],
                                                                    plu-
                                                                    gin_registry:
                                                                    MutableMap-
                                                                    ping[str,
                                                                    _Plug-
                                                                    inInstan-
                                                                    tiable-
                                                                    Type],
                                                                    plu-
                                                                    gin_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 callable 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: Generic-
                                                                    PluginInstance |
                                                                    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 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

<code>data_io_plugin_table</code>	Return registered data io plugins and which functions they support as markdown table.
<code>get_data_io</code>	Retrieve a data io plugin from the data_io registry.
<code>get_data_loader</code>	Retrieve implementation of the <code>read_dataset</code> functionality for the format 'format_name'.
<code>get_data_saver</code>	Retrieve implementation of the <code>save_dataset</code> functionality for the format 'format_name'.
<code>is_known_data_format</code>	Check if a data format is in the data_io registry.
<code>known_data_formats</code>	Names of the registered data io plugins.
<code>load_dataset</code>	Read data from a file to <code>xarray.Dataset</code> or <code>xarray.DataArray</code> .
<code>register_data_io</code>	Register data io plugins to one or more formats.
<code>save_dataset</code>	Save data from <code>xarray.Dataset</code> or <code>xarray.DataArray</code> to a file.
<code>set_data_plugin</code>	Set the plugin used for a specific data format.
<code>show_data_io_method_help</code>	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_data_loader` 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:
                                                                    Lit-
                                                                    eral['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

<code>bool_str_repr</code>	Replace boolean value with string repr.
<code>bool_table_repr</code>	Replace boolean value with string repr for all table values.
<code>inferred_file_format</code>	Inferred format of a file if it exists.

continues on next page

Table 182 – continued from previous page

<code>not_implemented_to_value_error</code>	Decorate a function to raise <code>ValueError</code> instead of <code>NotImplementedError</code> .
<code>protect_from_overwrite</code>	Raise <code>FileExistsError</code> if files already exists and <code>allow_overwrite</code> isn't <code>True</code> .

## 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_str_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  /  *
--- - -
```

### inferred\_file\_format

`glotaran.plugin_system.io_plugin_utils.inferred_file_format(file_path: StrOrPath, *, needs_to_exist: bool = True, allow_folder=False) → str`

Inferred 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_utils.io_plugin_utils → glotaran.plugin_system.io_plugin_utils)`

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

<i>get_megacomplex</i>	Retrieve a megacomplex from the megacomplex registry.
<i>is_known_megacomplex</i>	Check if a megacomplex is in the megacomplex registry.
<i>known_megacomplex_names</i>	Names of the registered megacomplexs.
<i>megacomplex_plugin_table</i>	Return registered megacomplex plugins as mark-down table.
<i>register_megacomplex</i>	Add a megacomplex to the megacomplex registry.
<i>set_megacomplex_plugin</i>	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 megacomplexes.

**Parameters** `full_names` (*bool*) – Whether to display the full names the plugins are registered under as well.

**Returns** List of registered megacomplexes.

**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.MarkdownStr`

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, megacom-  
                                                                    plex:  
                                                                    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

<code>get_project_io</code>	Retrieve a data io plugin from the project_io registry.
<code>get_project_io_method</code>	Retrieve implementation of project io functionality for the format 'format_name'.
<code>is_known_project_format</code>	Check if a data format is in the project_io registry.
<code>known_project_formats</code>	Names of the registered project io plugins.
<code>load_model</code>	Create a Model instance from the specs defined in a file.
<code>load_parameters</code>	Create a ParameterGroup instance from the specs defined in a file.
<code>load_result</code>	Create a Result instance from the specs defined in a file.
<code>load_scheme</code>	Create a Scheme instance from the specs defined in a file.
<code>project_io_plugin_table</code>	Return registered project io plugins and which functions they support as markdown table.
<code>register_project_io</code>	Register project io plugins to one or more formats.
<code>save_model</code>	Save a Model instance to a spec file.
<code>save_parameters</code>	Save a ParameterGroup instance to a spec file.
<code>save_result</code>	Write a Result instance to a spec file.
<code>save_scheme</code>	Save a Scheme instance to a spec file.
<code>set_project_plugin</code>	Set the plugin used for a specific project format.
<code>show_project_io_method_help</code>	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(*, plu-  
gin_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:  
                                                                            Pro-  
                                                                            jec-  
                                                                            tIoMeth-  
                                                                            ods)  
                                                                            →  
                                                                            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**

<i>glotaran.project.dataclass_helpers</i>	Contains helper methods for dataclasses.
<i>glotaran.project.result</i>	The result class for global analysis.
<i>glotaran.project.scheme</i>	The module for :class:Scheme.

**dataclass\_helpers**

Contains helper methods for dataclasses.

**Functions****Summary**

<i>asdict</i>	Create a dictionary containing all fields of the dataclass.
<i>exclude_from_dict_field</i>	Create a dataclass field with which will be excluded from asdict.
<i>file_loadable_field</i>	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

<code>file_loader_factory</code>	Create <code>file_loader</code> functions to load <code>targetClass</code> from file.
<code>fromdict</code>	Create a dataclass instance from a dict and loads all file represented fields.
<code>init_file_loadable_fields</code>	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 instantiating 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

<i>Result</i>	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

<code>additional_penalty</code>	A vector with the value for each additional penalty, or None
<code>chi_square</code>	The chi-square of the optimization.
<code>cost</code>	The final cost.
<code>covariance_matrix</code>	Covariance matrix.
<code>degrees_of_freedom</code>	Degrees of freedom in optimization $N - N_{vars}$ .
<code>jacobian</code>	Modified Jacobian matrix at the solution
<code>model</code>	Return the model used to fit result.
<code>number_of_data_points</code>	Number of data points $N$ .
<code>number_of_jacobian_evaluations</code>	The number of jacobian evaluations.
<code>number_of_variables</code>	Number of variables in optimization $N_{vars}$
<code>optimality</code>	
<code>reduced_chi_square</code>	The reduced chi-square of the optimization.
<code>root_mean_square_error</code>	The root mean square error the optimization.
<code>source_path</code>	
<code>number_of_function_evaluations</code>	The number of function evaluations.
<code>success</code>	Indicates if the optimization was successful.
<code>termination_reason</code>	The reason (message when) the optimizer terminated
<code>glotaran_version</code>	The glotaran version used to create the result.
<code>free_parameter_labels</code>	List of labels of the free parameters used in optimization.
<code>scheme</code>	
<code>initial_parameters</code>	
<code>optimized_parameters</code>	
<code>parameter_history</code>	The parameter history.
<code>data</code>	The resulting data as a dictionary of <code>xarray.Dataset</code> .

### `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.

$$\chi^2 = \sum_i^N [\textit{Residual}_i]^2.$$

### 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.

$$\chi_{red}^2 = \chi^2 / (N - N_{vars}).$$

**root\_mean\_square\_error**

`Result.root_mean_square_error: float | None = None`

The root mean square error the optimization.

$$rms = \sqrt{\chi_{red}^2}$$

**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

<code>get_dataset</code>	Return the result dataset for the given dataset label.
<code>get_scheme</code>	Return a new scheme from the Result object with optimized parameters.
<code>loader</code>	Create a <a href="#"><i>Result</i></a> instance from the specs defined in a file.
<code>markdown</code>	Format the model as a markdown text.
<code>recreate</code>	Recreate a result from the initial parameters.
<code>save</code>	Save the result to given folder.
<code>verify</code>	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()**  $\rightarrow$  `glotaran.project.result.Result`

Recreate 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.

$$\chi_{red}^2 = \chi^2 / (N - N_{vars}).$$

**root\_mean\_square\_error:** `float | None = None`

The root mean square error the optimization.

$$rms = \sqrt{\chi_{red}^2}$$

**save(path: str)**  $\rightarrow$  `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()**  $\rightarrow$  `bool`

Verify a result.

**Returns** Whether 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

<code>add_svd</code>	
<code>clp_link_tolerance</code>	
<code>ftol</code>	
<code>global_dimensions</code>	Return the dataset model's global dimension.
<code>group</code>	
<code>group_tolerance</code>	
<code>gtol</code>	
<code>maximum_number_function_evaluations</code>	
<code>model_dimensions</code>	Return the dataset model's model dimension.
<code>non_negative_least_squares</code>	
<code>optimization_method</code>	
<code>result_path</code>	
<code>source_path</code>	
<code>xtol</code>	
<code>model</code>	
<code>parameters</code>	
<code>data</code>	

**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

<code>from_yaml_file</code>	Create <i>Scheme</i> from specs in yaml file.
<code>loader</code>	Create a <i>Scheme</i> instance from the specs defined in a file.
<code>markdown</code>	Format the <i>Scheme</i> as markdown string.
<code>problem_list</code>	Return a list with all problems in the model and missing parameters.
<code>valid</code>	Check if there are no problems with the model or the parameters.
<code>validate</code>	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 scheme

**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

<code>glotaran.testing.model_generators</code>	Model generators used to generate simple models from a set of inputs.
<code>glotaran.testing.plugin_system</code>	Mock functionality for the plugin system.

#### model\_generators

Model generators used to generate simple models from a set of inputs.

#### Classes

##### Summary

<code>SimpleModelGenerator</code>	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

<code>compartments</code>	A list of compartment names
<code>default_megacomplex</code>	The default_megacomplex identifier
<code>dispersion_center</code>	A value representing the dispersion center
<code>k_matrix</code>	"A <i>dict</i> with a k_matrix specification or <i>Literal</i> ['parallel', 'sequential']"
<code>model</code>	Return the generated model.
<code>model_and_parameters</code>	Return generated model and parameters.
<code>model_dict</code>	Return a dict representation of the generated model.
<code>parameters</code>	Return the generated parameters of type <code>glotaran.parameter.ParameterGroup</code> .
<code>parameters_dict</code>	Return a dict representation of the generated parameters.
<code>valid</code>	Check if the generator state is valid.
<code>rates</code>	A list of values representing decay rates
<code>irf</code>	A dict of items specifying an irf
<code>initial_concentration</code>	A list values representing the initial concentration
<code>dispersion_coefficients</code>	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

<code>markdown</code>	Return a markdown string representation of the generated model and parameters.
<code>validate</code>	Call <i>validate</i> 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

<code>monkeypatch_plugin_registry</code>	Contextmanager to monkeypatch multiple plugin registries at once.
<code>monkeypatch_plugin_registry_data_io</code>	Monkeypatch the DataIoInterface registry.
<code>monkeypatch_plugin_registry_megacomplex</code>	Monkeypatch the Megacomplex registry.
<code>monkeypatch_plugin_registry_project_io</code>	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**

<i>glotaran.typing.protocols</i>	Protocol like type definitions.
<i>glotaran.typing.types</i>	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

<code>glotaran.utils.io</code>	Glotaran IO utility module.
<code>glotaran.utils.ipython</code>	Glotaran module with utilities for ipython integration (e.g.
<code>glotaran.utils.regex</code>	Glotaran module with regular expression patterns and functions.
<code>glotaran.utils.sanitize</code>	Glotaran module with utilities for sanitation of parsed content.

## io

Glotaran IO utility module.

## Functions

### Summary

<code>load_datasets</code>	Load multiple datasets into a mapping (convenience function).
<code>relative_posix_path</code>	Ensure that <code>source_path</code> is a posix path, relative to <code>base_path</code> 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 have 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.

On Windows if `source_path` and `base_path` are on different drives, it will return the absolute posix path to the file.

### 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

---

<i>DatasetMapping</i>	Wrapper class for a mapping of datasets which can be used for a <code>file_loadable_field</code> .
-----------------------	--

---

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

<i>source_path</i>	Map the <code>source_path</code> 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

<i>clear</i>	
<i>get</i>	
<i>items</i>	
<i>keys</i>	
<i>loader</i>	Loader function utilized by <code>file_loadable_field</code> .
<i>pop</i>	If key is not found, d is returned if given, otherwise <code>KeyError</code> is raised.
<i>popitem</i>	as a 2-tuple; but raise <code>KeyError</code> if D is empty.
<i>setdefault</i>	

continues on next page

Table 209 – continued from previous page

<i>update</i>	If E present and has a <code>.keys()</code> method, does: for k in E: $D[k] = E[k]$ If E present and lacks <code>.keys()</code> method, does: for (k, v) in E: $D[k] = v$ In either case, this is followed by: for k, v in F.items(): $D[k] = v$
<i>values</i>	

---

**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

---

<code>display_file</code>	Display a file with syntax highlighting <code>syntax</code> .
---------------------------	---

---

### 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

<i>MarkdownStr</i>	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

<i>capitalize</i>	
<i>casefold</i>	
<i>center</i>	
<i>count</i>	
<i>encode</i>	
<i>endswith</i>	
<i>expandtabs</i>	
<i>find</i>	
<i>format</i>	
<i>format_map</i>	
<i>index</i>	Raises ValueError if the value is not present.
<i>isalnum</i>	

continues on next page

Table 212 – continued from previous page

<i>isalpha</i>	
<i>isascii</i>	
<i>isdecimal</i>	
<i>isdigit</i>	
<i>isidentifier</i>	
<i>islower</i>	
<i>isnumeric</i>	
<i>isprintable</i>	
<i>isspace</i>	
<i>istitle</i>	
<i>isupper</i>	
<i>join</i>	
<i>ljust</i>	
<i>lower</i>	
<i>lstrip</i>	
<i>maketrans</i>	Return a translation table usable for <code>str.translate()</code> .
<i>partition</i>	
<i>replace</i>	
<i>rfind</i>	
<i>rindex</i>	
<i>rjust</i>	
<i>rpartition</i>	
<i>rsplit</i>	
<i>rstrip</i>	
<i>split</i>	

---

continues on next page

Table 212 – continued from previous page

<i>splitlines</i>
<i>startswith</i>
<i>strip</i>
<i>swapcase</i>
<i>title</i>
<i>translate</i>
<i>upper</i>
<i>zfill</i>

**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*, \*\**kws*)

**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, \*\*kws)**

**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

<i>RegexPattern</i>	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

---

<i>number_scientific</i>
<i>tuple_number</i>
<i>tuple_word</i>
<i>word</i>

---

**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

<i>convert_scientific_to_float</i>	Convert value to float if it matches scientific notation string.
<i>list_string_to_tuple</i>	Convert a list of strings (representing tuples) to a list of tuples.
<i>sanitize_dict_keys</i>	Sanitize the stringified tuple dict keys in a yaml parsed dict.
<i>sanitize_dict_values</i>	Sanitizes a dict with broken tuples inside modifying it in-place.

continues on next page

Table 216 – continued from previous page

<code>sanitize_list_with_broken_tuples</code>	Sanitize a list with 'broken' tuples.
<code>sanitize_parameter_list</code>	Replace in a list strings matching scientific notation with floats.
<code>sanitize_yaml</code>	Sanitize a yaml-returned dict for key or (list) values containing tuples.
<code>sanity_scientific_notation_conversion</code>	Convert scientific notation string values to floats.
<code>string_to_tuple</code>	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 repressed 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 yaml 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*



## 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 documentation <[https://pyglotaran.readthedocs.io/en/latest/user\\_documentation/using\\_plugins.html](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_names`

```
[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 -0.006294 -0.01641 0.01701 ... 1.689 1.54
Attributes:
  source_path:  dataset_1.nc
```

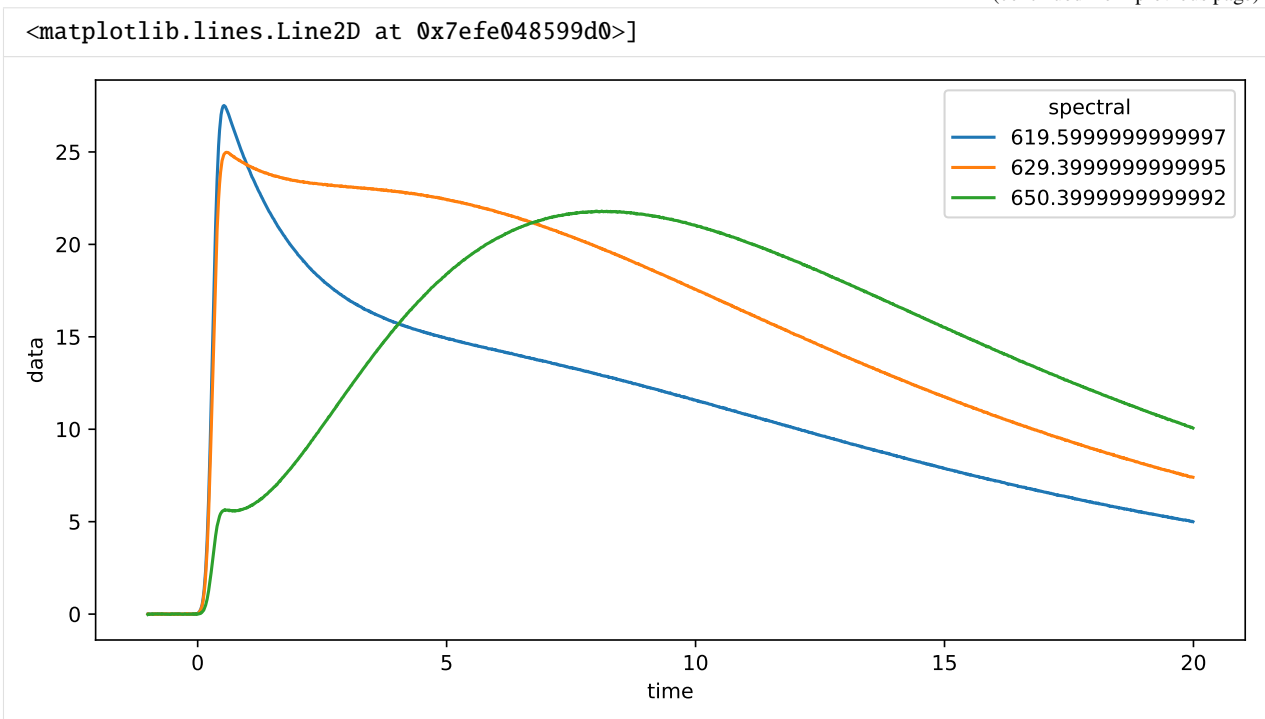
To get a feeling for our data, let's plot some traces.

```
[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 0x7efe04859910>,
<matplotlib.lines.Line2D at 0x7efe048598b0>]
```

(continues on next page)

(continued from previous page)



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)

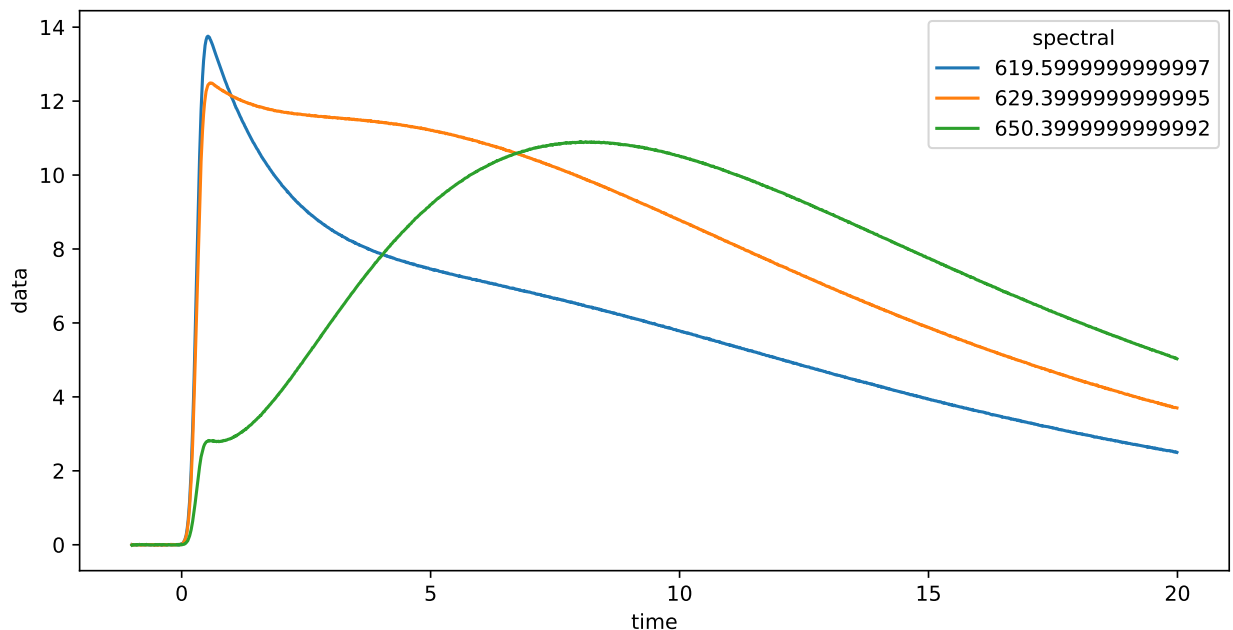
```

* 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 -0.003147 -0.008204 ... 0.8444 0.7698
Attributes:
  loader:    <function load_dataset at 0x7efe125a8310>
  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 0x7efdfef487f70>,
<matplotlib.lines.Line2D at 0x7efdfef487fd0>,
<matplotlib.lines.Line2D at 0x7efdfef498130>]
```



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 works analogous:

	DataIo plugin	ProjectIo plugin
Register function	<code>glotaran.plugin_system.data_io_registration.register_data_io</code>	<code>glotaran.plugin_system.project_io_registration.register_project_io</code>
Base-class	<code>glotaran.io.interface.DataIoInterface</code>	<code>glotaran.io.interface.DataIoInterface</code>
Possible methods	<code>load_dataset</code> , <code>save_dataset</code>	<code>load_model</code> , <code>save_model</code> , <code>load_parameters</code> , <code>save_parameters</code> , <code>load_scheme</code> , <code>save_scheme</code> , <code>load_result</code> , <code>save_result</code>

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.





## INDICES AND TABLES

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



## BIBLIOGRAPHY

- [1] [https://glotaran.github.io/legacy/file\\_formats](https://glotaran.github.io/legacy/file_formats)
- [1] [https://glotaran.github.io/legacy/file\\_formats](https://glotaran.github.io/legacy/file_formats)



## PYTHON MODULE INDEX

### g

glotaran, 53  
glotaran.analysis, 54  
glotaran.analysis.nnls, 54  
glotaran.analysis.optimization\_group, 55  
glotaran.analysis.optimization\_group\_calculator, 59  
glotaran.analysis.optimization\_group\_calculator\_linked, 58  
glotaran.analysis.optimization\_group\_calculator\_unlinked, 61  
glotaran.analysis.optimize, 72  
glotaran.analysis.simulation, 73  
glotaran.analysis.util, 74  
glotaran.analysis.variable\_projection, 78  
glotaran.builtin, 79  
glotaran.builtin.io, 79  
glotaran.builtin.io.ascii, 79  
glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file, 79  
glotaran.builtin.io.csv, 91  
glotaran.builtin.io.csv.csv, 91  
glotaran.builtin.io.folder, 94  
glotaran.builtin.io.folder.folder\_plugin, 94  
glotaran.builtin.io.netCDF, 99  
glotaran.builtin.io.netCDF.netCDF, 99  
glotaran.builtin.io.sdt, 100  
glotaran.builtin.io.sdt.sdt\_file\_reader, 100  
glotaran.builtin.io.yml, 102  
glotaran.builtin.io.yml.yml, 102  
glotaran.builtin.megacomplexes, 106  
glotaran.builtin.megacomplexes.baseline, 106  
glotaran.builtin.megacomplexes.baseline.baseline\_megacomplex, 106  
glotaran.builtin.megacomplexes.coherent\_artifact, 111  
glotaran.builtin.megacomplexes.coherent\_artifact.coherent\_artifact\_megacomplex, 111  
glotaran.builtin.megacomplexes.damped\_oscillation, 116  
glotaran.builtin.megacomplexes.damped\_oscillation.damped\_oscillation\_megacomplex, 116  
glotaran.builtin.megacomplexes.decay, 123  
glotaran.builtin.megacomplexes.decay.decay\_megacomplex, 123  
glotaran.builtin.megacomplexes.decay.initial\_concentration, 129  
glotaran.builtin.megacomplexes.decay.irf, 132  
glotaran.builtin.megacomplexes.decay.k\_matrix, 158  
glotaran.builtin.megacomplexes.decay.util, 164  
glotaran.builtin.megacomplexes.spectral, 167  
glotaran.builtin.megacomplexes.spectral.shape, 167  
glotaran.builtin.megacomplexes.spectral.spectral\_megacomplex, 181  
glotaran.cli, 185  
glotaran.cli.commands, 186  
glotaran.cli.commands.explore, 186  
glotaran.cli.commands.export, 186  
glotaran.cli.commands.optimize, 186  
glotaran.cli.commands.pluginlist, 187  
glotaran.cli.commands.print, 187  
glotaran.cli.commands.util, 187  
glotaran.cli.commands.validate, 193  
glotaran.deprecation, 194  
glotaran.deprecation.deprecation\_utils, 194  
glotaran.deprecation.modules, 204  
glotaran.deprecation.modules.builtin\_io.yml, 205  
glotaran.deprecation.modules.glotaran\_root, 205  
glotaran.examples, 207  
glotaran.examples.sequential, 208  
glotaran.io, 208  
glotaran.io.interface, 208  
glotaran.io.prepare\_dataset, 213  
glotaran.model, 214  
glotaran.model.coherent\_artifact\_megacomplex, 214  
glotaran.model.clp\_penalties, 214  
glotaran.model.constraint, 219  
glotaran.model.dataset\_group, 225  
glotaran.model.dataset\_model, 227  
glotaran.model.damped\_oscillation\_megacomplex, 227  
glotaran.model.interval\_property, 232

- glotaran.model.item, [235](#)
- glotaran.model.model, [236](#)
- glotaran.model.property, [242](#)
- glotaran.model.relation, [249](#)
- glotaran.model.util, [253](#)
- glotaran.model.weight, [255](#)
- glotaran.parameter, [258](#)
- glotaran.parameter.parameter, [258](#)
- glotaran.parameter.parameter\_group, [268](#)
- glotaran.parameter.parameter\_history, [282](#)
- glotaran.plugin\_system, [286](#)
- glotaran.plugin\_system.base\_registry, [287](#)
- glotaran.plugin\_system.data\_io\_registration,  
[295](#)
- glotaran.plugin\_system.io\_plugin\_utils, [299](#)
- glotaran.plugin\_system.megacomplex\_registration,  
[302](#)
- glotaran.plugin\_system.project\_io\_registration,  
[304](#)
- glotaran.project, [314](#)
- glotaran.project.dataclass\_helpers, [314](#)
- glotaran.project.result, [317](#)
- glotaran.project.scheme, [326](#)
- glotaran.testing, [333](#)
- glotaran.testing.model\_generators, [333](#)
- glotaran.testing.plugin\_system, [339](#)
- glotaran.typing, [341](#)
- glotaran.typing.protocols, [342](#)
- glotaran.typing.types, [343](#)
- glotaran.utils, [343](#)
- glotaran.utils.io, [343](#)
- glotaran.utils.ipython, [348](#)
- glotaran.utils.regex, [359](#)
- glotaran.utils.sanitize, [361](#)

## Symbols

--data <data>  
     glotaran-optimize command line option, 41  
 --dataformat <dataformat>  
     glotaran-optimize command line option, 41  
 --model\_file <model\_file>  
     glotaran-optimize command line option, 42  
     glotaran-print command line option, 42  
     glotaran-validate command line option, 43  
 --nfev <nfev>  
     glotaran-optimize command line option, 41  
 --nnls  
     glotaran-optimize command line option, 41  
 --out <out>  
     glotaran-optimize command line option, 41  
 --outformat <outformat>  
     glotaran-optimize command line option, 41  
 --parameters\_file <parameters\_file>  
     glotaran-optimize command line option, 42  
     glotaran-print command line option, 42  
     glotaran-validate command line option, 43  
 --version  
     glotaran command line option, 41  
 --yes  
     glotaran-optimize command line option, 42  
 -d  
     glotaran-optimize command line option, 41  
 -dfmt  
     glotaran-optimize command line option, 41  
 -m  
     glotaran-optimize command line option, 42  
     glotaran-print command line option, 42  
     glotaran-validate command line option, 43  
 -n  
     glotaran-optimize command line option, 41  
 -o  
     glotaran-optimize command line option, 41  
 -ofmt  
     glotaran-optimize command line option, 41  
 -p  
     glotaran-optimize command line option, 42  
     glotaran-print command line option, 42

glotaran-validate command line option, 43  
 -y  
     glotaran-optimize command line option, 42

## A

a\_matrix() (*glotaran.builtin.megacomplexes.decay.k\_matrix.KMatrix* method), 162  
 a\_matrix\_as\_markdown()  
     (*glotaran.builtin.megacomplexes.decay.k\_matrix.KMatrix* method), 162  
 a\_matrix\_non\_unibranch()  
     (*glotaran.builtin.megacomplexes.decay.k\_matrix.KMatrix* method), 162  
 a\_matrix\_unibranch()  
     (*glotaran.builtin.megacomplexes.decay.k\_matrix.KMatrix* method), 163  
 add\_data\_row() (*glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.T* method), 87  
 add\_data\_row() (*glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.W* method), 90  
 add\_group() (*glotaran.parameter.parameter\_group.ParameterGroup* method), 277  
 add\_instantiated\_plugin\_to\_registry() (in module *glotaran.plugin\_system.base\_registry*), 288  
 add\_parameter() (*glotaran.parameter.parameter\_group.ParameterGroup* method), 277  
 add\_plugin\_to\_registry() (in module *glotaran.plugin\_system.base\_registry*), 288  
 add\_svd (*glotaran.project.scheme.Scheme* attribute), 331  
 add\_svd\_to\_dataset() (in module *glotaran.io.prepare\_dataset*), 213  
 add\_type() (*glotaran.builtin.megacomplexes.decay.irf.Irf* class method), 133  
 add\_type() (*glotaran.builtin.megacomplexes.spectral.shape.SpectralShape* class method), 168  
 add\_type() (*glotaran.model.constraint.Constraint* class method), 220  
 additional\_penalty (*glotaran.analysis.optimization\_group.Optimization* property), 58  
 additional\_penalty (*glotaran.project.result.Result* attribute), 324  
 all() (*glotaran.parameter.parameter\_group.ParameterGroup*

method), 277

amplitude (glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeGaussian property), 171

amplitude (glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeSkewedGaussian property), 177

append() (glotaran.parameter.parameter\_history.ParameterHistory method), 218

method), 285

applies() (glotaran.model.clp\_penalties.EqualAreaPenalty method), 218

applies() (glotaran.model.constraint.OnlyConstraint method), 222

applies() (glotaran.model.constraint.ZeroConstraint method), 224

applies() (glotaran.model.interval\_property.IntervalProperty method), 234

applies() (glotaran.model.relation.Relation method), 252

apply\_constraints() (in module glotaran.analysis.util), 75

apply\_relations() (in module glotaran.analysis.util), 75

apply\_spectral\_penalties() (in module glotaran.model.clp\_penalties), 214

apply\_weight() (in module glotaran.analysis.util), 75

arity (glotaran.cli.commands.util.ValOrRangeOrList attribute), 192

as\_dict() (glotaran.builtin.megacomplexes.baseline.baseline\_megacomplex method), 110

as\_dict() (glotaran.builtin.megacomplexes.coherent\_artifact.coherent\_artifact\_megacomplex method), 115

as\_dict() (glotaran.builtin.megacomplexes.damped\_oscillation.damped\_oscillation\_megacomplex method), 122

as\_dict() (glotaran.builtin.megacomplexes.decay.decay\_megacomplex method), 128

as\_dict() (glotaran.builtin.megacomplexes.decay.initial\_concentration.initial\_concentration\_megacomplex method), 131

as\_dict() (glotaran.builtin.megacomplexes.decay.irf.IrfGaussian property), 137

as\_dict() (glotaran.builtin.megacomplexes.decay.irf.IrfMeasured property), 140

as\_dict() (glotaran.builtin.megacomplexes.decay.irf.IrfMultiGaussian property), 144

as\_dict() (glotaran.builtin.megacomplexes.decay.irf.IrfSpectralGaussian property), 150

as\_dict() (glotaran.builtin.megacomplexes.decay.irf.IrfSpectralMultiGaussian property), 156

as\_dict() (glotaran.builtin.megacomplexes.decay.irf.IrfSpectralSkewedGaussian property), 169

as\_dict() (glotaran.builtin.megacomplexes.decay.k\_matrix.KMatrix method), 163

as\_dict() (glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeGaussian method), 171

as\_dict() (glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeOne method), 173

as\_dict() (glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeSkewedGaussian method), 177

as\_dict() (glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeTwo method), 180

as\_dict() (glotaran.builtin.megacomplexes.spectral.spectral\_megacomplex method), 187

as\_dict() (glotaran.model.clp\_penalties.EqualAreaPenalty method), 218

as\_dict() (glotaran.model.constraint.OnlyConstraint method), 222

as\_dict() (glotaran.model.constraint.ZeroConstraint method), 224

as\_dict() (glotaran.model.interval\_property.IntervalProperty method), 234

as\_dict() (glotaran.model.model.Model method), 240

as\_dict() (glotaran.model.relation.Relation method), 252

as\_dict() (glotaran.model.weight.Weight method), 257

as\_dict() (glotaran.parameter.parameter.Parameter method), 265

AsciiDataIo (class in glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file), 80

asdict() (in module glotaran.project.dataclass\_helpers), 315

axis (glotaran.analysis.optimization\_group\_calculator\_linked.DatasetIndex attribute), 63

## B

BaselineMegacomplex (class in glotaran.builtin.megacomplexes.baseline.baseline\_megacomplex), 106

backsweep (glotaran.builtin.megacomplexes.decay.irf.IrfGaussian property), 137

backsweep (glotaran.builtin.megacomplexes.decay.irf.IrfMultiGaussian property), 144

backsweep (glotaran.builtin.megacomplexes.decay.irf.IrfSpectralGaussian property), 150

backsweep (glotaran.builtin.megacomplexes.decay.irf.IrfSpectralMultiGaussian property), 156

backsweep\_period (glotaran.builtin.megacomplexes.decay.irf.IrfGaussian property), 137

backsweep\_period (glotaran.builtin.megacomplexes.decay.irf.IrfMultiGaussian property), 144

backsweep\_period (glotaran.builtin.megacomplexes.decay.irf.IrfSpectralGaussian property), 150

backsweep\_period (glotaran.builtin.megacomplexes.decay.irf.IrfSpectralMultiGaussian property), 156

bag (glotaran.analysis.optimization\_group\_calculator\_linked.OptimizationGroup attribute), 69

BaselineMegacomplex (class in glotaran.builtin.megacomplexes.baseline.baseline\_megacomplex), 106

bool\_list\_repr() (in module glotaran.plugin\_system.io\_plugin\_utils), 300

bool\_table\_repr() (in module glotaran.plugin\_system.io\_plugin\_utils), 300



## C

`calculate()` (`glotaran.builtin.megacomplexes.decay.irf.IrfGaussian` method), 61  
`calculate_matrices()` (`glotaran.analysis.optimization_group_calculator_linked.OptimizationGroupCalculatorLinked` method), 69  
`calculate()` (`glotaran.builtin.megacomplexes.decay.irf.IrfMultiGaussian` method), 144  
`calculate_matrices()` (`glotaran.analysis.optimization_group_calculator_unlinked.OptimizationGroupCalculatorUnlinked` method), 72  
`calculate()` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralGaussian` method), 150  
`calculate_matrix()` (`glotaran.builtin.megacomplexes.baseline.baseline` method), 110  
`calculate()` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeGaussian` method), 115  
`calculate()` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeOne` method), 122  
`calculate()` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeZero` method), 128  
`calculate()` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeSkewedGaussian` method), 184  
`calculate_clip_penalties()` (in module `calculate_matrix()` (in module `glotaran.analysis.util`), 75  
`calculate_damped_oscillation_matrix_gaussian_irf()` (`glotaran.analysis.optimization_group_calculator.OptimizationGroupCalculator` method), 61  
`calculate_damped_oscillation_matrix_no_irf()` (`glotaran.analysis.optimization_group_calculator_linked.OptimizationGroupCalculatorLinked` method), 69  
`calculate_decay_matrix_gaussian_irf()` (in module `glotaran.builtin.megacomplexes.decay.util`), 165  
`calculate_decay_matrix_no_irf()` (in module `glotaran.builtin.megacomplexes.decay.util`), 165  
`calculate_dispersion()` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralGaussian` method), 150  
`calculate_dispersion()` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralMultiGaussian` method), 156  
`calculate_full_penalty()` (`glotaran.analysis.optimization_group_calculator.OptimizationGroupCalculator` method), 61  
`calculate_full_penalty()` (`glotaran.analysis.optimization_group_calculator_linked.OptimizationGroupCalculatorLinked` method), 69  
`calculate_full_penalty()` (`glotaran.analysis.optimization_group_calculator_unlinked.OptimizationGroupCalculatorUnlinked` method), 72  
`calculate_index_dependent_matrices()` (`glotaran.analysis.optimization_group_calculator_linked.OptimizationGroupCalculatorLinked` method), 69  
`calculate_index_independent_matrices()` (`glotaran.analysis.optimization_group_calculator_linked.OptimizationGroupCalculatorLinked` method), 69  
`calculate_matrices()` (`glotaran.analysis.optimization_group_calculator.OptimizationGroupCalculator` method), 61  
`calculate_matrices()` (`glotaran.analysis.optimization_group_calculator_unlinked.OptimizationGroupCalculatorUnlinked` method), 72  
`calculate_matrix()` (`glotaran.builtin.megacomplexes.baseline.baseline` method), 110  
`calculate_matrix()` (`glotaran.builtin.megacomplexes.coherent_artifact` method), 115  
`calculate_matrix()` (`glotaran.builtin.megacomplexes.damped_oscillation` method), 122  
`calculate_matrix()` (`glotaran.builtin.megacomplexes.decay.decay_mega` method), 128  
`calculate_matrix()` (`glotaran.builtin.megacomplexes.spectral.spectral` method), 184  
`calculate_residual()` (`glotaran.analysis.optimization_group_calculator.OptimizationGroupCalculator` method), 61  
`calculate_residual()` (`glotaran.analysis.optimization_group_calculator_linked.OptimizationGroupCalculatorLinked` method), 69  
`calculate_residual()` (`glotaran.analysis.optimization_group_calculator_unlinked.OptimizationGroupCalculatorUnlinked` method), 72  
`CalculatedMatrix` (class in `glotaran.analysis.util`), 77  
`capitalize()` (`glotaran.utils.ipython.MarkdownStr` method), 357  
`casefold()` (`glotaran.utils.ipython.MarkdownStr` method), 357  
`center` (`glotaran.builtin.megacomplexes.decay.irf.IrfGaussian` property), 137  
`center` (`glotaran.builtin.megacomplexes.decay.irf.IrfMultiGaussian` property), 144  
`center` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralGaussian` property), 150  
`center` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralMultiGaussian` property), 157  
`center()` (`glotaran.utils.ipython.MarkdownStr` method), 357  
`center_dispersion_coefficients` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralGaussian` property), 150  
`center_dispersion_coefficients` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralMultiGaussian` property), 157  
`check_overdue()` (in module `glotaran.deprecation.deprecation_utils`), 195  
`check_qualnames_in_tests()` (in module `glotaran.deprecation.deprecation_utils`), 195

195  
 chi\_square (glotaran.project.result.Result attribute), 324  
 clear() (glotaran.parameter.parameter\_group.ParameterGroup method), 277  
 clear() (glotaran.utils.io.DatasetMapping method), 347  
 clp\_labels (glotaran.analysis.util.CalculatedMatrix attribute), 78  
 clp\_link\_tolerance (glotaran.project.scheme.Scheme attribute), 331  
 clps (glotaran.analysis.optimization\_group.OptimizationGroup property), 58  
 CoherentArtifactMegacomplex (class in glotaran.builtin.megacomplexes.coherent\_artifact.coherent\_artifact\_megacomplex), 111  
 combine() (glotaran.builtin.megacomplexes.decay.k\_matrix.KMatrix method), 163  
 combine\_matrices() (in module glotaran.analysis.optimization\_group\_calculator\_linked), 61  
 combine\_matrix() (in module glotaran.analysis.util), 75  
 compartments (glotaran.builtin.megacomplexes.decay.initial\_concentration.InitialConcentration property), 131  
 compartments (glotaran.testing.model\_generators.SimpleModelGenerator attribute), 337  
 compartments() (glotaran.builtin.megacomplexes.coherent\_artifact.coherent\_artifact\_megacomplex method), 115  
 Constraint (class in glotaran.model.constraint), 219  
 convert() (glotaran.cli.commands.util.ValOrRangeOrList method), 192  
 convert\_scientific\_to\_float() (in module glotaran.utils.sanitize), 362  
 copy() (glotaran.parameter.parameter\_group.ParameterGroup method), 277  
 cost (glotaran.analysis.optimization\_group.OptimizationGroup property), 58  
 cost (glotaran.project.result.Result attribute), 324  
 count() (glotaran.analysis.optimization\_group\_calculator\_linked.Dataset method), 63  
 count() (glotaran.analysis.optimization\_group\_calculator\_linked.DatasetIndexModel method), 66  
 count() (glotaran.analysis.util.CalculatedMatrix method), 78  
 count() (glotaran.utils.ipython.MarkdownStr method), 357  
 covariance\_matrix (glotaran.project.result.Result attribute), 324  
 create\_dataset\_model\_type() (in module glotaran.model.dataset\_model), 227  
 create\_index\_dependent\_result\_dataset() (glotaran.analysis.optimization\_group\_calculator\_linked.OptimizationGroup method), 61  
 create\_index\_dependent\_result\_dataset() (glotaran.analysis.optimization\_group\_calculator\_linked.OptimizationGroup method), 69  
 create\_index\_independent\_result\_dataset() (glotaran.analysis.optimization\_group\_calculator.OptimizationGroup method), 61  
 create\_index\_independent\_result\_dataset() (glotaran.analysis.optimization\_group\_calculator\_linked.OptimizationGroup method), 69  
 create\_index\_independent\_result\_dataset() (glotaran.analysis.optimization\_group\_calculator\_unlinked.OptimizationGroup method), 72  
 create\_result\_data() (glotaran.analysis.optimization\_group\_calculator\_unlinked.OptimizationGroup method), 72  
 create\_result\_data() (glotaran.analysis.optimization\_group.OptimizationGroup method), 58  
 create\_result\_dataset() (glotaran.analysis.optimization\_group\_calculator\_linked.OptimizationGroup method), 58  
 CsvProjectIo (class in glotaran.builtin.io.csv.csv), 91  
**D**  
 DampedOscillationMegacomplex (class in glotaran.builtin.megacomplexes.damped\_oscillation.damped\_oscillation\_megacomplex), 118  
 data (glotaran.analysis.optimization\_group.OptimizationGroup property), 58  
 data (glotaran.analysis.optimization\_group\_calculator\_linked.DatasetIndexModel attribute), 66  
 data (glotaran.project.result.Result attribute), 324  
 data (glotaran.project.scheme.Scheme attribute), 331  
 data\_filter (glotaran.plugin\_system.project\_io\_registration.SavingOptions attribute), 314  
 data\_format (glotaran.plugin\_system.project\_io\_registration.SavingOptions attribute), 314  
 data\_io\_plugin\_table() (in module glotaran.plugin\_system.data\_io\_registration), 314  
 data\_sizes (glotaran.analysis.optimization\_group\_calculator\_linked.DatasetIndexModel attribute), 66  
 DataFileType (class in glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file), 82  
 DataIoInterface (class in glotaran.io.interface), 208  
 dataset() (glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.ExplicitFile method), 84  
 dataset() (glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.TimeExplicitFile method), 87  
 dataset() (glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.WavelengthTimeExplicitFile method), 90  
 dataset\_group\_models (glotaran.model.model.Model property), 240

[dataset\\_models \(glotaran.analysis.optimization\\_group.OptimizationGroup property\)](#), 58  
[dataset\\_models \(glotaran.analysis.optimization\\_group\\_calculator\\_linked\\_dataset\\_index\\_model\\_group attribute\)](#), 66  
[dataset\\_models \(glotaran.model.dataset\\_group.DatasetGroup property\)](#), 150  
[dataset\\_models \(glotaran.model.dataset\\_group.DatasetGroup attribute\)](#), 226  
[DatasetGroup \(class in glotaran.model.dataset\\_group\)](#), 225  
[DatasetGroupModel \(class in glotaran.model.dataset\\_group\)](#), 226  
[DatasetIndexModel \(class in glotaran.analysis.optimization\\_group\\_calculator\\_linked\)](#), 62  
[DatasetIndexModelGroup \(class in glotaran.analysis.optimization\\_group\\_calculator\\_linked\)](#), 64  
[DatasetMapping \(class in glotaran.utils.io\)](#), 345  
[DatasetModel \(class in glotaran.model.dataset\\_model\)](#), 228  
[datasets \(glotaran.model.weight.Weight property\)](#), 257  
[decay\\_matrix\\_implementation\(\) \(in module glotaran.builtin.megacomplexes.decay.util\)](#), 165  
[DecayMegacomplex \(class in glotaran.builtin.megacomplexes.decay.decay\\_megacomplex\)](#), 124  
[default\\_megacomplex \(glotaran.model.model.Model property\)](#), 240  
[default\\_megacomplex \(glotaran.testing.model\\_generators.SimpleModelGenerator attribute\)](#), 337  
[degrees\\_of\\_freedom \(glotaran.project.result.Result attribute\)](#), 324  
[deleter\(\) \(glotaran.model.property.ModelProperty method\)](#), 247  
[deprecate\(\) \(in module glotaran.deprecation.deprecation\\_utils\)](#), 195  
[deprecate\\_dict\\_entry\(\) \(in module glotaran.deprecation.deprecation\\_utils\)](#), 197  
[deprecate\\_module\\_attribute\(\) \(in module glotaran.deprecation.deprecation\\_utils\)](#), 199  
[deprecate\\_submodule\(\) \(in module glotaran.deprecation.deprecation\\_utils\)](#), 200  
[dimension \(glotaran.builtin.megacomplexes.baseline.baseline\\_megacomplex.BaselineMegacomplex property\)](#), 110  
[dimension \(glotaran.builtin.megacomplexes.coherent\\_artifact.coherent\\_artifact\\_megacomplex.CoherentArtifactMegacomplex property\)](#), 115  
[dimension \(glotaran.builtin.megacomplexes.damped\\_oscillation.damped\\_oscillation\\_megacomplex.DampedOscillationMegacomplex property\)](#), 122  
[dimension \(glotaran.builtin.megacomplexes.decay.decay\\_megacomplex.DecayMegacomplex property\)](#), 128  
[dimension \(glotaran.builtin.megacomplexes.spectral.spectral\\_megacomplex.SpectralMegacomplex property\)](#), 134  
[dispersion\\_center \(glotaran.builtin.megacomplexes.decay.irf.IrfSpectralMegacomplex property\)](#), 150  
[dispersion\\_center \(glotaran.builtin.megacomplexes.decay.irf.IrfSpectralMegacomplex property\)](#), 157  
[dispersion\\_center \(glotaran.testing.model\\_generators.SimpleModelGenerator attribute\)](#), 337  
[dispersion\\_coefficients \(glotaran.testing.model\\_generators.SimpleModelGenerator attribute\)](#), 337  
[display\\_file\(\) \(in module glotaran.utils.ipython\)](#), 348  
[eigen\(\) \(glotaran.builtin.megacomplexes.decay.k\\_matrix.KMatrix method\)](#), 163  
[elements\\_in\\_string\\_of\\_list \(glotaran.utils.regex.RegexPattern attribute\)](#), 361  
[empty\(\) \(glotaran.builtin.megacomplexes.decay.k\\_matrix.KMatrix class method\)](#), 163  
[encode\(\) \(glotaran.utils.ipython.MarkdownStr method\)](#), 357  
[encode\\_with\(\) \(glotaran.utils.ipython.MarkdownStr method\)](#), 357  
[ensure\\_oscillation\\_parameter\(\) \(glotaran.builtin.megacomplexes.damped\\_oscillation.damped\\_oscillation\\_megacomplex.DampedOscillationMegacomplex method\)](#), 122  
[ensure\\_unique\\_megacomplexes\(\) \(glotaran.model.dataset\\_model.DatasetModel method\)](#), 231  
[envvar\\_list\\_splitter \(glotaran.cli.commands.util.ValOrRangeOrList attribute\)](#), 192  
[EqualAreaPenalty \(class in glotaran.model.clp\\_penalties\)](#), 215  
[exclude\\_from\\_dict\\_field\(\) \(in module glotaran.project.dataclass\\_helpers\)](#), 315  
[exclude\\_from\\_normalize \(glotaran.builtin.megacomplexes.decay.initial\\_concentration.InitialConcentrationMegacomplex property\)](#), 131  
[expandtabs\(\) \(glotaran.utils.ipython.MarkdownStr method\)](#), 357  
[ExplicitFile \(class in glotaran.builtin.io.ascii.wavelength\\_time\\_explicit\\_file\)](#), 82  
[export\(\) \(in module glotaran.cli.commands.explore\)](#), 186  
[EXPR \(glotaran.parameter.parameter.Keys attribute\)](#), 268  
[expression \(glotaran.parameter.parameter.Parameter property\)](#), 269

## F

- `fail()` (`glotaran.cli.commands.util.ValOrRangeOrList` method), 192
- `fdel` (`glotaran.model.property.ModelProperty` attribute), 247
- `fget` (`glotaran.model.property.ModelProperty` attribute), 247
- `file_loadable_field()` (in module `glotaran.project.dataclass_helpers`), 315
- `file_loader_factory()` (in module `glotaran.project.dataclass_helpers`), 316
- `FileLoadableProtocol` (class in `glotaran.typing.protocols`), 342
- `fill()` (`glotaran.builtin.megacomplexes.baseline.baseline_megacomplex` method), 110
- `fill()` (`glotaran.builtin.megacomplexes.coherent_artifact.coherent_artifact_megacomplex` method), 115
- `fill()` (`glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation_megacomplex` method), 122
- `fill()` (`glotaran.builtin.megacomplexes.decay.decay_megacomplex` method), 128
- `fill()` (`glotaran.builtin.megacomplexes.decay.initial_concentration.InitialConcentration` method), 131
- `fill()` (`glotaran.builtin.megacomplexes.decay.irf.IrfGaussian` method), 137
- `fill()` (`glotaran.builtin.megacomplexes.decay.irf.IrfMeasured` method), 140
- `fill()` (`glotaran.builtin.megacomplexes.decay.irf.IrfMultiGaussian` method), 144
- `fill()` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralGaussian` method), 150
- `fill()` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralMultiGaussian` method), 157
- `fill()` (`glotaran.builtin.megacomplexes.decay.k_matrix.KMatrix` method), 163
- `fill()` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeGaussian` method), 171
- `fill()` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeOne` method), 173
- `fill()` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeSkewedGaussian` method), 177
- `fill()` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeZero` method), 180
- `fill()` (`glotaran.builtin.megacomplexes.spectral.spectral_megacomplex.SpectralMegaComplex` method), 184
- `fill()` (`glotaran.model.clp_penalties.EqualAreaPenalty` method), 218
- `fill()` (`glotaran.model.constraint.OnlyConstraint` method), 222
- `fill()` (`glotaran.model.constraint.ZeroConstraint` method), 224
- `fill()` (`glotaran.model.interval_property.IntervalProperty` method), 234
- `fill()` (`glotaran.model.relation.Relation` method), 252
- `fill()` (`glotaran.model.weight.Weight` method), 257
- `finalize_data()` (`glotaran.builtin.megacomplexes.baseline.baseline_megacomplex` method), 110
- `finalize_data()` (`glotaran.builtin.megacomplexes.coherent_artifact.coherent_artifact_megacomplex` method), 115
- `finalize_data()` (`glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation_megacomplex` method), 122
- `finalize_data()` (`glotaran.builtin.megacomplexes.decay.decay_megacomplex` method), 128
- `finalize_data()` (`glotaran.builtin.megacomplexes.spectral.spectral_megacomplex` method), 184
- `finalize_data()` (`glotaran.model.dataset_model.DatasetModel` method), 231
- `find()` (`glotaran.utils.ipython.MarkdownStr` method), 357
- `find_closest_index()` (in module `glotaran.analysis.util`), 76
- `find_overlap()` (in module `glotaran.analysis.util`), 76
- `FolderProjectIo` (class in `glotaran.builtin.io.folder.folder_plugin`), 95
- `format()` (`glotaran.utils.ipython.MarkdownStr` method), 357
- `format_map()` (`glotaran.utils.ipython.MarkdownStr` method), 357
- `free_parameter_labels` (`glotaran.project.result.Result` attribute), 324
- `frequencies` (`glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation_megacomplex` property), 122
- `from_csv()` (`glotaran.parameter.parameter_history.ParameterHistory` class method), 285
- `from_dataframe()` (`glotaran.parameter.parameter_group.ParameterGroup` class method), 277
- `from_dataframe()` (`glotaran.parameter.parameter_history.ParameterHistory` class method), 285
- `from_dict()` (`glotaran.builtin.megacomplexes.baseline.baseline_megacomplex` class method), 110
- `from_dict()` (`glotaran.builtin.megacomplexes.coherent_artifact.coherent_artifact_megacomplex` class method), 115
- `from_dict()` (`glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation_megacomplex` class method), 122
- `from_dict()` (`glotaran.builtin.megacomplexes.decay.decay_megacomplex` class method), 128
- `from_dict()` (`glotaran.builtin.megacomplexes.decay.initial_concentration.InitialConcentration` class method), 131
- `from_dict()` (`glotaran.builtin.megacomplexes.decay.irf.IrfGaussian` class method), 137
- `from_dict()` (`glotaran.builtin.megacomplexes.decay.irf.IrfMeasured` class method), 140
- `from_dict()` (`glotaran.builtin.megacomplexes.decay.irf.IrfMultiGaussian` class method), 144
- `from_dict()` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralGaussian` class method), 150
- `from_dict()` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralMultiGaussian` class method), 157



class method), 157  
 from\_dict() (glotaran.builtin.megacomplexes.decay.k\_matrix.KMatrix property), 58  
 class method), 163  
 from\_dict() (glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeGaussian plugin system.base\_registry), 289  
 class method), 171  
 from\_dict() (glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeOne class method), 173  
 from\_dict() (glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeSkewedGaussian class method), 177  
 from\_dict() (glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeZero class method), 180  
 from\_dict() (glotaran.builtin.megacomplexes.spectral.spectral\_megacomplex.SpectralMegacomplex class method), 185  
 from\_dict() (glotaran.model.clp\_penalties.EqualAreaPenalty class method), 218  
 from\_dict() (glotaran.model.constraint.OnlyConstraint class method), 222  
 from\_dict() (glotaran.model.constraint.ZeroConstraint class method), 224  
 from\_dict() (glotaran.model.interval\_property.IntervalProperty class method), 234  
 from\_dict() (glotaran.model.model.Model class method), 240  
 from\_dict() (glotaran.model.relation.Relation class method), 252  
 from\_dict() (glotaran.model.weight.Weight class method), 257  
 from\_dict() (glotaran.parameter.parameter.Parameter class method), 265  
 from\_dict() (glotaran.parameter.parameter\_group.ParameterGroup class method), 278  
 from\_list() (glotaran.parameter.parameter\_group.ParameterGroup class method), 278  
 from\_list\_or\_value() (glotaran.parameter.parameter.Parameter class method), 265  
 from\_parameter\_dict\_list() (glotaran.parameter.parameter\_group.ParameterGroup class method), 278  
 from\_yaml\_file() (glotaran.project.scheme.Scheme static method), 331  
 fromdict() (in module glotaran.project.dataclass\_helpers), 316  
 fromkeys() (glotaran.parameter.parameter\_group.ParameterGroup method), 278  
 fset (glotaran.model.property.ModelProperty attribute), 247  
 ftol (glotaran.project.scheme.Scheme attribute), 332  
 full() (glotaran.builtin.megacomplexes.decay.k\_matrix.KMatrix method), 163  
 full\_k\_matrix() (glotaran.builtin.megacomplexes.decay.decay\_megacomplex.DecayMegacomplex method), 128  
 full\_label (glotaran.parameter.parameter.Parameter property), 266  
 full\_penalty (glotaran.analysis.optimization\_group.OptimizationGroup property), 58  
 full\_plugin\_name() (in module glotaran.plugin\_system.base\_registry), 289  
**G**  
 get() (glotaran.parameter.parameter\_group.ParameterGroup method), 278  
 get() (glotaran.utils.io.DatasetMapping method), 347  
 get\_coordinates() (glotaran.model.dataset\_model.DatasetModel method), 231  
 get\_data() (glotaran.model.dataset\_model.DatasetModel method), 231  
 get\_data\_file\_format() (in module glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file), 80  
 get\_data\_io() (in module glotaran.plugin\_system.data\_io\_registration), 296  
 get\_data\_row() (glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.ExplicitFile method), 84  
 get\_data\_row() (glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.TimeExplicitFile method), 87  
 get\_data\_row() (glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.WavelengthTimeExplicitFile method), 90  
 get\_data\_loader() (in module glotaran.plugin\_system.data\_io\_registration), 296  
 get\_datasaver() (in module glotaran.plugin\_system.data\_io\_registration), 296  
 get\_dataset() (glotaran.project.result.Result method), 324  
 get\_dataset\_groups() (glotaran.model.model.Model method), 240  
 get\_default\_type() (glotaran.builtin.megacomplexes.decay.irf.Irf class method), 133  
 get\_default\_type() (glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeOne class method), 168  
 get\_default\_type() (glotaran.model.constraint.Constraint class method), 220  
 get\_explicit\_axis() (glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.ExplicitFile method), 84  
 get\_explicit\_axis() (glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.TimeExplicitFile method), 87  
 get\_explicit\_axis() (glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.WavelengthTimeExplicitFile method), 90  
 get\_format\_name() (glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.ExplicitFile method), 84  
 get\_format\_name() (glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.TimeExplicitFile method), 87

`get_format_name()` (*glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.WavelengthExplicitFile* method), 90  
`get_global_axis()` (*glotaran.model.dataset\_model.DatasetModel* method), 231  
`get_global_dimension()` (*glotaran.model.dataset\_model.DatasetModel* method), 231  
`get_group_for_parameter_by_label()` (*glotaran.parameter.parameter\_group.ParameterGroup* method), 279  
`get_idx_from_interval()` (in module *glotaran.analysis.util*), 76  
`get_interval_number()` (in module *glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.ExplicitFile*), 80  
`get_label_value_and_bounds_arrays()` (*glotaran.parameter.parameter\_group.ParameterGroup* method), 279  
`get_megacomplex()` (in module *glotaran.plugin\_system.megacomplex\_registration*), 302  
`get_metavar()` (*glotaran.cli.commands.util.ValOrRangeOrList* method), 192  
`get_method_from_plugin()` (in module *glotaran.plugin\_system.base\_registry*), 289  
`get_min_max_from_interval()` (in module *glotaran.analysis.util*), 76  
`get_missing_message()` (*glotaran.cli.commands.util.ValOrRangeOrList* method), 192  
`get_model_axis()` (*glotaran.model.dataset\_model.DatasetModel* method), 231  
`get_model_dimension()` (*glotaran.model.dataset\_model.DatasetModel* method), 231  
`get_nr_roots()` (*glotaran.parameter.parameter\_group.ParameterGroup* method), 279  
`get_observations()` (*glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.ExplicitFile* method), 84  
`get_observations()` (*glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.TimeExplicitFile* method), 87  
`get_observations()` (*glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.WavelengthExplicitFile* method), 90  
`get_parameters()` (*glotaran.model.model.Model* method), 240  
`get_parameters()` (*glotaran.parameter.parameter\_history.ParameterHistory* method), 285  
`get_plugin_from_registry()` (in module *glotaran.plugin\_system.base\_registry*), 290  
`get_project_io()` (in module *glotaran.plugin\_system.project\_io\_registration*), 305  
`get_project_io_method()` (in module *glotaran.plugin\_system.project\_io\_registration*), 306  
`get_scheme()` (*glotaran.project.result.Result* method), 324  
`get_secondary_axis()` (*glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.ExplicitFile* method), 84  
`get_secondary_axis()` (*glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.TimeExplicitFile* method), 87  
`get_secondary_axis()` (*glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.WavelengthExplicitFile* method), 90  
`get_subtype()` (in module *glotaran.model.util*), 253  
`get_value_and_bounds_for_optimization()` (*glotaran.parameter.parameter.Parameter* method), 266  
`get_weight()` (*glotaran.model.dataset\_model.DatasetModel* method), 231  
`getter()` (*glotaran.model.property.ModelProperty* method), 247  
`global_dimension` (*glotaran.model.model.Model* property), 241  
`global_dimensions` (*glotaran.project.scheme.Scheme* property), 332  
`global_interval` (*glotaran.model.weight.Weight* property), 257  
`global_matrices` (*glotaran.analysis.optimization\_group\_calculator\_unlinked* property), 72  
`global_megacomplex` (*glotaran.model.model.Model* property), 241  
`glotaran` module, 53  
`glotaran` command line option `--version`, 41  
`glotaran.analysis` module, 54  
`glotaran.analysis.nnls` module, 54  
`glotaran.analysis.explicit_file.ExplicitFile` module, 54  
`glotaran.analysis.optimization_group` module, 54  
`glotaran.analysis.optimization_group_calculator` module, 54  
`glotaran.analysis.optimization_group_calculator_linked` module, 61  
`glotaran.analysis.optimization_group_calculator_unlinked` module, 61  
`glotaran.analysis.optimize` module, 72  
`glotaran.analysis.simulation` module, 73  
`glotaran.analysis.util` module, 74  
`glotaran.analysis.variable_projection` module, 78

glotaran.builtin	glotaran.builtin.megacomplexes.spectral
module, 79	module, 167
glotaran.builtin.io	glotaran.builtin.megacomplexes.spectral.shape
module, 79	module, 167
glotaran.builtin.io.ascii	glotaran.builtin.megacomplexes.spectral.spectral_megacompl
module, 79	module, 181
glotaran.builtin.io.ascii.wavelength_time_explicit_cli	glotaran.builtin.megacomplexes.spectral.spectral_megacompl
module, 79	module, 185
glotaran.builtin.io.csv	glotaran.cli.commands
module, 91	module, 186
glotaran.builtin.io.csv.csv	glotaran.cli.commands.explore
module, 91	module, 186
glotaran.builtin.io.folder	glotaran.cli.commands.export
module, 94	module, 186
glotaran.builtin.io.folder.folder_plugin	glotaran.cli.commands.optimize
module, 94	module, 186
glotaran.builtin.io.netCDF	glotaran.cli.commands.pluginlist
module, 99	module, 187
glotaran.builtin.io.netCDF.netCDF	glotaran.cli.commands.print
module, 99	module, 187
glotaran.builtin.io.sdt	glotaran.cli.commands.util
module, 100	module, 187
glotaran.builtin.io.sdt.sdt_file_reader	glotaran.cli.commands.validate
module, 100	module, 193
glotaran.builtin.io.yml	glotaran.deprecation
module, 102	module, 194
glotaran.builtin.io.yml.yml	glotaran.deprecation.deprecation_utils
module, 102	module, 194
glotaran.builtin.megacomplexes	glotaran.deprecation.modules
module, 106	module, 204
glotaran.builtin.megacomplexes.baseline	glotaran.deprecation.modules.builtin_io_yaml
module, 106	module, 205
glotaran.builtin.megacomplexes.baseline.baseline_megacomplex	glotaran.deprecation.modules.glotaran_root
module, 106	module, 205
glotaran.builtin.megacomplexes.coherent_artifact	glotaran.examples
module, 111	module, 207
glotaran.builtin.megacomplexes.coherent_artifact.coherent_artifact_fast_megacomplex	glotaran.io
module, 111	module, 208
glotaran.builtin.megacomplexes.damped_oscillation	glotaran.io.prepare_dataset
module, 116	module, 213
glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation_megacomplex	glotaran.io.prepare_dataset
module, 116	module, 213
glotaran.builtin.megacomplexes.decay	glotaran.model
module, 123	module, 214
glotaran.builtin.megacomplexes.decay.decay_megacomplex	glotaran.model.clp_penalties
module, 123	module, 214
glotaran.builtin.megacomplexes.decay.initial_constraint	glotaran.model.constraint
module, 129	module, 219
glotaran.builtin.megacomplexes.decay.irf	glotaran.model.dataset_group
module, 132	module, 225
glotaran.builtin.megacomplexes.decay.k_matrix	glotaran.model.dataset_model
module, 158	module, 227
glotaran.builtin.megacomplexes.decay.util	
module, 164	

glotaran.model.interval\_property  
    module, 232

glotaran.model.item  
    module, 235

glotaran.model.model  
    module, 236

glotaran.model.property  
    module, 242

glotaran.model.relation  
    module, 249

glotaran.model.util  
    module, 253

glotaran.model.weight  
    module, 255

glotaran.parameter  
    module, 258

glotaran.parameter.parameter  
    module, 258

glotaran.parameter.parameter\_group  
    module, 268

glotaran.parameter.parameter\_history  
    module, 282

glotaran.plugin\_system  
    module, 286

glotaran.plugin\_system.base\_registry  
    module, 287

glotaran.plugin\_system.data\_io\_registration  
    module, 295

glotaran.plugin\_system.io\_plugin\_utils  
    module, 299

glotaran.plugin\_system.megacomplex\_registration  
    module, 302

glotaran.plugin\_system.project\_io\_registration  
    module, 304

glotaran.project  
    module, 314

glotaran.project.dataclass\_helpers  
    module, 314

glotaran.project.result  
    module, 317

glotaran.project.scheme  
    module, 326

glotaran.testing  
    module, 333

glotaran.testing.model\_generators  
    module, 333

glotaran.testing.plugin\_system  
    module, 339

glotaran.typing  
    module, 341

glotaran.typing.protocols  
    module, 342

glotaran.typing.types  
    module, 343

glotaran.utils  
    module, 343

glotaran.utils.io  
    module, 343

glotaran.utils.ipython  
    module, 348

glotaran.utils.regex  
    module, 359

glotaran.utils.sanitize  
    module, 361

glotaran\_allow\_none  
    (*glotaran.model.property.ModelProperty*  
        *property*), 247

glotaran\_dataset\_model\_items()  
    (*glotaran.builtin.megacomplexes.baseline.baseline\_megacomplex*  
        *class method*), 110

glotaran\_dataset\_model\_items()  
    (*glotaran.builtin.megacomplexes.coherent\_artifact.coherent\_artif*  
        *class method*), 115

glotaran\_dataset\_model\_items()  
    (*glotaran.builtin.megacomplexes.damped\_oscillation.damped\_osc*  
        *class method*), 122

glotaran\_dataset\_model\_items()  
    (*glotaran.builtin.megacomplexes.decay.decay\_megacomplex.Decay*  
        *class method*), 128

glotaran\_dataset\_model\_items()  
    (*glotaran.builtin.megacomplexes.spectral.spectral\_megacomplex*  
        *class method*), 185

glotaran\_dataset\_properties()  
    (*glotaran.builtin.megacomplexes.baseline.baseline\_megacomplex*  
        *class method*), 110

glotaran\_dataset\_properties()  
    (*glotaran.builtin.megacomplexes.coherent\_artifact.coherent\_artif*  
        *class method*), 115

glotaran\_dataset\_properties()  
    (*glotaran.builtin.megacomplexes.damped\_oscillation.damped\_osc*  
        *class method*), 122

glotaran\_dataset\_properties()  
    (*glotaran.builtin.megacomplexes.decay.decay\_megacomplex.Decay*  
        *class method*), 128

glotaran\_dataset\_properties()  
    (*glotaran.builtin.megacomplexes.spectral.spectral\_megacomplex*  
        *class method*), 185

glotaran\_fill() (*glotaran.model.property.ModelProperty*  
    *method*), 247

glotaran\_format\_value()  
    (*glotaran.model.property.ModelProperty*  
        *method*), 248

glotaran\_is\_mapping\_property  
    (*glotaran.model.property.ModelProperty*  
        *property*), 248

glotaran\_is\_parameter\_property  
    (*glotaran.model.property.ModelProperty*  
        *property*), 248



```

glotaran_is_scalar_property
    (glotaran.model.property.ModelProperty
     property), 248
glotaran_is_sequence_property
    (glotaran.model.property.ModelProperty
     property), 248
glotaran_model_items()
    (glotaran.builtin.megacomplexes.baseline.baseline_mega-
     complex.BaselineMegaComplex
     class method), 110
glotaran_model_items()
    (glotaran.builtin.megacomplexes.coherent_artifact.coher-
     ent_artifact_megacomplex.CoherentArtifactMegaComplex
     class method), 115
glotaran_model_items()
    (glotaran.builtin.megacomplexes.damped_oscillation.damp-
     ed_oscillation_megacomplex.DampedOscillationMegaComplex
     class method), 122
glotaran_model_items()
    (glotaran.builtin.megacomplexes.decay.decay_mega-complex.
     DecayMegaComplex class method), 128
glotaran_model_items()
    (glotaran.builtin.megacomplexes.spectral.spectral_mega-
     complex.SpectralMegaComplex
     class method), 185
glotaran_property_subtype
    (glotaran.model.property.ModelProperty
     property), 248
glotaran_property_type
    (glotaran.model.property.ModelProperty
     property), 248
glotaran_replace_parameter_with_labels()
    (glotaran.model.property.ModelProperty
     method), 248
glotaran_unique() (glotaran.builtin.megacomplexes.baseli-
     ne_megacomplex.BaselineMegaComplex
     class method), 110
glotaran_unique() (glotaran.builtin.megacomplexes.coher-
     ent_artifact_megacomplex.CoherentArtifactMegaComplex
     class method), 115
glotaran_unique() (glotaran.builtin.megacomplexes.damp-
     ed_oscillation_megacomplex.DampedOscillationMegaCom-
     plex class method), 122
glotaran_unique() (glotaran.builtin.megacomplexes.decay.
     decay_megacomplex.DecayMegaComplex
     class method), 128
glotaran_unique() (glotaran.builtin.megacomplexes.spectra-
     l_megacomplex.SpectralMegaComplex
     class method), 185
glotaran_validate()
    (glotaran.model.property.ModelProperty
     method), 249
glotaran_value_as_markdown()
    (glotaran.model.property.ModelProperty
     method), 249
glotaran_version (glotaran.project.result.Result at-
     tribute), 325
glotaran_version() (in module
    glotaran.deprecation.deprecation_utils),
    201
glotaran-optimize command line option
    --data <data>, 41
    --dataformat <dataformat>, 41
--model_file <model_file>, 42
--nfev <nfev>, 41
--nnls, 41
--out <out>, 41
--outformat <outformat>, 41
--parameters_file <parameters_file>, 42
--yes, 42
dfmt, 41
-m, 42
-o, 41
-ofmt, 41
-p, 42
-y, 42
SCHEME_FILE, 42
glotaran-print command line option
--model_file <model_file>, 42
--parameters_file <parameters_file>, 42
-m, 42
-p, 42
SCHEME_FILE, 42
glotaran-validate command line option
--model_file <model_file>, 43
--parameters_file <parameters_file>, 43
-m, 43
-p, 43
SCHEME_FILE, 43
group (glotaran.analysis.optimization_group_calculator_linked.DatasetInd-
    ex attribute), 66
group (glotaran.builtin.megacomplexes.baseline_mega-complex.Baseli-
    neMegaComplex attribute), 332
group (glotaran.builtin.megacomplexes.coherent_artifact_mega-complex.Coher-
    entArtifactMegaComplex attribute), 332
group (glotaran.builtin.megacomplexes.damped_oscillation_mega-complex.Damp-
    edOscillationMegaComplex attribute), 69
group (glotaran.builtin.megacomplexes.decay_mega-complex.DecayMegaCom-
    plex attribute), 279
group (glotaran.builtin.megacomplexes.spectral_mega-complex.SpectralMega-
    Complex attribute), 332
H
has() (glotaran.parameter.parameter_group.ParameterGroup
    method), 279
has_global_model() (glotaran.model.dataset_model.DatasetModel
    method), 231
has_k_matrix() (glotaran.builtin.megacomplexes.decay.decay_mega-complex.
    DecayMegaComplex method), 128
has_scaling (glotaran.analysis.optimization_group_calculator_linked.Data-
    setIndex attribute), 66
has_spectral_penalties() (in module
    glotaran.model.clp_penalties), 215
I
index() (glotaran.analysis.optimization_group_calculator_linked.DatasetIn-

```

method), 63

`index()` (glotaran.analysis.optimization\_group\_calculator\_linked.DatasetIndexModelGenerator), 133

method), 66

`index()` (glotaran.analysis.util.CalculatedMatrix), 78

method), 78

`index()` (glotaran.utils.ipython.MarkdownStr method), 357

`index_dependent()` (glotaran.builtin.megacomplexes.baseline.baseline\_megacomplex.BaselineMegacomplex), 140

method), 110

`index_dependent()` (glotaran.builtin.megacomplexes.coherent\_artifact\_megacomplex.CoherentArtifactMegacomplex), 145

method), 115

`index_dependent()` (glotaran.builtin.megacomplexes.damped\_oscillation\_megacomplex.DampedOscillationMegacomplex), 122

method), 122

`index_dependent()` (glotaran.builtin.megacomplexes.decay.decay\_megacomplex.DecayMegacomplex), 151

method), 128

`index_dependent()` (glotaran.builtin.megacomplexes.spectral\_multi\_gaussian\_megacomplex.SpectralMultiGaussianMegacomplex), 192

method), 185

`indices` (glotaran.analysis.optimization\_group\_calculator\_linked.DatasetIndexModel (glotaran.model.model.Model attribute), 63

method), 241

`infer_file_format()` (in module `is_index_dependent()` (glotaran.builtin.megacomplexes.decay.irf.IrfGaussian method), 137

glotaran.plugin\_system.io\_plugin\_utils), 301

`init_bag()` (glotaran.analysis.optimization\_group\_calculator\_linked.DatasetIndexModelGenerator), 69

method), 69

`init_file_loadable_fields()` (in module `is_index_dependent()` (glotaran.builtin.megacomplexes.decay.irf.IrfMultiGaussian method), 144

glotaran.project.dataclass\_helpers), 317

`is_index_dependent()` (glotaran.builtin.megacomplexes.decay.irf.IrfSpectralGaussian method), 150

`initial_concentration` (glotaran.testing.model\_generators.SimpleModelGenerator method), 150

attribute), 337

`is_index_dependent()` (glotaran.builtin.megacomplexes.decay.irf.IrfSpectralMultiGaussian method), 157

`initial_parameters` (glotaran.project.result.Result attribute), 325

`InitialConcentration` (class in `is_index_dependent()` (glotaran.builtin.megacomplexes.decay.initial\_concentration) glotaran.model.dataset\_model.DatasetModel method), 231

129

`interval` (glotaran.model.constraint.OnlyConstraint property), 222

`is_known_data_format()` (in module `glotaran.plugin_system.data_io_registration`), 296

`interval` (glotaran.model.constraint.ZeroConstraint property), 224

`is_known_megacomplex()` (in module `glotaran.plugin_system.megacomplex_registration`), 303

`interval` (glotaran.model.interval\_property.IntervalProperty property), 234

`is_known_project_format()` (in module `glotaran.plugin_system.project_io_registration`), 306

`IntervalProperty` (class in `is_mapping_type()` (in module `glotaran.model.util`), 254

glotaran.model.interval\_property), 232

`involved_compartments` (glotaran.builtin.megacomplexes.decay.decay\_megacomplex.DecayMegacomplex (in module `glotaran.plugin_system.base_registry`), 290

property), 128

`involved_compartments()` (in module `glotaran.model.util`), 254

`is_scalar_type()` (in module `glotaran.model.util`), 254

(glotaran.builtin.megacomplexes.decay.k\_matrix.KMSequence method), 163

`is_sequence_type()` (in module `glotaran.model.util`), 254

`Irf` (class in `glotaran.builtin.megacomplexes.decay.irf`), 132

`is_unbranched()` (glotaran.builtin.megacomplexes.decay.k\_matrix.KMSequence method), 163

`irf` (glotaran.testing.model\_generators.SimpleModelGenerator attribute), 337

`isalnum()` (glotaran.utils.ipython.MarkdownStr method), 357

`isalpha()` (`glotaran.utils.ipython.MarkdownStr` method), 357  
`isascii()` (`glotaran.utils.ipython.MarkdownStr` method), 357  
`isdecimal()` (`glotaran.utils.ipython.MarkdownStr` method), 357  
`isdigit()` (`glotaran.utils.ipython.MarkdownStr` method), 357  
`isidentifier()` (`glotaran.utils.ipython.MarkdownStr` method), 357  
`islower()` (`glotaran.utils.ipython.MarkdownStr` method), 357  
`isnumeric()` (`glotaran.utils.ipython.MarkdownStr` method), 357  
`isprintable()` (`glotaran.utils.ipython.MarkdownStr` method), 358  
`isspace()` (`glotaran.utils.ipython.MarkdownStr` method), 358  
`istitle()` (`glotaran.utils.ipython.MarkdownStr` method), 358  
`isupper()` (`glotaran.utils.ipython.MarkdownStr` method), 358  
`items()` (`glotaran.parameter.parameter_group.ParameterGroup` method), 279  
`items()` (`glotaran.utils.io.DatasetMapping` method), 347  
`iterate_global_megacomplexes()` (`glotaran.model.dataset_model.DatasetModel` method), 231  
`iterate_megacomplexes()` (`glotaran.model.dataset_model.DatasetModel` method), 231

## J

`jacobian` (`glotaran.project.result.Result` attribute), 325  
`join()` (`glotaran.utils.ipython.MarkdownStr` method), 358

## K

`k_matrix` (`glotaran.builtin.megacomplexes.decay.decay_megacomplex` property), 128  
`k_matrix` (`glotaran.testing.model_generators.SimpleModelGenerator` attribute), 337  
`Keys` (class in `glotaran.parameter.parameter`), 258  
`keys()` (`glotaran.parameter.parameter_group.ParameterGroup` method), 280  
`keys()` (`glotaran.utils.io.DatasetMapping` method), 347  
`KMatrix` (class in `glotaran.builtin.megacomplexes.decay.k_matrix`), 158  
`known_data_formats()` (in module `glotaran.plugin_system.data_io_registration`), 297  
`known_megacomplex_names()` (in module `glotaran.plugin_system.megacomplex_registration`), 303  
`known_project_formats()` (in module `glotaran.plugin_system.project_io_registration`), 306

## L

`label` (`glotaran.analysis.optimization_group_calculator_linked.DatasetIndex` attribute), 63  
`label` (`glotaran.builtin.megacomplexes.baseline.baseline_megacomplex.Baseline` property), 110  
`label` (`glotaran.builtin.megacomplexes.coherent_artifact.coherent_artifact.CoherentArtifact` property), 115  
`label` (`glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation.DampedOscillation` property), 122  
`label` (`glotaran.builtin.megacomplexes.decay.decay_megacomplex.DecayMegaComplex` property), 128  
`label` (`glotaran.builtin.megacomplexes.decay.initial_concentration.InitialConcentration` property), 132  
`label` (`glotaran.builtin.megacomplexes.decay.irf.IrfGaussian` property), 137  
`label` (`glotaran.builtin.megacomplexes.decay.irf.IrfMeasured` property), 140  
`label` (`glotaran.builtin.megacomplexes.decay.irf.IrfMultiGaussian` property), 145  
`label` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralGaussian` property), 150  
`label` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralMultiGaussian` property), 157  
`label` (`glotaran.builtin.megacomplexes.decay.k_matrix.KMatrix` property), 163  
`label` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeGaussian` property), 171  
`label` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeOne` property), 174  
`label` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeSkewed` property), 177  
`label` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeZero` property), 180  
`label` (`glotaran.builtin.megacomplexes.spectral.spectral_megacomplex.SpectralMegaComplex` property), 185  
`label` (`glotaran.parameter.parameter.Parameter` property), 266  
`label` (`glotaran.parameter.parameter_group.ParameterGroup` property), 280  
`labels` (`glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation.DampedOscillation` property), 122  
`link_clp` (`glotaran.model.dataset_group.DatasetGroupModel` attribute), 227  
`list_string_to_tuple()` (in module `glotaran.utils.sanitize`), 362  
`list_with_tuples` (`glotaran.utils.regex.RegexPattern` attribute), 361  
`ljust()` (`glotaran.utils.ipython.MarkdownStr` method), 358

`load_dataset()` (`glotaran.builtin.io.ascii.wavelength_time_explicit_file.AsciiDataIo` method), 81  
`load_dataset()` (`glotaran.builtin.io.netCDF.netCDF.NetCDFDataIo` method), 98  
`load_dataset()` (`glotaran.builtin.io.sdt.sdt_file_reader.SdtDataIo` method), 105  
`load_dataset()` (`glotaran.io.interface.DataIoInterface` method), 209  
`load_dataset()` (in module `glotaran.plugin_system.data_io_registration`), 297  
`load_dataset_file()` (in module `glotaran.cli.commands.util`), 188  
`load_datasets()` (in module `glotaran.utils.io`), 344  
`load_model()` (`glotaran.builtin.io.csv.csv.CsvProjectIo` method), 93  
`load_model()` (`glotaran.builtin.io.folder.folder_plugin.FolderProjectIo` method), 97  
`load_model()` (`glotaran.builtin.io.yml.yml.YmlProjectIo` method), 105  
`load_model()` (`glotaran.io.interface.ProjectIoInterface` method), 212  
`load_model()` (in module `glotaran.plugin_system.project_io_registration`), 307  
`load_model_file()` (in module `glotaran.cli.commands.util`), 188  
`load_parameter_file()` (in module `glotaran.cli.commands.util`), 188  
`load_parameters()` (`glotaran.builtin.io.csv.csv.CsvProjectIo` method), 93  
`load_parameters()` (`glotaran.builtin.io.folder.folder_plugin.FolderProjectIo` method), 97  
`load_parameters()` (`glotaran.builtin.io.yml.yml.YmlProjectIo` method), 105  
`load_parameters()` (`glotaran.io.interface.ProjectIoInterface` method), 212  
`load_parameters()` (in module `glotaran.plugin_system.project_io_registration`), 307  
`load_plugins()` (in module `glotaran.plugin_system.base_registry`), 291  
`load_result()` (`glotaran.builtin.io.csv.csv.CsvProjectIo` method), 93  
`load_result()` (`glotaran.builtin.io.folder.folder_plugin.FolderProjectIo` method), 97  
`load_result()` (`glotaran.builtin.io.yml.yml.YmlProjectIo` method), 105  
`load_result()` (`glotaran.io.interface.ProjectIoInterface` method), 212  
`load_result()` (in module `glotaran.plugin_system.project_io_registration`), 307  
`load_scheme()` (`glotaran.builtin.io.csv.csv.CsvProjectIo` method), 81  
`load_scheme()` (`glotaran.builtin.io.folder.folder_plugin.FolderProjectIo` method), 98  
`load_scheme()` (`glotaran.builtin.io.yml.yml.YmlProjectIo` method), 105  
`load_scheme()` (`glotaran.io.interface.ProjectIoInterface` method), 212  
`load_scheme()` (in module `glotaran.plugin_system.project_io_registration`), 308  
`load_scheme_file()` (in module `glotaran.cli.commands.util`), 188  
`loader` (`glotaran.typing.protocols.FileLoadableProtocol` attribute), 343  
`loader()` (`glotaran.model.model.Model` method), 241  
`loader()` (`glotaran.parameter.parameter_group.ParameterGroup` method), 280  
`loader()` (`glotaran.parameter.parameter_history.ParameterHistory` class method), 285  
`loader()` (`glotaran.project.result.Result` method), 325  
`loader()` (`glotaran.project.scheme.Scheme` method), 332  
`loader()` (`glotaran.utils.io.DatasetMapping` class method), 347  
`location` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeG` property), 171  
`location` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeS` property), 178  
`lower()` (`glotaran.utils.ipython.MarkdownStr` method), 358  
`lstrip()` (`glotaran.utils.ipython.MarkdownStr` method), 358

## M

`main` (in module `glotaran.cli`), 194  
`maketrans()` (`glotaran.utils.ipython.MarkdownStr` method), 358  
`markdown()` (`glotaran.builtin.megacomplexes.baseline.baseline_megacomplex` method), 110  
`markdown()` (`glotaran.builtin.megacomplexes.coherent_artifact.coherent_a` method), 115  
`markdown()` (`glotaran.builtin.megacomplexes.damped_oscillation.damped` method), 122  
`markdown()` (`glotaran.builtin.megacomplexes.decay.decay_megacomplex.L` method), 128  
`markdown()` (`glotaran.builtin.megacomplexes.decay.initial_concentration.L` method), 132  
`markdown()` (`glotaran.builtin.megacomplexes.decay.irf.IrfGaussian` method), 137  
`markdown()` (`glotaran.builtin.megacomplexes.decay.irf.IrfMeasured` method), 140  
`markdown()` (`glotaran.builtin.megacomplexes.decay.irf.IrfMultiGaussian` method), 145  
`markdown()` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralGaussian` method), 151





[59](#)  
[glotaran.analysis.optimization\\_group\\_calculator\\_linked,](#)  
[61](#)  
[glotaran.analysis.optimization\\_group\\_calculator\\_linked,](#)  
[70](#)  
[glotaran.analysis.optimize,](#) [72](#)  
[glotaran.analysis.simulation,](#) [73](#)  
[glotaran.analysis.util,](#) [74](#)  
[glotaran.analysis.variable\\_projection,](#) [78](#)  
[glotaran.builtin,](#) [79](#)  
[glotaran.builtin.io,](#) [79](#)  
[glotaran.builtin.io.ascii,](#) [79](#)  
[glotaran.builtin.io.ascii.wavelength\\_time\\_explicit,](#) [79](#)  
[glotaran.builtin.io.csv,](#) [91](#)  
[glotaran.builtin.io.csv.csv,](#) [91](#)  
[glotaran.builtin.io.folder,](#) [94](#)  
[glotaran.builtin.io.folder.folder\\_plugin,](#)  
[94](#)  
[glotaran.builtin.io.netCDF,](#) [99](#)  
[glotaran.builtin.io.netCDF.netCDF,](#) [99](#)  
[glotaran.builtin.io.sdt,](#) [100](#)  
[glotaran.builtin.io.sdt.sdt\\_file\\_reader,](#)  
[100](#)  
[glotaran.builtin.io.yml,](#) [102](#)  
[glotaran.builtin.io.yml.yml,](#) [102](#)  
[glotaran.builtin.megacomplexes,](#) [106](#)  
[glotaran.builtin.megacomplexes.baseline,](#)  
[106](#)  
[glotaran.builtin.megacomplexes.baseline.baseline\\_megacomplex,](#)  
[106](#)  
[glotaran.builtin.megacomplexes.coherent\\_artifact,](#)  
[111](#)  
[glotaran.builtin.megacomplexes.coherent\\_artifact.coherent\\_artifact\\_megacomplex,](#)  
[111](#)  
[glotaran.builtin.megacomplexes.damped\\_oscillation,](#)  
[116](#)  
[glotaran.builtin.megacomplexes.damped\\_oscillation.damped\\_oscillation\\_megacomplex,](#)  
[116](#)  
[glotaran.builtin.megacomplexes.decay,](#) [123](#)  
[glotaran.builtin.megacomplexes.decay.decay\\_megacomplex,](#)  
[123](#)  
[glotaran.builtin.megacomplexes.decay.initial\\_concentration,](#)  
[129](#)  
[glotaran.builtin.megacomplexes.decay.irf,](#)  
[132](#)  
[glotaran.builtin.megacomplexes.decay.k\\_matrix,](#)  
[158](#)  
[glotaran.builtin.megacomplexes.decay.util,](#)  
[164](#)  
[glotaran.builtin.megacomplexes.spectral,](#)  
[167](#)  
[glotaran.builtin.megacomplexes.spectral.shape,](#)  
[167](#)

[glotaran.builtin.megacomplexes.spectral.spectral\\_megacomplex,](#)  
[167](#)  
[glotaran.cli,](#) [185](#)  
[glotaran.cli.commands,](#) [186](#)  
[glotaran.cli.commands.explore,](#) [186](#)  
[glotaran.cli.commands.export,](#) [186](#)  
[glotaran.cli.commands.optimize,](#) [186](#)  
[glotaran.cli.commands.pluginlist,](#) [187](#)  
[glotaran.cli.commands.print,](#) [187](#)  
[glotaran.cli.commands.util,](#) [187](#)  
[glotaran.cli.commands.validate,](#) [193](#)  
[glotaran.deprecation,](#) [194](#)  
[glotaran.deprecation.deprecation\\_utils,](#)  
[194](#)  
[glotaran.deprecation.modules,](#) [204](#)  
[glotaran.deprecation.modules.builtin\\_io\\_yaml,](#)  
[205](#)  
[glotaran.deprecation.modules.glotaran\\_root,](#)  
[205](#)  
[glotaran.examples,](#) [207](#)  
[glotaran.examples.sequential,](#) [208](#)  
[glotaran.io,](#) [208](#)  
[glotaran.io.interface,](#) [208](#)  
[glotaran.io.prepare\\_dataset,](#) [213](#)  
[glotaran.model,](#) [214](#)  
[glotaran.model.clp\\_penalties,](#) [214](#)  
[glotaran.model.constraint,](#) [219](#)  
[glotaran.model.dataset\\_group,](#) [225](#)  
[glotaran.model.dataset\\_model,](#) [227](#)  
[glotaran.model.interval\\_property,](#) [232](#)  
[glotaran.model.item,](#) [235](#)  
[glotaran.model.model,](#) [236](#)  
[glotaran.model.property,](#) [242](#)  
[glotaran.model.refraction\\_megacomplex,](#)  
[249](#)  
[glotaran.model.util,](#) [253](#)  
[glotaran.model.weight,](#) [255](#)  
[glotaran.parameter,](#) [258](#)  
[glotaran.parameter.parameter\\_group,](#) [268](#)  
[glotaran.parameter.parameter\\_history,](#) [282](#)  
[glotaran.plugin\\_system,](#) [286](#)  
[glotaran.plugin\\_system.base\\_registry,](#) [287](#)  
[glotaran.plugin\\_system.data\\_io\\_registration,](#)  
[295](#)  
[glotaran.plugin\\_system.io\\_plugin\\_utils,](#)  
[299](#)  
[glotaran.plugin\\_system.megacomplex\\_registration,](#)  
[302](#)  
[glotaran.plugin\\_system.project\\_io\\_registration,](#)  
[304](#)  
[glotaran.project,](#) [314](#)  
[glotaran.project.dataclass\\_helpers,](#) [314](#)  
[glotaran.project.result,](#) [317](#)  
[glotaran.project.scheme,](#) [326](#)

[glotaran.testing](#), 333  
[glotaran.testing.model\\_generators](#), 333  
[glotaran.testing.plugin\\_system](#), 339  
[glotaran.typing](#), 341  
[glotaran.typing.protocols](#), 342  
[glotaran.typing.types](#), 343  
[glotaran.utils](#), 343  
[glotaran.utils.io](#), 343  
[glotaran.utils.ipython](#), 348  
[glotaran.utils.regex](#), 359  
[glotaran.utils.sanitize](#), 361  
[module\\_attribute\(\)](#) (in [module](#) [glotaran.deprecation.deprecation\\_utils](#)), 201  
[monkeypatch\\_plugin\\_registry\(\)](#) (in [module](#) [glotaran.testing.plugin\\_system](#)), 339  
[monkeypatch\\_plugin\\_registry\\_data\\_io\(\)](#) (in [module](#) [glotaran.testing.plugin\\_system](#)), 340  
[monkeypatch\\_plugin\\_registry\\_megacomplex\(\)](#) (in [module](#) [glotaran.testing.plugin\\_system](#)), 340  
[monkeypatch\\_plugin\\_registry\\_project\\_io\(\)](#) (in [module](#) [glotaran.testing.plugin\\_system](#)), 341

## N

[name](#) ([glotaran.builtin.megacomplexes.baseline.baseline\\_megacomplex](#) attribute), 110  
[name](#) ([glotaran.builtin.megacomplexes.coherent\\_artifact.coherent\\_artifact\\_megacomplex](#) attribute), 116  
[name](#) ([glotaran.builtin.megacomplexes.damped\\_oscillation.damped\\_oscillation\\_megacomplex](#) attribute), 123  
[name](#) ([glotaran.builtin.megacomplexes.decay.decay\\_megacomplex](#) attribute), 128  
[name](#) ([glotaran.builtin.megacomplexes.spectral.spectral\\_megacomplex](#) attribute), 185  
[name](#) ([glotaran.cli.commands.util.ValOrRangeOrList](#) attribute), 192  
[need\\_index\\_dependent\(\)](#) ([glotaran.model.model.Model](#) method), 241  
[NetCDFDataIo](#) (class in [glotaran.builtin.io.netCDF.netCDF](#)), 99  
[NON\\_NEG](#) ([glotaran.parameter.parameter.Keys](#) attribute), 260  
[non\\_negative](#) ([glotaran.parameter.parameter.Parameter](#) property), 266  
[non\\_negative\\_least\\_squares](#) ([glotaran.project.scheme.Scheme](#) attribute), 332  
[normalize](#) ([glotaran.builtin.megacomplexes.decay.irf.IrfGaussian](#) property), 137  
[normalize](#) ([glotaran.builtin.megacomplexes.decay.irf.IrfMorse](#) property), 145  
[normalize](#) ([glotaran.builtin.megacomplexes.decay.irf.IrfSpectralGaussian](#) property), 151

[normalize](#) ([glotaran.builtin.megacomplexes.decay.irf.IrfSpectralMultiGaussian](#) property), 157  
[normalized\(\)](#) ([glotaran.builtin.megacomplexes.decay.initial\\_concentration](#) method), 132  
[not\\_implemented\\_to\\_value\\_error\(\)](#) (in [module](#) [glotaran.plugin\\_system.io\\_plugin\\_utils](#)), 301  
[number](#) ([glotaran.utils.regex.RegexPattern](#) attribute), 361  
[number\\_of\\_data\\_points](#) ([glotaran.project.result.Result](#) attribute), 325  
[number\\_of\\_function\\_evaluations](#) ([glotaran.project.result.Result](#) attribute), 325  
[number\\_of\\_jacobian\\_evaluations](#) ([glotaran.project.result.Result](#) attribute), 325  
[number\\_of\\_records](#) ([glotaran.parameter.parameter\\_history.ParameterHistory](#) property), 286  
[number\\_of\\_variables](#) ([glotaran.project.result.Result](#) attribute), 325  
[number\\_scientific](#) ([glotaran.utils.regex.RegexPattern](#) attribute), 361

## O

[OnlyFromBuiltinMegacomplex](#) ([glotaran.model.constraint](#)), 220  
[optimize](#) ([glotaran.analysis.optimize](#)), 72  
[optimize\\_cmd\(\)](#) (in [module](#) [glotaran.cli.commands.optimize](#)), 187  
[optimized\\_parameters](#) ([glotaran.project.result.Result](#) attribute), 325  
[order](#) ([glotaran.builtin.megacomplexes.coherent\\_artifact.coherent\\_artifact\\_megacomplex](#) property), 116  
[overwrite\\_index\\_dependent\(\)](#) ([glotaran.model.dataset\\_model.DatasetModel](#) method), 231

method), 232  
 overwrite\_model\_dimension()  
 (glotaran.model.dataset\_model.DatasetModel  
 method), 232

## P

Parameter (class in glotaran.parameter.parameter), 260  
 parameter (glotaran.model.clp\_penalties.EqualAreaPenalty  
 property), 218  
 parameter (glotaran.model.relation.Relation property),  
 252  
 parameter() (glotaran.builtin.megacomplexes.decay.irf.IrfGaussian  
 method), 137  
 parameter() (glotaran.builtin.megacomplexes.decay.irf.IrfLorentzian  
 method), 145  
 parameter() (glotaran.builtin.megacomplexes.decay.irf.IrfPseudoVoigt  
 method), 151  
 parameter() (glotaran.builtin.megacomplexes.decay.irf.IrfSpectralGaussian  
 method), 157  
 parameter\_format (glotaran.plugin\_system.project\_io\_registration\_options.  
 attribute), 314  
 parameter\_history (glotaran.project.result.Result at-  
 tribute), 325  
 parameter\_labels (glotaran.parameter.parameter\_history.ParameterHistory  
 property), 286  
 ParameterGroup (class in  
 glotaran.parameter.parameter\_group), 268  
 ParameterHistory (class in  
 glotaran.parameter.parameter\_history), 282  
 parameters (glotaran.analysis.optimization\_group.OptimizationGroup  
 property), 58  
 parameters (glotaran.builtin.megacomplexes.decay.initial\_concentration.  
 property), 132  
 parameters (glotaran.parameter.parameter\_history.ParameterHistory  
 property), 286  
 parameters (glotaran.project.scheme.Scheme attribute),  
 332  
 parameters (glotaran.testing.model\_generators.SimpleModelGenerator  
 property), 338  
 parameters\_dict (glotaran.testing.model\_generators.SimpleModelGenerator  
 property), 338  
 parse\_version() (in module  
 glotaran.deprecation.deprecation\_utils),  
 201  
 partition() (glotaran.utils.ipython.MarkdownStr  
 method), 358  
 plugin\_list\_cmd() (in module  
 glotaran.cli.commands.pluginlist), 187  
 pop() (glotaran.parameter.parameter\_group.ParameterGroup  
 method), 280  
 pop() (glotaran.utils.io.DatasetMapping method), 347  
 popitem() (glotaran.parameter.parameter\_group.ParameterGroup  
 method), 280

popitem() (glotaran.utils.io.DatasetMapping method),  
 348  
 prepare\_result\_creation()  
 (glotaran.analysis.optimization\_group\_calculator.OptimizationGroupCalculator  
 method), 61  
 prepare\_result\_creation()  
 (glotaran.analysis.optimization\_group\_calculator\_linked.OptimizationGroupCalculatorLinked  
 method), 69  
 prepare\_result\_creation()  
 (glotaran.analysis.optimization\_group\_calculator\_unlinked.OptimizationGroupCalculatorUnlinked  
 method), 72  
 prepare\_time\_trace\_dataset() (in module  
 glotaran.io.prepare\_dataset), 213  
 print\_cmd() (in module glotaran.cli.commands.print),  
 187  
 problem\_list() (glotaran.model.model.Model  
 method), 241  
 problem\_list() (glotaran.project.scheme.Scheme  
 method), 332  
 project\_io\_list\_supporting\_plugins() (in mod-  
 ule glotaran.cli.commands.util), 188  
 project\_io\_plugin\_table() (in module  
 glotaran.plugin\_system.project\_io\_registration),  
 308  
 ProjectIoInterface (class in glotaran.io.interface),  
 209  
 protect\_from\_overwrite() (in module  
 glotaran.plugin\_system.io\_plugin\_utils),  
 302  
 raise\_deprecation\_error() (in module  
 glotaran.deprecation.deprecation\_utils),  
 302  
 rates (glotaran.builtin.megacomplexes.damped\_oscillation.damped\_oscillation  
 property), 123  
 rates (glotaran.testing.model\_generators.SimpleModelGenerator  
 attribute), 338  
 rates() (glotaran.builtin.megacomplexes.decay.k\_matrix.KMatrix  
 method), 164  
 read() (glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.ExplicitFile  
 method), 84  
 read() (glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.TimeExplicitFile  
 method), 87  
 read() (glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.WavelengthTimeExplicitFile  
 method), 90  
 read\_model\_from\_yaml() (in module  
 glotaran.deprecation.modules.glotaran\_root),  
 206  
 read\_model\_from\_yaml\_file() (in module  
 glotaran.deprecation.modules.glotaran\_root),  
 206  
 read\_parameters\_from\_csv\_file() (in module  
 glotaran.deprecation.modules.glotaran\_root),



- 206  
[read\\_parameters\\_from\\_yaml\(\)](#) (in module [glotaran.deprecation.modules.glotaran\\_root](#)), 207  
[read\\_parameters\\_from\\_yaml\\_file\(\)](#) (in module [glotaran.deprecation.modules.glotaran\\_root](#)), 207  
[recreate\(\)](#) ([glotaran.project.result.Result](#) method), 326  
[reduce\\_matrix\(\)](#) (in module [glotaran.analysis.util](#)), 76  
[reduced\(\)](#) ([glotaran.builtin.megacomplexes.decay.k\\_matrix\\_kinetic](#) method), 164  
[reduced\\_chi\\_square](#) ([glotaran.project.result.Result](#) attribute), 326  
[reduced\\_clps](#) ([glotaran.analysis.optimization\\_group.OptimizationGroup](#) property), 58  
[reduced\\_matrices](#) ([glotaran.analysis.optimization\\_group.OptimizationGroup](#) property), 58  
[RegexPattern](#) (class in [glotaran.utils.regex](#)), 359  
[register\\_data\\_io\(\)](#) (in module [glotaran.plugin\\_system.data\\_io\\_registration](#)), 297  
[register\\_megacomplex\(\)](#) (in module [glotaran.plugin\\_system.megacomplex\\_registration](#)), 304  
[register\\_project\\_io\(\)](#) (in module [glotaran.plugin\\_system.project\\_io\\_registration](#)), 309  
[registered\\_plugins\(\)](#) (in module [glotaran.plugin\\_system.base\\_registry](#)), 293  
[Relation](#) (class in [glotaran.model.relation](#)), 250  
[relative\\_posix\\_path\(\)](#) (in module [glotaran.utils.io](#)), 344  
[replace\(\)](#) ([glotaran.utils.ipython.MarkdownStr](#) method), 358  
[report](#) ([glotaran.plugin\\_system.project\\_io\\_registration.SaveOptions](#) attribute), 314  
[reset\(\)](#) ([glotaran.analysis.optimization\\_group.OptimizationGroup](#) method), 58  
[residual\\_function](#) ([glotaran.model.dataset\\_group.DatasetGroupModel](#) attribute), 227  
[residual\\_nnls\(\)](#) (in module [glotaran.analysis.nnls](#)), 54  
[residual\\_variable\\_projection\(\)](#) (in module [glotaran.analysis.variable\\_projection](#)), 78  
[residuals](#) ([glotaran.analysis.optimization\\_group.OptimizationGroup](#) property), 58  
[Result](#) (class in [glotaran.project.result](#)), 317  
[result\\_path](#) ([glotaran.project.scheme.Scheme](#) attribute), 332  
[retrieve\\_clps\(\)](#) (in module [glotaran.analysis.util](#)), 76  
[retrieve\\_decay\\_associated\\_data\(\)](#) (in module [glotaran.builtin.megacomplexes.decay.util](#)), 166  
[retrieve\\_irf\(\)](#) (in module [glotaran.builtin.megacomplexes.decay.util](#)), 166  
[retrieve\\_species\\_associated\\_data\(\)](#) (in module [glotaran.builtin.megacomplexes.decay.util](#)), 166  
[rfind\(\)](#) ([glotaran.utils.ipython.MarkdownStr](#) method), 358  
[rindex\(\)](#) ([glotaran.utils.ipython.MarkdownStr](#) method), 358  
[rindex\(\)](#) ([glotaran.utils.ipython.MarkdownStr](#) method), 358  
[root\\_group](#) ([glotaran.parameter.parameter\\_group.ParameterGroup](#) property), 280  
[root\\_mean\\_square\\_error](#) ([glotaran.project.result.Result](#) attribute), 326  
[rpartition\(\)](#) ([glotaran.utils.ipython.MarkdownStr](#) method), 358  
[rsplit\(\)](#) ([glotaran.utils.ipython.MarkdownStr](#) method), 358  
[rstrip\(\)](#) ([glotaran.utils.ipython.MarkdownStr](#) method), 358
- ## S
- [sanitize\\_dict\\_keys\(\)](#) (in module [glotaran.utils.sanitize](#)), 362  
[sanitize\\_dict\\_values\(\)](#) (in module [glotaran.utils.sanitize](#)), 363  
[sanitize\\_list\\_with\\_broken\\_tuples\(\)](#) (in module [glotaran.utils.sanitize](#)), 363  
[sanitize\\_parameter\\_list\(\)](#) (in module [glotaran.utils.sanitize](#)), 363  
[sanitize\\_yaml\(\)](#) (in module [glotaran.utils.sanitize](#)), 363  
[save\\_ascii\\_scientific\\_notation\\_conversion\(\)](#) (in module [glotaran.utils.sanitize](#)), 364  
[save\\_csv\(\)](#) ([glotaran.project.result.Result](#) method), 326  
[save\\_dataset\(\)](#) ([glotaran.builtin.io.ascii.wavelength\\_time\\_explicit\\_file.AsciDataset](#) method), 81  
[save\\_dataset\(\)](#) ([glotaran.builtin.io.netCDF.netCDF.NetCDFDataIo](#) method), 100  
[save\\_dataset\(\)](#) ([glotaran.builtin.io.sdt.sdt\\_file\\_reader.SdtDataIo](#) method), 102  
[save\\_dataset\(\)](#) ([glotaran.io.interface.DataIoInterface](#) method), 209  
[save\\_dataset\(\)](#) (in module [glotaran.plugin\\_system.data\\_io\\_registration](#)), 298  
[save\\_model\(\)](#) ([glotaran.builtin.io.csv.csv.CsvProjectIo](#) method), 94  
[save\\_model\(\)](#) ([glotaran.builtin.io.folder.folder\\_plugin.FolderProjectIo](#) method), 98  
[save\\_model\(\)](#) ([glotaran.builtin.io.yaml.yaml.YmlProjectIo](#) method), 105

`save_model()` (`glotaran.io.interface.ProjectIoInterface` method), 212  
`save_model()` (in module `glotaran.plugin_system.project_io_registration`), 309  
`save_parameters()` (`glotaran.builtin.io.csv.csv.CsvProjectIo` method), 94  
`save_parameters()` (`glotaran.builtin.io.folder.folder_plugin.FolderProjectIo` method), 98  
`save_parameters()` (`glotaran.builtin.io.yml.yml.YmlProjectIo` method), 105  
`save_parameters()` (`glotaran.io.interface.ProjectIoInterface` method), 212  
`save_parameters()` (in module `glotaran.plugin_system.project_io_registration`), 310  
`save_result()` (`glotaran.builtin.io.csv.csv.CsvProjectIo` method), 94  
`save_result()` (`glotaran.builtin.io.folder.folder_plugin.FolderProjectIo` method), 98  
`save_result()` (`glotaran.builtin.io.yml.yml.YmlProjectIo` method), 105  
`save_result()` (`glotaran.io.interface.ProjectIoInterface` method), 212  
`save_result()` (in module `glotaran.plugin_system.project_io_registration`), 310  
`save_scheme()` (`glotaran.builtin.io.csv.csv.CsvProjectIo` method), 94  
`save_scheme()` (`glotaran.builtin.io.folder.folder_plugin.FolderProjectIo` method), 98  
`save_scheme()` (`glotaran.builtin.io.yml.yml.YmlProjectIo` method), 105  
`save_scheme()` (`glotaran.io.interface.ProjectIoInterface` method), 212  
`save_scheme()` (in module `glotaran.plugin_system.project_io_registration`), 311  
`SavingOptions` (class in module `glotaran.plugin_system.project_io_registration`), 313  
`scale` (`glotaran.builtin.megacomplexes.decay.irf.IrfGaussian` property), 137  
`scale` (`glotaran.builtin.megacomplexes.decay.irf.IrfMultiGaussian` property), 145  
`scale` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralGaussian` property), 151  
`scale` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralMultiGaussian` property), 157  
`Scheme` (class in `glotaran.project.scheme`), 327  
`scheme` (`glotaran.project.result.Result` attribute), 326  
`SCHEME_FILE`  
     `glotaran-optimize` command line option, 42  
     `glotaran-print` command line option, 42  
     `glotaran-validate` command line option, 43  
`scheme_spec_deprecations()` (in module `glotaran.deprecation.modules.builtin_io_yaml`), 205  
`SdtDataIo` (class in `glotaran.builtin.io.sdt.sdt_file_reader`), 101  
`select_data()` (in module `glotaran.cli.commands.util`), 189  
`select_name()` (in module `glotaran.cli.commands.util`), 189  
`set_coordinates()` (`glotaran.model.dataset_model.DatasetModel` method), 232  
`set_data()` (`glotaran.model.dataset_model.DatasetModel` method), 232  
`set_data_plugin()` (in module `glotaran.plugin_system.data_io_registration`), 299  
`set_explicit_axis()`  
     (`glotaran.builtin.io.ascii.wavelength_time_explicit_file.ExplicitFile` method), 84  
     (`glotaran.builtin.io.ascii.wavelength_time_explicit_file.TimeExplicitFile` method), 87  
     (`glotaran.builtin.io.ascii.wavelength_time_explicit_file.WavelengthTimeExplicitFile` method), 90  
`set_from_group()` (`glotaran.parameter.parameter.Parameter` method), 267  
`set_from_history()` (`glotaran.parameter.parameter_group.ParameterGroup` method), 280  
`set_from_label_and_value_arrays()`  
     (`glotaran.parameter.parameter_group.ParameterGroup` method), 280  
`set_megacomplex_plugin()` (in module `glotaran.plugin_system.megacomplex_registration`), 304  
`set_plugin()` (in module `glotaran.plugin_system.base_registry`), 293  
`set_project_plugin()` (in module `glotaran.plugin_system.project_io_registration`), 311  
`set_value_from_optimization()`  
     (`glotaran.parameter.parameter.Parameter` method), 267  
`setdefault()` (`glotaran.parameter.parameter_group.ParameterGroup` method), 281  
`setdefault()` (`glotaran.utils.io.DatasetMapping` method), 348  
`setter()` (`glotaran.model.property.ModelProperty` method), 249  
`shape` (`glotaran.builtin.megacomplexes.spectral.spectral_megacomplex.SpectralMegacomplex` property), 185  
`shell_complete()` (`glotaran.cli.commands.util.ValOrRangeOrList` method), 193

`shift` (`glotaran.builtin.megacomplexes.decay.irf.IrfGaussian` property), 137  
`shift` (`glotaran.builtin.megacomplexes.decay.irf.IrfMultiGaussian` property), 145  
`shift` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralShapeGaussian` property), 151  
`shift` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralShapeSkewedGaussian` property), 157  
`show_data_io_method_help()` (in module `glotaran.plugin_system.data_io_registration`), 299  
`show_method_help()` (in module `glotaran.plugin_system.base_registry`), 294  
`show_project_io_method_help()` (in module `glotaran.plugin_system.project_io_registration`), 312  
`signature_analysis()` (in module `glotaran.cli.commands.util`), 189  
`SimpleModelGenerator` (class in `glotaran.testing.model_generators`), 334  
`simulate()` (in module `glotaran.analysis.simulation`), 73  
`simulate_clp()` (in module `glotaran.analysis.simulation`), 73  
`simulate_global_model()` (in module `glotaran.analysis.simulation`), 74  
`skewness` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeSkewedGaussian` property), 178  
`source` (`glotaran.model.clp_penalties.EqualAreaPenalty` property), 218  
`source` (`glotaran.model.relation.Relation` property), 253  
`source_intervals` (`glotaran.model.clp_penalties.EqualAreaPenalty` property), 218  
`source_path` (`glotaran.project.result.Result` attribute), 326  
`source_path` (`glotaran.project.scheme.Scheme` attribute), 332  
`source_path` (`glotaran.typing.protocols.FileLoadableProtocol` attribute), 343  
`source_path` (`glotaran.utils.io.DatasetMapping` property), 348  
`SpectralMegaComplex` (class in `glotaran.builtin.megacomplexes.spectral.spectral_mega_complex`), 181  
`SpectralShape` (class in `glotaran.builtin.megacomplexes.spectral.shape`), 167  
`SpectralShapeGaussian` (class in `glotaran.builtin.megacomplexes.spectral.shape`), 168  
`SpectralShapeOne` (class in `glotaran.builtin.megacomplexes.spectral.shape`), 172  
`SpectralShapeSkewedGaussian` (class in `glotaran.builtin.megacomplexes.spectral.shape`), 174  
`SpectralShapeZero` (class in `glotaran.builtin.megacomplexes.spectral.shape`), 178  
`splitlines()` (`glotaran.utils.ipython.MarkdownStr` method), 358  
`splitlines()` (`glotaran.cli.commands.util.ValOrRangeOrList` method), 193  
`splitlines()` (`glotaran.utils.ipython.MarkdownStr` method), 358  
`standard_error` (`glotaran.parameter.parameter.Parameter` property), 267  
`startswith()` (`glotaran.utils.ipython.MarkdownStr` method), 359  
`STD_ERR` (`glotaran.parameter.parameter.Keys` attribute), 260  
`string_to_tuple()` (in module `glotaran.utils.sanitize`), 364  
`strip()` (`glotaran.utils.ipython.MarkdownStr` method), 359  
`success` (`glotaran.project.result.Result` attribute), 326  
`swap_dimensions()` (`glotaran.model.dataset_model.DatasetModel` method), 232  
`swapcase()` (`glotaran.utils.ipython.MarkdownStr` method), 359

## T

`target` (`glotaran.model.clp_penalties.EqualAreaPenalty` property), 218  
`target` (`glotaran.model.constraint.OnlyConstraint` property), 222  
`target` (`glotaran.model.constraint.ZeroConstraint` property), 224  
`target` (`glotaran.model.relation.Relation` property), 253  
`target_intervals` (`glotaran.model.clp_penalties.EqualAreaPenalty` property), 218  
`termination_reason` (`glotaran.project.result.Result` attribute), 326  
`time_explicit` (`glotaran.builtin.io.ascii.wavelength_time_explicit_file.DatasetMapping` attribute), 82  
`time_explicit_file` (class in `glotaran.builtin.io.ascii.wavelength_time_explicit_file`), 85  
`times()` (`glotaran.builtin.io.ascii.wavelength_time_explicit_file.WavelengthTimeExplicitFile` method), 90  
`title()` (`glotaran.utils.ipython.MarkdownStr` method), 359  
`to_csv()` (`glotaran.parameter.parameter_group.ParameterGroup` method), 281  
`to_csv()` (`glotaran.parameter.parameter_history.ParameterHistory` method), 286  
`to_dataframe()` (`glotaran.parameter.parameter_group.ParameterGroup` method), 281

`to_dataframe()` (`glotaran.parameter.parameter_history.ParameterHistory` method), 286  
`to_dataframe()` (`glotaran.parameter.parameter_group.ParameterGroup` method), 286  
`to_info_dict()` (`glotaran.cli.commands.util.ValOrRangeOrList` method), 193  
`to_parameter_dict_list()` (`glotaran.parameter.parameter_group.ParameterGroup` property), 338  
`transformed_expression` (`glotaran.parameter.parameter.Parameter` property), 267  
`translate()` (`glotaran.utils.ipython.MarkdownStr` method), 359  
`tuple_number` (`glotaran.utils.regex.RegexPattern` attribute), 361  
`tuple_word` (`glotaran.utils.regex.RegexPattern` attribute), 361  
`type` (`glotaran.builtin.megacomplexes.baseline.baseline_megacomplex` property), 110  
`type` (`glotaran.builtin.megacomplexes.coherent_artifact.coherent_artifact_megacomplex` property), 116  
`type` (`glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation_megacomplex` property), 123  
`type` (`glotaran.builtin.megacomplexes.decay.decay_megacomplex` property), 128  
`type` (`glotaran.builtin.megacomplexes.decay.irf.IrfGaussian` property), 138  
`type` (`glotaran.builtin.megacomplexes.decay.irf.IrfMeasured` property), 140  
`type` (`glotaran.builtin.megacomplexes.decay.irf.IrfMultiGaussian` property), 145  
`type` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralGaussian` property), 151  
`type` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralMultiGaussian` property), 157  
`type` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralMultiGaussian` property), 157  
`type` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShape` property), 171  
`type` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShape` property), 171  
`type` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShape` property), 174  
`type` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShape` property), 174  
`type` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShape` property), 178  
`type` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShape` property), 180  
`type` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShape` property), 180  
`type` (`glotaran.builtin.megacomplexes.spectral.spectral_megacomplex` property), 185  
`type` (`glotaran.builtin.megacomplexes.spectral.spectral_megacomplex` property), 185  
**U**  
`update()` (`glotaran.parameter.parameter_group.ParameterGroup` method), 281  
`update()` (`glotaran.utils.io.DatasetMapping` method), 348  
`update_parameter_expression()` (`glotaran.parameter.parameter_group.ParameterGroup` method), 281  
`valid` (`glotaran.testing.model_generators.SimpleModelGenerator` property), 338  
`valid()` (`glotaran.model.model.Model` method), 241  
`valid()` (`glotaran.project.scheme.Scheme` method), 333  
`valid_label()` (`glotaran.parameter.parameter.Parameter` static method), 267  
`validate()` (`glotaran.builtin.megacomplexes.baseline.baseline_megacomplex` method), 110  
`validate()` (`glotaran.builtin.megacomplexes.coherent_artifact.coherent_artifact_megacomplex` method), 116  
`validate()` (`glotaran.builtin.megacomplexes.damped_oscillation.damped_oscillation_megacomplex` method), 123  
`validate()` (`glotaran.builtin.megacomplexes.decay.decay_megacomplex` method), 129  
`validate()` (`glotaran.builtin.megacomplexes.decay.irf.IrfGaussian` method), 138  
`validate()` (`glotaran.builtin.megacomplexes.decay.irf.IrfMeasured` method), 140  
`validate()` (`glotaran.builtin.megacomplexes.decay.irf.IrfMultiGaussian` method), 145  
`validate()` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralGaussian` method), 151  
`validate()` (`glotaran.builtin.megacomplexes.decay.irf.IrfSpectralMultiGaussian` method), 157  
`validate()` (`glotaran.builtin.megacomplexes.decay.k_matrix.KMatrix` method), 164  
`validate()` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShape` method), 171  
`validate()` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShape` method), 174  
`validate()` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShape` method), 178  
`validate()` (`glotaran.builtin.megacomplexes.spectral.shape.SpectralShape` method), 180  
`validate()` (`glotaran.builtin.megacomplexes.spectral.spectral_megacomplex` method), 185  
`validate()` (`glotaran.builtin.megacomplexes.spectral.spectral_megacomplex` method), 185  
`validate()` (`glotaran.model.constraint.OnlyConstraint` method), 222  
`validate()` (`glotaran.model.constraint.ZeroConstraint` method), 225  
`validate()` (`glotaran.model.interval_property.IntervalProperty` method), 234  
`validate()` (`glotaran.model.model.Model` method), 242  
`validate()` (`glotaran.model.relation.Relation` method), 253



[validate\(\)](#) (*glotaran.model.weight.Weight* method), 258  
[validate\(\)](#) (*glotaran.project.scheme.Scheme* method), 333  
[validate\(\)](#) (*glotaran.testing.model\_generators.SimpleModelGenerator* method), 338  
[validate\\_cmd\(\)](#) (in module *glotaran.cli.commands.validate*), 193  
[ValOrRangeOrList](#) (class in *glotaran.cli.commands.util*), 189  
[value](#) (*glotaran.model.weight.Weight* property), 258  
[value](#) (*glotaran.parameter.parameter.Parameter* property), 267  
[values\(\)](#) (*glotaran.parameter.parameter\_group.ParameterGroup* method), 281  
[values\(\)](#) (*glotaran.utils.io.DatasetMapping* method), 348  
[VARY](#) (*glotaran.parameter.parameter.Keys* attribute), 260  
[vary](#) (*glotaran.parameter.parameter.Parameter* property), 267  
[verify\(\)](#) (*glotaran.project.result.Result* method), 326

## W

[warn\\_deprecated\(\)](#) (in module *glotaran.deprecation.deprecation\_utils*), 202  
[wavelength\\_explicit](#) (*glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.DataType* attribute), 82  
[WavelengthExplicitFile](#) (class in *glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file*), 88  
[wavelengths\(\)](#) (*glotaran.builtin.io.ascii.wavelength\_time\_explicit\_file.WavelengthExplicitFile* method), 90  
[Weight](#) (class in *glotaran.model.weight*), 255  
[weight](#) (*glotaran.analysis.optimization\_group\_calculator\_linked.DatasetIndexModelGroup* attribute), 66  
[weight](#) (*glotaran.model.clp\_penalties.EqualAreaPenalty* property), 218  
[weighted\\_residuals](#) (*glotaran.analysis.optimization\_group.OptimizationGroup* property), 58  
[width](#) (*glotaran.builtin.megacomplexes.coherent\_artifact.coherent\_artifact\_megacomplex.CoherentArtifactMegacomplex* property), 116  
[width](#) (*glotaran.builtin.megacomplexes.decay.irf.IrfGaussian* property), 138  
[width](#) (*glotaran.builtin.megacomplexes.decay.irf.IrfMultiGaussian* property), 145  
[width](#) (*glotaran.builtin.megacomplexes.decay.irf.IrfSpectralGaussian* property), 151  
[width](#) (*glotaran.builtin.megacomplexes.decay.irf.IrfSpectralMultiGaussian* property), 158  
[width](#) (*glotaran.builtin.megacomplexes.spectral.shape.SpectralShapeGaussian* property), 172

## X

[xtol](#) (*glotaran.project.scheme.Scheme* attribute), 333

## Y

[YmlProjectIo](#) (class in *glotaran.builtin.io.yml.yml*), 103

## Z

[zero\\_constraint](#) (class in *glotaran.model.constraint*), 222  
[zfill\(\)](#) (*glotaran.utils.ipython.MarkdownStr* method), 359