Spine Database API Documentation

Release 0.31.1

Spine project consortium

May 14, 2024

CONTENTS:

CHAPTER

ONE

FRONT MATTER

Information about the Spine database API project.

1.1 Project homepage

Spine database API is hosted on GitHub at https://github.com/spine-tools/Spine-Database-API under the Spine project.

1.2 Installation

Install released versions of Spine database API from the project repository with pip or a similar tool:

```
pip install git+https://github.com/spine-tools/Spine-Database-API.git
```

1.3 Dependencies

Spine database API's install process will ensure that SQLAlchemy is installed, in addition to other dependencies. Spine database API will work with SQLAlchemy as of version 1.3.0.

1.4 Bugs

Bugs and feature enhancements to Spine database API should be reported on the GitHub issue tracker.

CHAPTER

TWO

TUTORIAL

The Spine DB API allows one to create and manipulate Spine databases in a standard way, using SQLAlchemy as the underlying engine. This tutorial provides a quick introduction to the usage of the package.

To begin, make sure Spine database API is installed as described in *Installation*.

2.1 Database Mapping

The main mean of communication with a Spine DB is the *DatabaseMapping*, specially designed to retrieve and modify data from the DB. To create a *DatabaseMapping*, we just pass the URL of the DB to the class constructor:

```
import spinedb_api as api
from spinedb_api import DatabaseMapping

url = "mysql+pymysql://spine_db"  # The URL of an existing Spine DB

with DatabaseMapping(url) as db_map:
    # Do something with db_map
    pass
```

The URL should be formatted following the RFC-1738 standard, as described here.

Note: Currently supported database backends are SQLite and MySQL.

2.2 Creating a DB

If you're following this tutorial, chances are you don't have a Spine DB to play with just yet. We can remediate this by creating a SQLite DB (which is just a file in your system), as follows:

```
import spinedb_api as api
from spinedb_api import DatabaseMapping

url = "sqlite:///first.sqlite"

with DatabaseMapping(url, create=True) as db_map:
    # Do something with db_map
    pass
```

The above will create a file called `first.sqlite` in your current working directoy. Note that we pass the keyword argument `create=True` to *DatabaseMapping* to explicitly say that we want the DB to be created at the given URL if it does not exists already.

Note: In the remainder we will skip the above step and work directly with `db_map`. In other words, all the examples below assume we are inside the `with` block above.

2.3 Adding data

To insert data, we use e.g. *add_entity_class_item()*, *add_entity_item()*, and so on.

Let's begin the party by adding a couple of entity classes:

```
db_map.add_entity_class_item(name="fish", description="It swims.")
db_map.add_entity_class_item(name="cat", description="Eats fish.")
```

Now let's add a multi-dimensional entity class between the two above. For this we need to specify the class names as *dimension_name_list*:

```
db_map.add_entity_class_item(
    name="fish__cat",
    dimension_name_list=("fish", "cat"),
    description="A fish getting eaten by a cat?",
)
```

Let's add entities to our zero-dimensional classes:

```
db_map.add_entity_item(entity_class_name="fish", name="Nemo", description="Lost (for␣
↪now).")
db_map.add_entity_item(
    entity_class_name="cat", name="Felix", description="The wonderful wonderful cat."
)
```

Let's add a multi-dimensional entity to our multi-dimensional class. For this we need to specify the entity names as *element_name_list*:

```
db_map.add_entity_item(entity_class_name="fish__cat", element_name_list=("Nemo", "Felix
↪"))
```

Let's add a parameter definition for one of our entity classes:

```
db_map.add_parameter_definition_item(entity_class_name="fish", name="color")
```

Finally, let's specify a parameter value for one of our entities. First, we use *to_database()* to convert the value we want to give into a tuple of `value` and `type`:

```
value, type_ = api.to_database("mainly orange")
```

Now we create our parameter value:

```
db_map.add_parameter_value_item(
    entity_class_name="fish",
```

(continues on next page)

(continued from previous page)

```
    entity_byname=("Nemo",),
    parameter_definition_name="color",
    alternative_name="Base",
    value=value,
    type=type_
)
```

Note that in the above, we refer to the entity by its *byname*. We also set the value to belong to an *alternative* called `Base` which is readily available in new databases.

Note: The data we've added so far is not yet in the DB, but only in an in-memory mapping within our `db_map` object. Don't worry, we will save it to the DB soon (see *Committing data* if you're impatient).

2.4 Retrieving data

To retrieve data, we use e.g. *get_entity_class_item()*, *get_entity_item()*, etc. This implicitly fetches data from the DB into the in-memory mapping, if not already there. For example, let's find one of the entities we inserted above:

```
felix_item = db_map.get_entity_item(entity_class_name="cat", name="Felix")
assert felix_item["description"] == "The wonderful wonderful cat."
```

Above, `felix_item` is a `PublicItem` object, representing an item.

Let's find our multi-dimensional entity:

```
nemo_felix_item = db_map.get_entity_item("entity", entity_class_name="fish__cat",␣
↪element_name_list=("Nemo", "Felix"))
assert nemo_felix_item["dimension_name_list"] == ('fish', 'cat')
```

Now let's retrieve our parameter value:

```
nemo_color_item = db_map.get_parameter_value_item(
    entity_class_name="fish",
    entity_byname=("Nemo",),
    parameter_definition_name="color",
    alternative_name="Base"
)
```

We can use the `"parsed_value"` field to access our original value:

```
nemo_color = nemo_color_item["parsed_value"]
assert nemo_color == "mainly orange"
```

To retrieve all the items of a given type, we use *get_items()*:

```
assert [entity["entity_byname"] for entity in db_map.get_items("entity")] == [
    ("Nemo",), ("Felix",), ("Nemo", "Felix")
]
```

Now you should use the above to try and find Nemo.

2.5 Updating data

To update data, we use the `update()` method of `PublicItem`.

Let's rename our fish entity to avoid any copyright infringements:

```
db_map.get_entity_item(entity_class_name="fish", name="Nemo").update(name="NotNemo")
```

To be safe, let's also change the color:

```
new_value, new_type = api.to_database("not that orange")
db_map.get_parameter_value_item(
    entity_class_name="fish",
    entity_byname=("NotNemo",),
    parameter_definition_name="color",
    alternative_name="Base",
).update(value=new_value, type=new_type)
```

Note how we need to use then new entity name `NotNemo` to retrieve the parameter value. This makes sense.

2.6 Removing data

You know what, let's just remove the entity entirely. To do this we use the `remove()` method of `PublicItem`:

```
db_map.get_entity_item(entity_class_name="fish", name="NotNemo").remove()
```

Note that the above call removes items in *cascade*, meaning that items that depend on `"NotNemo"` will get removed as well. We have one such item in the database, namely the `"color"` parameter value which also gets dropped when the above method is called.

2.7 Restoring data

TODO

2.8 Committing data

Enough messing around. To save the contents of the in-memory mapping into the DB, we use *commit_session()*:

```
db_map.commit_session("Find Nemo, then lose him again")
```

CHAPTER

THREE

PARAMETER VALUE FORMAT

Note: Client code should almost never convert parameter values to JSON and back manually. For most cases, JSON should be considered an implementation detail. Clients should rather use *to_database()* and *from_database()* which shield from abrupt changes in the database representation.

Parameter values are specified using JSON in the `value` field of the `parameter_value` table. This document describes the JSON specification for parameter values of special type (namely, date-time, duration, time-pattern, time-series, array, and map.)

A value of special type is a JSON object with two mandatory properties, `type` and `data`:

- `type` indicates the value *type* and must be a JSON string (either `date_time`, `duration`, `dictionary`, `time_pattern`, `time_series`, `array`, or `map`).
- `data` specifies the value *itself* and must be a JSON object in accordance with `type` as explained below.

3.1 Date-time

If the `type` property is `date_time`, then the `data` property specifies a date/time and must be a JSON string in the ISO8601 format.

3.1.1 Example

```
{
  "type": "date_time",
  "data": "2019-06-01T22:15:00+01:00"
}
```

3.2 Duration

If the `type` property is `duration`, then the `data` property specifies an extension of time where the accepted values are the following:

- The number of time-units, specified as a 'verbose' JSON string. The format is `x unit`, where `x` is an integer and `unit` is either `year`, `month`, `day`, `hour`, `minute`, or `second` (either singular or plural).
- The number of time-units, specified as a 'compact' JSON string. The format is `xU`, where `x` is an integer and `U` is either `Y` (for year), `M` (for month), `D` (for day), `h` (for hour), `m` (for minute), or `s` (for second).

- The number of *minutes*, specified as a JSON integer.

Note: The array version of Duration is deprecated and no longer supported. Use the Array type for variable durations.

3.2.1 Examples

Verbose string:

```
{
  "type": "duration",
  "data": "1 hour"
}
```

Compact string:

```
{
  "type": "duration",
  "data": "1h"
}
```

Integer:

```
{
  "type": "duration",
  "data": 60
}
```

3.3 Time-pattern

If the `type` property is `time_pattern`, then the `data` property specifies *time-patterned data*. This is data that varies *periodically* in time taking specific *values* at specific *time-periods* (such as summer and winter). Values must be JSON numbers, whereas time-periods must be JSON strings where the accepted values are the following:

- An interval of time in a given time-unit. The format is `Ua-b`, where `U` is either `Y` (for year), `M` (for month), `D` (for day), `WD` (for weekday), `h` (for hour), `m` (for minute), or `s` (for second); and `a` and `b` are two integers corresponding to the lower and upper bound, respectively.
- An intersection of intervals. The format is `s1;s2;...`, where `s1`, `s2`, ..., are intervals as described above.
- A union of ranges. The format is `r1,r2,...`, where `r1`, `r2`, ..., are either intervals or intersections of intervals as described above.

The `data` property must be a JSON object mapping time periods to values.

A time-pattern may have an additional property, `index_name`. `index_name` must be a JSON string. If not specified, a default name 'p' will be used.

3.3.1 Example

The following corresponds to a parameter which takes the value `300` in months 1 to 4 *and* 9 to 12, and the value `221.5` in months 5 to 8.

```
{
  "type": "time_pattern",
  "data": {
    "M1-4,M9-12": 300,
    "M5-8": 221.5
  }
}
```

3.4 Time-series

If the `type` property is `time_series`, then the `data` property specifies time-series data. This is data that varies *arbitrarily* in time taking specific *values* at specific *time-stamps*. Values must be JSON numbers, whereas time-stamps must be JSON strings in the ISO8601 format.

Accepted values for the `data` property are the following:

- A JSON object mapping time-stamps to values.
- A two-column JSON array listing tuples of the form [time-stamp, value].
- A (one-column) JSON array of values. In this case it is assumed that the time-series begins at the first hour of *any* year, has a resolution of one hour, and repeats cyclically until the *end* of time.

In case of time-series, the specification may have two additional properties, `index` and `index_name`. `index` must be a JSON object with the following properties, all of them optional:

- `start`: the *first* time-stamp, used in case `data` is a one-column array (ignored otherwise). It must be a JSON string in the ISO8601 format. The default is `0001-01-01T00:00:00`.
- `resolution`: the 'time between stamps', used in case `data` is a one-column array (ignored otherwise). Accepted values are the same as for the `data` property of [duration](#duration) values. The default is `1 hour`. If `resolution` is itself an array, then it is either trunk or repeated so as to fit `data`.
- `ignore_year`: a JSON boolean to indicate whether or not the time-series should apply to *any* year. The default is `false`, unless `data` is a one-column array and `start` is not given.
- `repeat`: a JSON boolean whether or not the time-series should repeat cyclically until the *end* of time. The default is `false`, unless `data` is a one-column array and `start` is not given.

`index_name` must be a JSON string. If not specified, a default name 't' will be used.

3.4.1 Examples

Dictionary:

```
{
  "type": "time_series",
  "data": {
    "2019-01-01T00:00": 1,
    "2019-01-01T01:30": 5,
```

(continues on next page)

(continued from previous page)

```
    "2019-01-01T02:00": 8
  }
}
```

Two-column array:

```
{

  "type": "time_series",
  "data": [
    ["2019-01-01T00:00", 1],
    ["2019-01-01T00:30", 2],
    ["2019-01-01T02:00", 8]
  ]
}
```

One-column array with implicit (default) indices:

```
{
  "type": "time_series",
  "data": [1, 2, 3, 5, 8]
}
```

One-column array with explicit (custom) indices:

```
{
  "type": "time_series",
  "data": [1, 2, 3, 5, 8],
  "index": {
    "start": "2019-01-01T00:00",
    "resolution": "30 minutes",
    "ignore_year": false,
    "repeat": true
  }
}
```

Two-column array with named indices:

```
{

  "type": "time_series",
  "data": [
    ["2019-01-01T00:00", 1],
    ["2019-01-01T00:30", 2],
    ["2019-01-01T02:00", 8]
  ],
  "index_name": "Time stamps"
}
```

3.5 Array

If the `type` property is `array`, then the `data` property specifies a one dimensional array. This is a list of values with zero based indexing. All values are of the same type which is specified by an optional `value_type` property. If specified, `value_type` must be one of the following: `float`, `str`, `duration`, or `date_time`. If omitted, `value_type` defaults to `float`

The `data` property must be a JSON list. The elements depend on `value_type`:

- If `value_type` is `float` then all elements in `data` must be JSON numbers.
- If `value_type` is `str` then all elements in `data` must be JSON strings.
- If `value_type` is `duration` then all elements in `data` must be single extensions of time.
- If `value_type` is `date_time` then all elements in `data` must be JSON strings in the ISO8601 format.

An array may have an additional property, `index_name`. `index_name` must be a JSON string. If not specified, a default name 'i' will be used.

3.5.1 Examples

An array of numbers:

```
{
  "type": "array",
  "data": [2.3, 23.0, 5.0]
}
```

An array of durations:

```
{
  "type": "array",
  "value_type": "duration",
  "data": ["3 months", "2Y", "4 minutes"]
}
```

An array of strings with index name:

```
{
  "type": "array",
  "data": ["one", "two"],
  "index_name": "step"
}
```

3.6 Map

If the `type` property is `map`, then the `data` property specifies indexed array data. An additional `index_type` specifies the type of the index and must be one of the following: `float`, `str`, `duration`, or `date_time`.

The `data` property can be a JSON mapping with the following properties:

- Every key in the map must be a scalar of the same type as given by `index_type`:
  - floats are represented by JSON numbers, e.g. `5.5`

  - strings are represented by JSON strings, e.g. `"key_1"`
  - durations are represented by duration strings, e.g. `"1 hour"`. Note that *variable* durations are not supported
  - datetimes are represented by ISO8601 time stamps, e.g. `"2020-01-01T12:00"`
- Every value in the map can be
  - a float, e.g. `5.5`
  - a duration, e.g. `{"type": "duration", "data": "3 days"}`
  - a datetime, e.g. `{"type": "date_time", "data": "2020-01-01T12:00"}`
  - a map, e.g. `{"type": "map", "index_type": "str", "data":{"a": 2, "b": 3}}`
  - any of the following: time-series, array, time-pattern

Optionally, the `data` property can be a two-column JSON array where the first element is the key and the second the value.

A map may have an additional property, `index_name`. `index_name` must be a JSON string. If not specified, a default name 'x' will be used.

3.6.1 Examples

Dictionary:

```
{
  "type": "map",
  "index_type": "date_time",
  "data": {
    "2010-01-01T00:00": {
      "type": "map",
      "index_type": "duration",
      "data": [["1D", -1.0], ["1D", -1.5]]
    },
    "2010-02-01-T00:00": {
      "type": "map",
      "index_type": "duration",
      "data": [["1 month", 2.3], ["2 months", 2.5]]
    }
  }
}
```

Two-column array:

```
{
  "type": "map",
  "index_type": "str",
  "data": [["cell_1", 1.0], ["cell_2", 2.0], ["cell_3", 3.0]]
}
```

Stochastic time series corresponding to the table below:

Forecast time	Target time	Stochastic scenario	Value
2020-04-17T08:00	2020-04-17T08:00	0	23.0
2020-04-17T08:00	2020-04-17T09:00	0	24.0
2020-04-17T08:00	2020-04-17T10:00	0	25.0
2020-04-17T08:00	2020-04-17T08:00	1	5.5
2020-04-17T08:00	2020-04-17T09:00	1	6.6
2020-04-17T08:00	2020-04-17T10:00	1	7.7


```
{
  "type": "map",
  "index_type": "date_time",
  "index_name": "Forecast time",
  "data": [
    ["2020-04-17T08:00",
     {"type": "map", "index_type": "date_time", "index_name": "Target time", "data": [
       [
         "2020-04-17T08:00", {"type": "map",
                              "index_type": "float",
                              "index_name": "Stochastic scenario",
                              "data": [[0, 23.0], [1, 5.5]]}
       ],
       [
         "2020-04-17T09:00", {"type": "map",
                              "index_type": "float",
                              "index_name": "Stochastic scenario",
                              "data": [[0, 24.0], [1, 6.6]]}
       ],
       [
         "2020-04-17T10:00", {"type": "map",
                              "index_type": "float",
                              "index_name": "Stochastic scenario",
                              "data": [[0, 25.0], [1, 7.7]]}
       ]
     ]}
    ]
  ]
}
```

CHAPTER

FOUR

METADATA

Metadata can be used to provide additional information about the data in Spine data structure. Every entity and parameter value can have metadata associated with it.

A metadata "item" has a *name* and a *value*, e.g. "authors" and "N.N, M.M et al.". The same metadata item can be referenced by multiple entities and parameter values. Entities and values can also refer to multiple items of metadata.

Note: Referring to multiple items of metadata from a huge number of entities or parameter values may take up a lot of space in the database. Therefore, it might make more sense, for example, to list all contributors to the data in a single metadata value than having each contributor as a separate name-value pair.

Choosing suitable names and values for metadata is left up to the user. However, some suggestions and recommendations are presented below.

title
: One sentence description for the data.

sources
: The raw sources of the data.

tools
: Names and versions of tools that were used to process the data.

contributors
: The people or organisations who contributed to the data.

created
: The date this data was created or put together, e.g. in ISO8601 format (YYYY-MM-DDTHH:MM).

description
: A more complete description of the data than the title.

keywords
: Keywords that categorize the data.

homepage
: A URL for the home on the web that is related to the data.

id
: Globally unique id, such as UUID or DOI.

licenses
: Licences that apply to the data.

temporal
: Temporal properties of the data.

spatial
Spatial properties of the data.

unitOfMeasurement
Unit of measurement.

CHAPTER

FIVE

DB MAPPING SCHEMA

The DB mapping schema is a close cousin of the Spine DB schema with some extra flexibility, like the ability to specify references by name rather than by numerical id.

The schema defines the following item types: `alternative`, `scenario`, `scenario_alternative`, `entity_class`, `superclass_subclass`, `entity`, `entity_group`, `entity_alternative`, `parameter_value_list`, `list_value`, `parameter_definition`, `parameter_value`, `metadata`, `entity_metadata`, `parameter_value_metadata`. As you can see, these follow the names of some of the tables in the Spine DB schema.

The following subsections provide all the details you need to know about the different item types, namely, their fields, values, and unique keys.

5.1 alternative

Table 1: Fields and values

field	type	value
name	str	The alternative name.
description	str, optional	The alternative description.

Table 2: Unique keys

name

5.2 scenario

Table 3: Fields and values

field	type	value
name	str	The scenario name.
description	str, optional	The scenario description.
active	bool, optional	Not in use at the moment.

Table 4: Unique keys

name

5.3 scenario_alternative

Table 5: Fields and values

field	type	value
scenario_name	str	The scenario name.
alternative_name	str	The alternative name.
rank	int	The rank - higher has precedence.

Table 6: Unique keys

scenario_name, alternative_name
scenario_name, rank

5.4 entity_class

Table 7: Fields and values

field	type	value
name	str	The class name.
dimension_name_list	tuple, optional	The dimension names for a multi-dimensional class.
description	str, optional	The class description.
display_icon	int, optional	An integer representing an icon within your application.
display_order	int, optional	Not in use at the moment.
hidden	int, optional	Not in use at the moment.
active_by_default	bool, optional	Default activity for the entity alternatives of the class.

Table 8: Unique keys

name

5.5 superclass_subclass

Table 9: Fields and values

field	type	value
superclass_name	str	The superclass name.
subclass_name	str	The subclass name.

Table 10: Unique keys

subclass_name

5.6 entity

Table 11: Fields and values

field	type	value
entity_class_name	str	The entity class name.
name	str	The entity name.
element_name_list	tuple	The element names if the entity is multi-dimensional.
entity_byname	tuple	A tuple with the entity name as single element if the entity is zero-dimensional,or the element names if it is multi-dimensional.
description	str, optional	The entity description.

Table 12: Unique keys

entity_class_name, name
entity_class_name, entity_byname

5.7 entity_group

Table 13: Fields and values

field	type	value
entity_class_name	str	The entity class name.
group_name	str	The group entity name.
member_name	str	The member entity name.

Table 14: Unique keys

entity_class_name, group_name, member_name

5.8 entity_alternative

Table 15: Fields and values

field	type	value
entity_class_name	str	The entity class name.
entity_byname	tuple	A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
alternative_name	str	The alternative name.
active	bool, optional	Whether the entity is active in the alternative - defaults to True.

Table 16: Unique keys

entity_class_name, entity_byname, alternative_name

5.9 parameter_value_list

Table 17: Fields and values

field	type	value
name	str	The parameter value list name.

Table 18: Unique keys

name

5.10 list_value

Table 19: Fields and values

field	type	value
parameter_value_list_name	str	The parameter value list name.
value	bytes	The value.
type	str, optional	The value type.
index	int, optional	The value index.

Table 20: Unique keys

parameter_value_list_name, value_and_type
parameter_value_list_name, index

5.11 parameter_definition

Table 21: Fields and values

field	type	value
entity_class_name	str	The entity class name.
name	str	The parameter name.
default_value	bytes, optional	The default value.
default_type	str, optional	The default value type.
parameter_value_list_name	str, optional	The parameter value list name if any.
description	str, optional	The parameter description.

Table 22: Unique keys

entity_class_name, name

5.12 parameter_value

Table 23: Fields and values

field	type	value
entity_class_name	str	The entity class name.
parameter_definition_name	str	The parameter name.
entity_byname	tuple	A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
value	bytes	The value.
type	str, optional	The value type.
alternative_name	str, optional	The alternative name - defaults to 'Base'.

Table 24: Unique keys

entity_class_name, parameter_definition_name, entity_byname, alternative_name

5.13 metadata

Table 25: Fields and values

field	type	value
name	str	The metadata entry name.
value	str	The metadata entry value.

Table 26: Unique keys

name, value

5.14 entity_metadata

Table 27: Fields and values

field	type	value
entity_class_name	str	The entity class name.
entity_byname	tuple	A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
metadata_name	str	The metadata entry name.
metadata_value	str	The metadata entry value.

Table 28: Unique keys

entity_class_name, entity_byname, metadata_name, metadata_value

5.15 parameter_value_metadata

Table 29: Fields and values

field	type	value
entity_class_name	str	The entity class name.
parameter_definition_name	str	The parameter name.
entity_byname	tuple	A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
alternative_name	str	The alternative name.
metadata_name	str	The metadata entry name.
metadata_value	str	The metadata entry value.

Table 30: Unique keys

entity_class_name, parameter_definition_name, entity_byname, alternative_name, metadata_name, metadata_value

CHAPTER

SIX

API REFERENCE

This page contains auto-generated API reference documentation[1].

6.1 spinedb_api

A package to interact with Spine DBs.

6.1.1 Submodules

spinedb_api.db_mapping

This module defines the *DatabaseMapping* class, the main mean to communicate with a Spine DB. If you're planning to use this class, it is probably a good idea to first familiarize yourself a little bit with the *DB mapping schema*.

Module Contents

Classes

DatabaseMapping	Enables communication with a Spine DB.

class spinedb_api.db_mapping.**DatabaseMapping**(*db_url*, *username=None*, *upgrade=False*, *backup_url=''*, *codename=None*, *create=False*, *apply_filters=True*, *memory=False*, *sqlite_timeout=1800*)

Bases: spinedb_api.db_mapping_query_mixin.DatabaseMappingQueryMixin, spinedb_api.db_mapping_commit_mixin.DatabaseMappingCommitMixin, spinedb_api.db_mapping_base.DatabaseMappingBase

Enables communication with a Spine DB.

The DB is incrementally mapped into memory as data is requested/modified, following the *DB mapping schema*.

Data is typically retrieved using *get_item()* or *get_items()*. If the requested data is already in memory, it is returned from there; otherwise it is fetched from the DB, stored in memory, and then returned. In other words, the data is fetched from the DB exactly once.

For convenience, we also provide specialized 'get' methods for each item type, e.g., *get_entity_item()* and *get_entity_items()*.

[1] Created with sphinx-autoapi

Data is added via *add_item()*; updated via *update_item()*; removed via *remove_item()*; and restored via *restore_item()*. All the above methods modify the in-memory mapping (not the DB itself). These methods also fetch data from the DB into the in-memory mapping to perform the necessary integrity checks (unique and foreign key constraints).

For convenience, we also provide specialized 'add', 'update', 'remove', and 'restore' methods for each item type, e.g., *add_entity_item()*, *update_entity_item()*, *remove_entity_item()*, *restore_entity_item()*.

Modifications to the in-memory mapping are committed (written) to the DB via *commit_session()*, or rolled back (discarded) via *rollback_session()*.

The DB fetch status is reset via *refresh_session()*. This allows new items in the DB (added by other clients in the meantime) to be retrieved as well.

You can also control the fetching process via *fetch_more()* and/or *fetch_all()*. For example, you can call *fetch_more()* in a dedicated thread while you do some work on the main thread. This will nicely place items in the in-memory mapping so you can access them later, without the overhead of fetching them from the DB.

The *query()* method is also provided as an alternative way to retrieve data from the DB while bypassing the in-memory mapping entirely.

You can use this class as a context manager, e.g.:

```
with DatabaseMapping(db_url) as db_map:
    # Do stuff with db_map
    ...
```

Parameters

- **db_url** (str or URL) – A URL in RFC-1738 format pointing to the database to be mapped, or to a DB server.
- **username** (*str, optional*) – A user name. If not given, it gets replaced by the string *anon*.
- **upgrade** (*bool, optional*) – Whether the DB at the given *url* should be upgraded to the most recent version.
- **backup_url** (*str, optional*) – A URL to backup the DB before upgrading.
- **codename** (*str, optional*) – A name to identify this object in your application.
- **create** (*bool, optional*) – Whether to create a new Spine DB at the given *url* if it's not already one.
- **apply_filters** (*bool, optional*) – Whether to apply filters in the *url*'s query segment.
- **memory** (*bool, optional*) – Whether to use a SQLite memory DB as replacement for the original one.
- **sqlite_timeout** (*int, optional*) – The number of seconds to wait before raising SQLite connection errors.

static get_upgrade_db_prompt_data(*url*, *create=False*)

Returns data to prompt the user what to do if the DB at the given url is not the latest version. If it is, then returns None.

Parameters

- **url** (*str*) –
- **create** (*bool,optional*) –

Returns
The title of the prompt str: The text of the prompt dict: Mapping different options, to kwargs to pass to DatabaseMapping constructor in order to apply them dict or None: Mapping different options, to additional notes int or None: The preferred option if any

Return type
str

get_item(*item_type*, *fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns an item matching the arguments, or None if none found.

Example:

```
with DatabaseMapping(db_url) as db_map:
    prince = db_map.get_item("entity", entity_class_name="musician", name=
↪"Prince")
```

Parameters

- **item_type** (*str*) – One of alternative, scenario, scenario_alternative, entity_class, superclass_subclass, entity, entity_group, entity_alternative, parameter_value_list, list_value, parameter_definition, parameter_value, metadata, entity_metadata, parameter_value_metadata.
- **fetch** (*bool, optional*) – Whether to fetch the DB in case the item is not found in memory.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- ****kwargs** – Fields and values for one the unique keys as specified for the item type in *DB mapping schema*.

Returns
PublicItem or None

get_items(*item_type*, *fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns all the items of one type.

Parameters

- **item_type** (*str*) – One of alternative, scenario, scenario_alternative, entity_class, superclass_subclass, entity, entity_group, entity_alternative, parameter_value_list, list_value, parameter_definition, parameter_value, metadata, entity_metadata, parameter_value_metadata.
- **fetch** (*bool, optional*) – Whether to fetch the DB before returning the items.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- ****kwargs** – Fields and values for one the unique keys as specified for the item type in *DB mapping schema*.

Returns
The items.

Return type
list(PublicItem)

add_item(*item_type*, *check=True*, ***kwargs*)

Adds an item to the in-memory mapping.

Example:

```
with DatabaseMapping(db_url) as db_map:
    db_map.add_item("entity_class", name="musician")
    db_map.add_item("entity", entity_class_name="musician", name="Prince")
```

Parameters

- **item_type** (*str*) – One of `alternative`, `scenario`, `scenario_alternative`, `entity_class`, `superclass_subclass`, `entity`, `entity_group`, `entity_alternative`, `parameter_value_list`, `list_value`, `parameter_definition`, `parameter_value`, `metadata`, `entity_metadata`, `parameter_value_metadata`.
- ****kwargs** – Fields and values as specified for the item type in *DB mapping schema*.

Returns
The added item and any errors.

Return type
tuple(`PublicItem` or None, str)

add_items(*item_type*, **items*, *check=True*, *strict=False*)

Adds many items to the in-memory mapping.

Parameters

- **item_type** (*str*) – One of `alternative`, `scenario`, `scenario_alternative`, `entity_class`, `superclass_subclass`, `entity`, `entity_group`, `entity_alternative`, `parameter_value_list`, `list_value`, `parameter_definition`, `parameter_value`, `metadata`, `entity_metadata`, `parameter_value_metadata`.
- ***items** (*Iterable(dict)*) – One or more `dict` objects mapping fields to values of the item type, as specified in *DB mapping schema*.
- **strict** (*bool*) – Whether or not the method should raise `SpineIntegrityError` if the insertion of one of the items violates an integrity constraint.

Returns
items successfully added and found violations.

Return type
tuple(list(`PublicItem`),list(str))

update_item(*item_type*, *check=True*, ***kwargs*)

Updates an item in the in-memory mapping.

Example:

```
with DatabaseMapping(db_url) as db_map:
    prince = db_map.get_item("entity", entity_class_name="musician", name=
↪"Prince")
    db_map.update_item(
        "entity", id=prince["id"], name="the Artist", description="Formerly␣
```

(continues on next page)

(continued from previous page)

```
↪known as Prince."
    )
```

Parameters

- **item_type** (*str*) – One of alternative, scenario, scenario_alternative, entity_class, superclass_subclass, entity, entity_group, entity_alternative, parameter_value_list, list_value, parameter_definition, parameter_value, metadata, entity_metadata, parameter_value_metadata.
- **id** (*int*) – The id of the item to update.
- ****kwargs** – Fields to update and their new values as specified for the item type in *DB mapping schema*.

Returns
The updated item and any errors.

Return type
tuple(PublicItem or None, str)

update_items(*item_type*, **items*, *check=True*, *strict=False*)

Updates many items in the in-memory mapping.

Parameters

- **item_type** (*str*) – One of alternative, scenario, scenario_alternative, entity_class, superclass_subclass, entity, entity_group, entity_alternative, parameter_value_list, list_value, parameter_definition, parameter_value, metadata, entity_metadata, parameter_value_metadata.
- ***items** (*Iterable(dict)*) – One or more dict objects mapping fields to values of the item type, as specified in *DB mapping schema* and including the *id*.
- **strict** (*bool*) – Whether or not the method should raise SpineIntegrityError if the update of one of the items violates an integrity constraint.

Returns
items successfully updated and found violations.

Return type
tuple(list(PublicItem),list(str))

add_update_item(*item_type*, *check=True*, ***kwargs*)

Adds an item to the in-memory mapping if it doesn't exist; otherwise updates the current one.

Parameters

- **item_type** (*str*) – One of alternative, scenario, scenario_alternative, entity_class, superclass_subclass, entity, entity_group, entity_alternative, parameter_value_list, list_value, parameter_definition, parameter_value, metadata, entity_metadata, parameter_value_metadata.
- ****kwargs** – Fields and values as specified for the item type in *DB mapping schema*.

Returns

The added item if any,
the updated item if any, and any errors.

Return type
tuple(PublicItem or None, PublicItem or None, str)

add_update_items(*item_type*, **items*, *check=True*, *strict=False*)

Adds or updates many items into the in-memory mapping.

Parameters

- **item_type** (*str*) – One of alternative, scenario, scenario_alternative, entity_class, superclass_subclass, entity, entity_group, entity_alternative, parameter_value_list, list_value, parameter_definition, parameter_value, metadata, entity_metadata, parameter_value_metadata.
- ***items** (*Iterable(dict)*) – One or more dict objects mapping fields to values of the item type, as specified in *DB mapping schema*.
- **strict** (*bool*) – Whether or not the method should raise SpineIntegrityError if the insertion of one of the items violates an integrity constraint.

Returns

items successfully added,
items successfully updated, and found violations.

Return type
tuple(list(PublicItem),list(PublicItem),list(str))

remove_item(*item_type*, *id_*, *check=True*)

Removes an item from the in-memory mapping.

Example:

```
with DatabaseMapping(db_url) as db_map:
    prince = db_map.get_item("entity", entity_class_name="musician", name=
↪"Prince")
    db_map.remove_item("entity", prince["id"])
```

Parameters

- **item_type** (*str*) – One of alternative, scenario, scenario_alternative, entity_class, superclass_subclass, entity, entity_group, entity_alternative, parameter_value_list, list_value, parameter_definition, parameter_value, metadata, entity_metadata, parameter_value_metadata.
- **id** (*int*) – The id of the item to remove.

Returns
The removed item and any errors.

Return type
tuple(PublicItem or None, str)

remove_items(*item_type*, **ids*, *check=True*, *strict=False*)

Removes many items from the in-memory mapping.

Parameters

- **item_type** (*str*) – One of alternative, scenario, scenario_alternative, entity_class, superclass_subclass, entity, entity_group, entity_alternative, parameter_value_list, list_value, parameter_definition, parameter_value, metadata, entity_metadata, parameter_value_metadata.
- ***ids** (*Iterable(int)*) – Ids of items to be removed.
- **strict** (*bool*) – Whether or not the method should raise SpineIntegrityError if the update of one of the items violates an integrity constraint.

Returns
items successfully removed and found violations.

Return type
tuple(list(PublicItem),list(str))

restore_item(*item_type*, *id_*)

Restores a previously removed item into the in-memory mapping.

Example:

```
with DatabaseMapping(db_url) as db_map:
    prince = db_map.get_item("entity", skip_remove=False, entity_class_name=
↪"musician", name="Prince")
    db_map.restore_item("entity", prince["id"])
```

Parameters

- **item_type** (*str*) – One of alternative, scenario, scenario_alternative, entity_class, superclass_subclass, entity, entity_group, entity_alternative, parameter_value_list, list_value, parameter_definition, parameter_value, metadata, entity_metadata, parameter_value_metadata.
- **id** (*int*) – The id of the item to restore.

Returns
The restored item if any.

Return type
tuple(PublicItem or None, str)

restore_items(*item_type*, **ids*)

Restores many previously removed items into the in-memory mapping.

Parameters

- **item_type** (*str*) – One of alternative, scenario, scenario_alternative, entity_class, superclass_subclass, entity, entity_group, entity_alternative, parameter_value_list, list_value, parameter_definition, parameter_value, metadata, entity_metadata, parameter_value_metadata.
- ***ids** (*Iterable(int)*) – Ids of items to be removed.

Returns
the restored items.

Return type
list(PublicItem)

purge_items(*item_type*)

Removes all items of one type.

Parameters
item_type (*str*) – One of alternative, scenario, scenario_alternative, entity_class, superclass_subclass, entity, entity_group, entity_alternative, parameter_value_list, list_value, parameter_definition, parameter_value, metadata, entity_metadata, parameter_value_metadata.

Returns
True if any data was removed, False otherwise.

Return type
bool

fetch_more(*item_type*, *offset=0*, *limit=None*, ***kwargs*)

Fetches items from the DB into the in-memory mapping, incrementally.

Parameters

- **item_type** (*str*) – One of alternative, scenario, scenario_alternative, entity_class, superclass_subclass, entity, entity_group, entity_alternative, parameter_value_list, list_value, parameter_definition, parameter_value, metadata, entity_metadata, parameter_value_metadata.
- **offset** (*int*) – The initial row.
- **limit** (*int*) – The maximum number of rows to fetch.
- ****kwargs** – Fields and values for one the unique keys as specified for the item type in *DB mapping schema*.

Returns
The items fetched.

Return type
list(PublicItem)

fetch_all(**item_types*)

Fetches items from the DB into the in-memory mapping. Unlike *fetch_more()*, this method fetches entire tables.

Parameters
***item_types** (*Iterable(str)*) – One or more of alternative, scenario, scenario_alternative, entity_class, superclass_subclass, entity, entity_group, entity_alternative, parameter_value_list, list_value, parameter_definition, parameter_value, metadata, entity_metadata, parameter_value_metadata. If none given, then the entire DB is fetched.

query(**args*, ***kwargs*)

Returns a Query object to execute against the mapped DB.

To perform custom SELECT statements, call this method with one or more of the documented subquery properties of DatabaseMappingQueryMixin returning Alias objetcs. For example, to select the entity class with id equal to 1:

```
from spinedb_api import DatabaseMapping
url = 'sqlite:///spine.db'
```

(continues on next page)

(continued from previous page)

```
...
db_map = DatabaseMapping(url)
db_map.query(db_map.entity_class_sq).filter_by(id=1).one_or_none()
```

To perform more complex queries, just use the `Query` interface (which is a close clone of SQL Alchemy's `Query`). For example, to select all entity class names and the names of their entities concatenated in a comma-separated string:

```
from sqlalchemy import func

db_map.query(
    db_map.entity_class_sq.c.name, func.group_concat(db_map.entity_sq.c.name)
).filter(
    db_map.entity_sq.c.class_id == db_map.entity_class_sq.c.id
).group_by(db_map.entity_class_sq.c.name).all()
```

Returns
The resulting query.

Return type
`Query`

commit_session(*comment*, *apply_compatibility_transforms=True*)

Commits the changes from the in-memory mapping to the database.

Parameters

- **comment** (*str*) – commit message
- **apply_compatibility_transforms** (*bool*) – if True, apply compatibility transforms

Returns
compatibility transformations

Return type
tuple(list, list)

rollback_session()

Discards all the changes from the in-memory mapping.

refresh_session()

Resets the fetch status so new items from the DB can be retrieved.

has_external_commits()

Tests whether the database has had commits from other sources than this mapping.

Returns
True if database has external commits, False otherwise

Return type
bool

close()

Closes this DB mapping. This is only needed if you're keeping a long-lived session. For instance:

```
class MyDBMappingWrapper:
    def __init__(self, url):
        self._db_map = DatabaseMapping(url)

    # More methods that do stuff with self._db_map

    def __del__(self):
        self._db_map.close()
```

Otherwise, the usage as context manager is recommended:

```
with DatabaseMapping(url) as db_map:
    # Do stuff with db_map
    ...
    # db_map.close() is automatically called when leaving this block
```

get_filter_configs()

Returns the filters from this mapping's URL.

Return type
list(dict)

get_alternative_item(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns an *alternative* item matching the arguments, or None if none found.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB in case the item is not found in memory.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **name** (*str*) – The alternative name.

Returns
PublicItem or None

get_scenario_item(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns a *scenario* item matching the arguments, or None if none found.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB in case the item is not found in memory.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **name** (*str*) – The scenario name.

Returns
PublicItem or None

get_scenario_alternative_item(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns a *scenario_alternative* item matching the arguments, or None if none found.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB in case the item is not found in memory.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.

- **alternative_name** (*str*) – The alternative name.
- **scenario_name** (*str*) – The scenario name.
- **rank** (*int*) – The rank - higher has precedence.

Returns
PublicItem or None

get_entity_class_item(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns an *entity_class* item matching the arguments, or None if none found.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB in case the item is not found in memory.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **name** (*str*) – The class name.

Returns
PublicItem or None

get_superclass_subclass_item(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns a *superclass_subclass* item matching the arguments, or None if none found.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB in case the item is not found in memory.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **subclass_name** (*str*) – The subclass name.

Returns
PublicItem or None

get_entity_item(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns an *entity* item matching the arguments, or None if none found.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB in case the item is not found in memory.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **entity_class_name** (*str*) – The entity class name.
- **name** (*str*) – The entity name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional,or the element names if it is multi-dimensional.

Returns
PublicItem or None

get_entity_group_item(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns an *entity_group* item matching the arguments, or None if none found.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB in case the item is not found in memory.

- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **entity_class_name** (*str*) – The entity class name.
- **group_name** (*str*) – The group entity name.
- **member_name** (*str*) – The member entity name.

Returns
PublicItem or None

get_entity_alternative_item(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns an *entity_alternative* item matching the arguments, or None if none found.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB in case the item is not found in memory.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **entity_class_name** (*str*) – The entity class name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
- **alternative_name** (*str*) – The alternative name.

Returns
PublicItem or None

get_parameter_value_list_item(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns a *parameter_value_list* item matching the arguments, or None if none found.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB in case the item is not found in memory.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **name** (*str*) – The parameter value list name.

Returns
PublicItem or None

get_list_value_item(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns a *list_value* item matching the arguments, or None if none found.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB in case the item is not found in memory.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **parameter_value_list_name** (*str*) – The parameter value list name.
- **index** (*int, optional*) – The value index.

Returns
PublicItem or None

get_parameter_definition_item(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns a *parameter_definition* item matching the arguments, or None if none found.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB in case the item is not found in memory.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **entity_class_name** (*str*) – The entity class name.
- **name** (*str*) – The parameter name.

Returns

PublicItem or None

get_parameter_value_item(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns a *parameter_value* item matching the arguments, or None if none found.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB in case the item is not found in memory.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **entity_class_name** (*str*) – The entity class name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
- **parameter_definition_name** (*str*) – The parameter name.
- **alternative_name** (*str, optional*) – The alternative name - defaults to 'Base'.

Returns

PublicItem or None

get_metadata_item(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns a *metadata* item matching the arguments, or None if none found.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB in case the item is not found in memory.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **name** (*str*) – The metadata entry name.
- **value** (*str*) – The metadata entry value.

Returns

PublicItem or None

get_entity_metadata_item(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns an *entity_metadata* item matching the arguments, or None if none found.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB in case the item is not found in memory.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.

- **metadata_name** (*str*) – The metadata entry name.
- **entity_class_name** (*str*) – The entity class name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
- **metadata_value** (*str*) – The metadata entry value.

Returns
PublicItem or None

get_parameter_value_metadata_item(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns a *parameter_value_metadata* item matching the arguments, or None if none found.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB in case the item is not found in memory.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **entity_class_name** (*str*) – The entity class name.
- **metadata_name** (*str*) – The metadata entry name.
- **parameter_definition_name** (*str*) – The parameter name.
- **alternative_name** (*str*) – The alternative name.
- **metadata_value** (*str*) – The metadata entry value.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.

Returns
PublicItem or None

get_alternative_items(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns all alternative items.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB before returning the items.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **name** (*str*) – The alternative name.

Returns
The items.

Return type
list(PublicItem)

get_scenario_items(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns all scenario items.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB before returning the items.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **name** (*str*) – The scenario name.

Returns
The items.

Return type
list(PublicItem)

`get_scenario_alternative_items`(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns all scenario_alternative items.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB before returning the items.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **alternative_name** (*str*) – The alternative name.
- **scenario_name** (*str*) – The scenario name.
- **rank** (*int*) – The rank - higher has precedence.

Returns
The items.

Return type
list(PublicItem)

`get_entity_class_items`(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns all entity_class items.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB before returning the items.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **name** (*str*) – The class name.

Returns
The items.

Return type
list(PublicItem)

`get_superclass_subclass_items`(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns all superclass_subclass items.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB before returning the items.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **subclass_name** (*str*) – The subclass name.

Returns
The items.

Return type
list(PublicItem)

`get_entity_items`(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns all entity items.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB before returning the items.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **entity_class_name** (*str*) – The entity class name.
- **name** (*str*) – The entity name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional,or the element names if it is multi-dimensional.

Returns
The items.

Return type
list(PublicItem)

get_entity_group_items(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns all entity_group items.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB before returning the items.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **entity_class_name** (*str*) – The entity class name.
- **group_name** (*str*) – The group entity name.
- **member_name** (*str*) – The member entity name.

Returns
The items.

Return type
list(PublicItem)

get_entity_alternative_items(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns all entity_alternative items.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB before returning the items.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **entity_class_name** (*str*) – The entity class name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
- **alternative_name** (*str*) – The alternative name.

Returns
The items.

Return type
list(PublicItem)

get_parameter_value_list_items(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns all parameter_value_list items.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB before returning the items.

- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **name** (*str*) – The parameter value list name.

Returns
The items.

Return type
list(PublicItem)

get_list_value_items(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns all list_value items.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB before returning the items.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **parameter_value_list_name** (*str*) – The parameter value list name.
- **index** (*int, optional*) – The value index.

Returns
The items.

Return type
list(PublicItem)

get_parameter_definition_items(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns all parameter_definition items.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB before returning the items.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **entity_class_name** (*str*) – The entity class name.
- **name** (*str*) – The parameter name.

Returns
The items.

Return type
list(PublicItem)

get_parameter_value_items(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns all parameter_value items.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB before returning the items.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **entity_class_name** (*str*) – The entity class name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
- **parameter_definition_name** (*str*) – The parameter name.
- **alternative_name** (*str, optional*) – The alternative name - defaults to 'Base'.

Returns
The items.

Return type
list(PublicItem)

`get_metadata_items`(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns all metadata items.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB before returning the items.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **name** (*str*) – The metadata entry name.
- **value** (*str*) – The metadata entry value.

Returns
The items.

Return type
list(PublicItem)

`get_entity_metadata_items`(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns all entity_metadata items.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB before returning the items.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **metadata_name** (*str*) – The metadata entry name.
- **entity_class_name** (*str*) – The entity class name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
- **metadata_value** (*str*) – The metadata entry value.

Returns
The items.

Return type
list(PublicItem)

`get_parameter_value_metadata_items`(*fetch=True*, *skip_removed=True*, ***kwargs*)

Finds and returns all parameter_value_metadata items.

Parameters

- **fetch** (*bool, optional*) – Whether to fetch the DB before returning the items.
- **skip_removed** (*bool, optional*) – Whether to ignore removed items.
- **entity_class_name** (*str*) – The entity class name.
- **metadata_name** (*str*) – The metadata entry name.
- **parameter_definition_name** (*str*) – The parameter name.
- **alternative_name** (*str*) – The alternative name.
- **metadata_value** (*str*) – The metadata entry value.

- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.

Returns
The items.

Return type
list(PublicItem)

add_alternative_item(*check=True*, ***kwargs*)

Adds an *alternative* item to the in-memory mapping.

Parameters

- **name** (*str*) – The alternative name.
- **description** (*str, optional*) – The alternative description.

Returns
The added item and any errors.

Return type
tuple(PublicItem or None, str)

add_scenario_item(*check=True*, ***kwargs*)

Adds a *scenario* item to the in-memory mapping.

Parameters

- **name** (*str*) – The scenario name.
- **description** (*str, optional*) – The scenario description.
- **active** (*bool, optional*) – Not in use at the moment.

Returns
The added item and any errors.

Return type
tuple(PublicItem or None, str)

add_scenario_alternative_item(*check=True*, ***kwargs*)

Adds a *scenario_alternative* item to the in-memory mapping.

Parameters

- **scenario_name** (*str*) – The scenario name.
- **alternative_name** (*str*) – The alternative name.
- **rank** (*int*) – The rank - higher has precedence.

Returns
The added item and any errors.

Return type
tuple(PublicItem or None, str)

add_entity_class_item(*check=True*, ***kwargs*)

Adds an *entity_class* item to the in-memory mapping.

Parameters

- **name** (*str*) – The class name.

- **dimension_name_list** (*tuple, optional*) – The dimension names for a multi-dimensional class.
- **description** (*str, optional*) – The class description.
- **display_icon** (*int, optional*) – An integer representing an icon within your application.
- **display_order** (*int, optional*) – Not in use at the moment.
- **hidden** (*int, optional*) – Not in use at the moment.
- **active_by_default** (*bool, optional*) – Default activity for the entity alternatives of the class.

Returns
The added item and any errors.

Return type
tuple(PublicItem or None, str)

add_superclass_subclass_item(*check=True*, ***kwargs*)

Adds a *superclass_subclass* item to the in-memory mapping.

Parameters

- **superclass_name** (*str*) – The superclass name.
- **subclass_name** (*str*) – The subclass name.

Returns
The added item and any errors.

Return type
tuple(PublicItem or None, str)

add_entity_item(*check=True*, ***kwargs*)

Adds an *entity* item to the in-memory mapping.

Parameters

- **entity_class_name** (*str*) – The entity class name.
- **name** (*str*) – The entity name.
- **element_name_list** (*tuple*) – The element names if the entity is multi-dimensional.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional,or the element names if it is multi-dimensional.
- **description** (*str, optional*) – The entity description.

Returns
The added item and any errors.

Return type
tuple(PublicItem or None, str)

add_entity_group_item(*check=True*, ***kwargs*)

Adds an *entity_group* item to the in-memory mapping.

Parameters

- **entity_class_name** (*str*) – The entity class name.
- **group_name** (*str*) – The group entity name.

- **member_name** (*str*) – The member entity name.

Returns
The added item and any errors.

Return type
tuple(PublicItem or None, str)

add_entity_alternative_item(*check=True*, ***kwargs*)

Adds an *entity_alternative* item to the in-memory mapping.

Parameters

- **entity_class_name** (*str*) – The entity class name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
- **alternative_name** (*str*) – The alternative name.
- **active** (*bool, optional*) – Whether the entity is active in the alternative - defaults to True.

Returns
The added item and any errors.

Return type
tuple(PublicItem or None, str)

add_parameter_value_list_item(*check=True*, ***kwargs*)

Adds a *parameter_value_list* item to the in-memory mapping.

Parameters
name (*str*) – The parameter value list name.

Returns
The added item and any errors.

Return type
tuple(PublicItem or None, str)

add_list_value_item(*check=True*, ***kwargs*)

Adds a *list_value* item to the in-memory mapping.

Parameters

- **parameter_value_list_name** (*str*) – The parameter value list name.
- **value** (*bytes*) – The value.
- **type** (*str, optional*) – The value type.
- **index** (*int, optional*) – The value index.

Returns
The added item and any errors.

Return type
tuple(PublicItem or None, str)

add_parameter_definition_item(*check=True*, ***kwargs*)

Adds a *parameter_definition* item to the in-memory mapping.

Parameters

- **entity_class_name** (*str*) – The entity class name.
- **name** (*str*) – The parameter name.
- **default_value** (*bytes, optional*) – The default value.
- **default_type** (*str, optional*) – The default value type.
- **parameter_value_list_name** (*str, optional*) – The parameter value list name if any.
- **description** (*str, optional*) – The parameter description.

Returns
The added item and any errors.

Return type
tuple(PublicItem or None, str)

add_parameter_value_item(*check=True*, ***kwargs*)

Adds a *parameter_value* item to the in-memory mapping.

Parameters

- **entity_class_name** (*str*) – The entity class name.
- **parameter_definition_name** (*str*) – The parameter name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
- **value** (*bytes*) – The value.
- **type** (*str, optional*) – The value type.
- **alternative_name** (*str, optional*) – The alternative name - defaults to 'Base'.

Returns
The added item and any errors.

Return type
tuple(PublicItem or None, str)

add_metadata_item(*check=True*, ***kwargs*)

Adds a *metadata* item to the in-memory mapping.

Parameters

- **name** (*str*) – The metadata entry name.
- **value** (*str*) – The metadata entry value.

Returns
The added item and any errors.

Return type
tuple(PublicItem or None, str)

add_entity_metadata_item(*check=True*, ***kwargs*)

Adds an *entity_metadata* item to the in-memory mapping.

Parameters

- **entity_class_name** (*str*) – The entity class name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.

- **metadata_name** (*str*) – The metadata entry name.
- **metadata_value** (*str*) – The metadata entry value.

Returns
The added item and any errors.

Return type
tuple(PublicItem or None, str)

add_parameter_value_metadata_item(*check=True*, ***kwargs*)

Adds a *parameter_value_metadata* item to the in-memory mapping.

Parameters

- **entity_class_name** (*str*) – The entity class name.
- **parameter_definition_name** (*str*) – The parameter name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
- **alternative_name** (*str*) – The alternative name.
- **metadata_name** (*str*) – The metadata entry name.
- **metadata_value** (*str*) – The metadata entry value.

Returns
The added item and any errors.

Return type
tuple(PublicItem or None, str)

update_alternative_item(*check=True*, ***kwargs*)

Updates an *alternative* item in the in-memory mapping.

Parameters

- **id** (*int*) – The id of the item to update.
- **name** (*str*) – The alternative name.
- **description** (*str, optional*) – The alternative description.

Returns
The updated item and any errors.

Return type
tuple(PublicItem or None, str)

update_scenario_item(*check=True*, ***kwargs*)

Updates a *scenario* item in the in-memory mapping.

Parameters

- **id** (*int*) – The id of the item to update.
- **name** (*str*) – The scenario name.
- **description** (*str, optional*) – The scenario description.
- **active** (*bool, optional*) – Not in use at the moment.

Returns
The updated item and any errors.

Return type
tuple(PublicItem or None, str)

update_scenario_alternative_item(*check=True*, ***kwargs*)

Updates a *scenario_alternative* item in the in-memory mapping.

Parameters

- **id** (*int*) – The id of the item to update.
- **scenario_name** (*str*) – The scenario name.
- **alternative_name** (*str*) – The alternative name.
- **rank** (*int*) – The rank - higher has precedence.

Returns
The updated item and any errors.

Return type
tuple(PublicItem or None, str)

update_entity_class_item(*check=True*, ***kwargs*)

Updates an *entity_class* item in the in-memory mapping.

Parameters

- **id** (*int*) – The id of the item to update.
- **name** (*str*) – The class name.
- **dimension_name_list** (*tuple, optional*) – The dimension names for a multi-dimensional class.
- **description** (*str, optional*) – The class description.
- **display_icon** (*int, optional*) – An integer representing an icon within your application.
- **display_order** (*int, optional*) – Not in use at the moment.
- **hidden** (*int, optional*) – Not in use at the moment.
- **active_by_default** (*bool, optional*) – Default activity for the entity alternatives of the class.

Returns
The updated item and any errors.

Return type
tuple(PublicItem or None, str)

update_superclass_subclass_item(*check=True*, ***kwargs*)

Updates a *superclass_subclass* item in the in-memory mapping.

Parameters

- **id** (*int*) – The id of the item to update.
- **superclass_name** (*str*) – The superclass name.
- **subclass_name** (*str*) – The subclass name.

Returns
The updated item and any errors.

Return type
tuple(PublicItem or None, str)

update_entity_item(*check=True*, ***kwargs*)

Updates an *entity* item in the in-memory mapping.

Parameters

- **id** (*int*) – The id of the item to update.
- **entity_class_name** (*str*) – The entity class name.
- **name** (*str*) – The entity name.
- **element_name_list** (*tuple*) – The element names if the entity is multi-dimensional.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional,or the element names if it is multi-dimensional.
- **description** (*str, optional*) – The entity description.

Returns
The updated item and any errors.

Return type
tuple(PublicItem or None, str)

update_entity_group_item(*check=True*, ***kwargs*)

Updates an *entity_group* item in the in-memory mapping.

Parameters

- **id** (*int*) – The id of the item to update.
- **entity_class_name** (*str*) – The entity class name.
- **group_name** (*str*) – The group entity name.
- **member_name** (*str*) – The member entity name.

Returns
The updated item and any errors.

Return type
tuple(PublicItem or None, str)

update_entity_alternative_item(*check=True*, ***kwargs*)

Updates an *entity_alternative* item in the in-memory mapping.

Parameters

- **id** (*int*) – The id of the item to update.
- **entity_class_name** (*str*) – The entity class name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
- **alternative_name** (*str*) – The alternative name.
- **active** (*bool, optional*) – Whether the entity is active in the alternative - defaults to True.

Returns
The updated item and any errors.

Return type
tuple(PublicItem or None, str)

update_parameter_value_list_item(*check=True*, ***kwargs*)

Updates a *parameter_value_list* item in the in-memory mapping.

Parameters

- **id** (*int*) – The id of the item to update.
- **name** (*str*) – The parameter value list name.

Returns
The updated item and any errors.

Return type
tuple(PublicItem or None, str)

update_list_value_item(*check=True*, ***kwargs*)

Updates a *list_value* item in the in-memory mapping.

Parameters

- **id** (*int*) – The id of the item to update.
- **parameter_value_list_name** (*str*) – The parameter value list name.
- **value** (*bytes*) – The value.
- **type** (*str, optional*) – The value type.
- **index** (*int, optional*) – The value index.

Returns
The updated item and any errors.

Return type
tuple(PublicItem or None, str)

update_parameter_definition_item(*check=True*, ***kwargs*)

Updates a *parameter_definition* item in the in-memory mapping.

Parameters

- **id** (*int*) – The id of the item to update.
- **entity_class_name** (*str*) – The entity class name.
- **name** (*str*) – The parameter name.
- **default_value** (*bytes, optional*) – The default value.
- **default_type** (*str, optional*) – The default value type.
- **parameter_value_list_name** (*str, optional*) – The parameter value list name if any.
- **description** (*str, optional*) – The parameter description.

Returns
The updated item and any errors.

Return type
tuple(PublicItem or None, str)

update_parameter_value_item(*check=True*, ***kwargs*)

Updates a *parameter_value* item in the in-memory mapping.

Parameters

- **id** (*int*) – The id of the item to update.
- **entity_class_name** (*str*) – The entity class name.
- **parameter_definition_name** (*str*) – The parameter name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
- **value** (*bytes*) – The value.
- **type** (*str, optional*) – The value type.
- **alternative_name** (*str, optional*) – The alternative name - defaults to 'Base'.

Returns
The updated item and any errors.

Return type
tuple(PublicItem or None, str)

update_metadata_item(*check=True*, ***kwargs*)

Updates a *metadata* item in the in-memory mapping.

Parameters

- **id** (*int*) – The id of the item to update.
- **name** (*str*) – The metadata entry name.
- **value** (*str*) – The metadata entry value.

Returns
The updated item and any errors.

Return type
tuple(PublicItem or None, str)

update_entity_metadata_item(*check=True*, ***kwargs*)

Updates an *entity_metadata* item in the in-memory mapping.

Parameters

- **id** (*int*) – The id of the item to update.
- **entity_class_name** (*str*) – The entity class name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
- **metadata_name** (*str*) – The metadata entry name.
- **metadata_value** (*str*) – The metadata entry value.

Returns
The updated item and any errors.

Return type
tuple(PublicItem or None, str)

update_parameter_value_metadata_item(*check=True*, ***kwargs*)

Updates a *parameter_value_metadata* item in the in-memory mapping.

Parameters

- **id** (*int*) – The id of the item to update.
- **entity_class_name** (*str*) – The entity class name.
- **parameter_definition_name** (*str*) – The parameter name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
- **alternative_name** (*str*) – The alternative name.
- **metadata_name** (*str*) – The metadata entry name.
- **metadata_value** (*str*) – The metadata entry value.

Returns
The updated item and any errors.

Return type
tuple(PublicItem or None, str)

add_update_alternative_item(*check=True*, ***kwargs*)

Adds an *alternative* item to the in-memory mapping if it doesn't exist; otherwise updates the current one.

Parameters

- **name** (*str*) – The alternative name.
- **description** (*str, optional*) – The alternative description.

Returns

The added item if any,
the updated item if any, and any errors.

Return type
tuple(PublicItem or None, PublicItem or None, str)

add_update_scenario_item(*check=True*, ***kwargs*)

Adds a *scenario* item to the in-memory mapping if it doesn't exist; otherwise updates the current one.

Parameters

- **name** (*str*) – The scenario name.
- **description** (*str, optional*) – The scenario description.
- **active** (*bool, optional*) – Not in use at the moment.

Returns

The added item if any,
the updated item if any, and any errors.

Return type
tuple(PublicItem or None, PublicItem or None, str)

add_update_scenario_alternative_item(*check=True*, ***kwargs*)

Adds a *scenario_alternative* item to the in-memory mapping if it doesn't exist; otherwise updates the current one.

Parameters

- **scenario_name** (*str*) – The scenario name.
- **alternative_name** (*str*) – The alternative name.
- **rank** (*int*) – The rank - higher has precedence.

Returns

The added item if any,
the updated item if any, and any errors.

Return type
tuple(PublicItem or None, PublicItem or None, str)

add_update_entity_class_item(*check=True*, ***kwargs*)

Adds an *entity_class* item to the in-memory mapping if it doesn't exist; otherwise updates the current one.

Parameters

- **name** (*str*) – The class name.
- **dimension_name_list** (*tuple, optional*) – The dimension names for a multi-dimensional class.
- **description** (*str, optional*) – The class description.
- **display_icon** (*int, optional*) – An integer representing an icon within your application.
- **display_order** (*int, optional*) – Not in use at the moment.
- **hidden** (*int, optional*) – Not in use at the moment.
- **active_by_default** (*bool, optional*) – Default activity for the entity alternatives of the class.

Returns

The added item if any,
the updated item if any, and any errors.

Return type
tuple(PublicItem or None, PublicItem or None, str)

add_update_superclass_subclass_item(*check=True*, ***kwargs*)

Adds a *superclass_subclass* item to the in-memory mapping if it doesn't exist; otherwise updates the current one.

Parameters

- **superclass_name** (*str*) – The superclass name.
- **subclass_name** (*str*) – The subclass name.

Returns

The added item if any,
the updated item if any, and any errors.

Return type
tuple(PublicItem or None, PublicItem or None, str)

add_update_entity_item(*check=True*, ***kwargs*)

Adds an *entity* item to the in-memory mapping if it doesn't exist; otherwise updates the current one.

Parameters

- **entity_class_name** (*str*) – The entity class name.
- **name** (*str*) – The entity name.
- **element_name_list** (*tuple*) – The element names if the entity is multi-dimensional.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional,or the element names if it is multi-dimensional.
- **description** (*str, optional*) – The entity description.

Returns

The added item if any,
the updated item if any, and any errors.

Return type
tuple(PublicItem or None, PublicItem or None, str)

add_update_entity_group_item(*check=True*, ***kwargs*)

Adds an *entity_group* item to the in-memory mapping if it doesn't exist; otherwise updates the current one.

Parameters

- **entity_class_name** (*str*) – The entity class name.
- **group_name** (*str*) – The group entity name.
- **member_name** (*str*) – The member entity name.

Returns

The added item if any,
the updated item if any, and any errors.

Return type
tuple(PublicItem or None, PublicItem or None, str)

add_update_entity_alternative_item(*check=True*, ***kwargs*)

Adds an *entity_alternative* item to the in-memory mapping if it doesn't exist; otherwise updates the current one.

Parameters

- **entity_class_name** (*str*) – The entity class name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
- **alternative_name** (*str*) – The alternative name.
- **active** (*bool, optional*) – Whether the entity is active in the alternative - defaults to True.

Returns

The added item if any,
the updated item if any, and any errors.

Return type
tuple(PublicItem or None, PublicItem or None, str)

add_update_parameter_value_list_item(*check=True*, ***kwargs*)

Adds a *parameter_value_list* item to the in-memory mapping if it doesn't exist; otherwise updates the current one.

Parameters
name (*str*) – The parameter value list name.

Returns

The added item if any,
the updated item if any, and any errors.

Return type
tuple(PublicItem or None, PublicItem or None, str)

add_update_list_value_item(*check=True*, ***kwargs*)

Adds a *list_value* item to the in-memory mapping if it doesn't exist; otherwise updates the current one.

Parameters

- **parameter_value_list_name** (*str*) – The parameter value list name.
- **value** (*bytes*) – The value.
- **type** (*str, optional*) – The value type.
- **index** (*int, optional*) – The value index.

Returns

The added item if any,
the updated item if any, and any errors.

Return type
tuple(PublicItem or None, PublicItem or None, str)

add_update_parameter_definition_item(*check=True*, ***kwargs*)

Adds a *parameter_definition* item to the in-memory mapping if it doesn't exist; otherwise updates the current one.

Parameters

- **entity_class_name** (*str*) – The entity class name.
- **name** (*str*) – The parameter name.
- **default_value** (*bytes, optional*) – The default value.
- **default_type** (*str, optional*) – The default value type.
- **parameter_value_list_name** (*str, optional*) – The parameter value list name if any.
- **description** (*str, optional*) – The parameter description.

Returns

The added item if any,
the updated item if any, and any errors.

Return type
tuple(PublicItem or None, PublicItem or None, str)

add_update_parameter_value_item(*check=True*, ***kwargs*)

Adds a *parameter_value* item to the in-memory mapping if it doesn't exist; otherwise updates the current one.

Parameters

- **entity_class_name** (*str*) – The entity class name.
- **parameter_definition_name** (*str*) – The parameter name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
- **value** (*bytes*) – The value.
- **type** (*str, optional*) – The value type.
- **alternative_name** (*str, optional*) – The alternative name - defaults to 'Base'.

Returns

The added item if any,
the updated item if any, and any errors.

Return type
tuple(PublicItem or None, PublicItem or None, str)

add_update_metadata_item(*check=True*, ***kwargs*)

Adds a *metadata* item to the in-memory mapping if it doesn't exist; otherwise updates the current one.

Parameters

- **name** (*str*) – The metadata entry name.
- **value** (*str*) – The metadata entry value.

Returns

The added item if any,
the updated item if any, and any errors.

Return type
tuple(PublicItem or None, PublicItem or None, str)

add_update_entity_metadata_item(*check=True*, ***kwargs*)

Adds an *entity_metadata* item to the in-memory mapping if it doesn't exist; otherwise updates the current one.

Parameters

- **entity_class_name** (*str*) – The entity class name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
- **metadata_name** (*str*) – The metadata entry name.
- **metadata_value** (*str*) – The metadata entry value.

Returns

The added item if any,
the updated item if any, and any errors.

Return type
tuple(PublicItem or None, PublicItem or None, str)

add_update_parameter_value_metadata_item(*check=True*, ***kwargs*)

Adds a *parameter_value_metadata* item to the in-memory mapping if it doesn't exist; otherwise updates the current one.

Parameters

- **entity_class_name** (*str*) – The entity class name.
- **parameter_definition_name** (*str*) – The parameter name.
- **entity_byname** (*tuple*) – A tuple with the entity name as single element if the entity is zero-dimensional, or the element names if the entity is multi-dimensional.
- **alternative_name** (*str*) – The alternative name.
- **metadata_name** (*str*) – The metadata entry name.
- **metadata_value** (*str*) – The metadata entry value.

Returns

The added item if any,
the updated item if any, and any errors.

Return type
tuple(PublicItem or None, PublicItem or None, str)

remove_alternative_item(*id*)

Removes a *alternative* item from the in-memory mapping.

Parameters
id (*int*) – the id of the item to remove.

Returns
The removed item if any.

Return type
tuple(PublicItem or None, str)

remove_scenario_item(*id*)

Removes a *scenario* item from the in-memory mapping.

Parameters
id (*int*) – the id of the item to remove.

Returns
The removed item if any.

Return type
tuple(PublicItem or None, str)

remove_scenario_alternative_item(*id*)

Removes a *scenario_alternative* item from the in-memory mapping.

Parameters
id (*int*) – the id of the item to remove.

Returns
The removed item if any.

Return type
tuple(PublicItem or None, str)

`remove_entity_class_item`(*id*)

Removes a *entity_class* item from the in-memory mapping.

Parameters
id (*int*) – the id of the item to remove.

Returns
The removed item if any.

Return type
tuple(`PublicItem` or None, str)

`remove_superclass_subclass_item`(*id*)

Removes a *superclass_subclass* item from the in-memory mapping.

Parameters
id (*int*) – the id of the item to remove.

Returns
The removed item if any.

Return type
tuple(`PublicItem` or None, str)

`remove_entity_item`(*id*)

Removes a *entity* item from the in-memory mapping.

Parameters
id (*int*) – the id of the item to remove.

Returns
The removed item if any.

Return type
tuple(`PublicItem` or None, str)

`remove_entity_group_item`(*id*)

Removes a *entity_group* item from the in-memory mapping.

Parameters
id (*int*) – the id of the item to remove.

Returns
The removed item if any.

Return type
tuple(`PublicItem` or None, str)

`remove_entity_alternative_item`(*id*)

Removes a *entity_alternative* item from the in-memory mapping.

Parameters
id (*int*) – the id of the item to remove.

Returns
The removed item if any.

Return type
tuple(`PublicItem` or None, str)

remove_parameter_value_list_item(*id*)

Removes a *parameter_value_list* item from the in-memory mapping.

Parameters
id (*int*) – the id of the item to remove.

Returns
The removed item if any.

Return type
tuple(PublicItem or None, str)

remove_list_value_item(*id*)

Removes a *list_value* item from the in-memory mapping.

Parameters
id (*int*) – the id of the item to remove.

Returns
The removed item if any.

Return type
tuple(PublicItem or None, str)

remove_parameter_definition_item(*id*)

Removes a *parameter_definition* item from the in-memory mapping.

Parameters
id (*int*) – the id of the item to remove.

Returns
The removed item if any.

Return type
tuple(PublicItem or None, str)

remove_parameter_value_item(*id*)

Removes a *parameter_value* item from the in-memory mapping.

Parameters
id (*int*) – the id of the item to remove.

Returns
The removed item if any.

Return type
tuple(PublicItem or None, str)

remove_metadata_item(*id*)

Removes a *metadata* item from the in-memory mapping.

Parameters
id (*int*) – the id of the item to remove.

Returns
The removed item if any.

Return type
tuple(PublicItem or None, str)

`remove_entity_metadata_item`(*id*)

Removes a *entity_metadata* item from the in-memory mapping.

Parameters
id (*int*) – the id of the item to remove.

Returns
The removed item if any.

Return type
tuple(`PublicItem` or None, str)

`remove_parameter_value_metadata_item`(*id*)

Removes a *parameter_value_metadata* item from the in-memory mapping.

Parameters
id (*int*) – the id of the item to remove.

Returns
The removed item if any.

Return type
tuple(`PublicItem` or None, str)

`restore_alternative_item`(*id*)

Restores a previously removed *alternative* item into the in-memory mapping.

Parameters
id (*int*) – the id of the item to restore.

Returns
The restored item if any.

Return type
tuple(`PublicItem` or None, str)

`restore_scenario_item`(*id*)

Restores a previously removed *scenario* item into the in-memory mapping.

Parameters
id (*int*) – the id of the item to restore.

Returns
The restored item if any.

Return type
tuple(`PublicItem` or None, str)

`restore_scenario_alternative_item`(*id*)

Restores a previously removed *scenario_alternative* item into the in-memory mapping.

Parameters
id (*int*) – the id of the item to restore.

Returns
The restored item if any.

Return type
tuple(`PublicItem` or None, str)

`restore_entity_class_item`(*id*)

Restores a previously removed *entity_class* item into the in-memory mapping.

Parameters
id (*int*) – the id of the item to restore.

Returns
The restored item if any.

Return type
tuple(`PublicItem` or None, str)

`restore_superclass_subclass_item`(*id*)

Restores a previously removed *superclass_subclass* item into the in-memory mapping.

Parameters
id (*int*) – the id of the item to restore.

Returns
The restored item if any.

Return type
tuple(`PublicItem` or None, str)

`restore_entity_item`(*id*)

Restores a previously removed *entity* item into the in-memory mapping.

Parameters
id (*int*) – the id of the item to restore.

Returns
The restored item if any.

Return type
tuple(`PublicItem` or None, str)

`restore_entity_group_item`(*id*)

Restores a previously removed *entity_group* item into the in-memory mapping.

Parameters
id (*int*) – the id of the item to restore.

Returns
The restored item if any.

Return type
tuple(`PublicItem` or None, str)

`restore_entity_alternative_item`(*id*)

Restores a previously removed *entity_alternative* item into the in-memory mapping.

Parameters
id (*int*) – the id of the item to restore.

Returns
The restored item if any.

Return type
tuple(`PublicItem` or None, str)

`restore_parameter_value_list_item`(*id*)

Restores a previously removed *parameter_value_list* item into the in-memory mapping.

Parameters
id (*int*) – the id of the item to restore.

Returns
The restored item if any.

Return type
tuple(`PublicItem` or None, str)

`restore_list_value_item`(*id*)

Restores a previously removed *list_value* item into the in-memory mapping.

Parameters
id (*int*) – the id of the item to restore.

Returns
The restored item if any.

Return type
tuple(`PublicItem` or None, str)

`restore_parameter_definition_item`(*id*)

Restores a previously removed *parameter_definition* item into the in-memory mapping.

Parameters
id (*int*) – the id of the item to restore.

Returns
The restored item if any.

Return type
tuple(`PublicItem` or None, str)

`restore_parameter_value_item`(*id*)

Restores a previously removed *parameter_value* item into the in-memory mapping.

Parameters
id (*int*) – the id of the item to restore.

Returns
The restored item if any.

Return type
tuple(`PublicItem` or None, str)

`restore_metadata_item`(*id*)

Restores a previously removed *metadata* item into the in-memory mapping.

Parameters
id (*int*) – the id of the item to restore.

Returns
The restored item if any.

Return type
tuple(`PublicItem` or None, str)

restore_entity_metadata_item(*id*)

Restores a previously removed *entity_metadata* item into the in-memory mapping.

Parameters
id (*int*) – the id of the item to restore.

Returns
The restored item if any.

Return type
tuple(PublicItem or None, str)

restore_parameter_value_metadata_item(*id*)

Restores a previously removed *parameter_value_metadata* item into the in-memory mapping.

Parameters
id (*int*) – the id of the item to restore.

Returns
The restored item if any.

Return type
tuple(PublicItem or None, str)

spinedb_api.graph_layout_generator

This module defines the *GraphLayoutGenerator* class.

Module Contents

Classes

GraphLayoutGenerator	A class to build an optimised layout for an undirected graph.

class spinedb_api.graph_layout_generator.**GraphLayoutGenerator**(*vertex_count*, *src_inds=()*, *dst_inds=()*, *spread=0*, *heavy_positions=None*, *max_iters=12*, *weight_exp=-2*, *is_stopped=lambda : ...*, *preview_available=lambda x, y: ...*, *layout_available=lambda x, y: ...*, *layout_progressed=lambda iter: ...*)

A class to build an optimised layout for an undirected graph. This can help visualizing the Spine data structure of multi-dimensional entities.

Parameters

- **vertex_count** (*int*) – The number of vertices in the graph. Graph vertices will have indices 0, 1, 2, ...
- **src_inds** (*tuple, optional*) – The indices of the source vertices of each edge.
- **dst_inds** (*tuple, optional*) – The indices of the destination vertices of each edge.

- **spread** (*int, optional*) – the ideal edge length.
- **heavy_positions** (*dict, optional*) – a dictionary mapping vertex indices to another dictionary with keys "x" and "y" specifying the position it should have in the generated layout.
- **max_iters** (*int, optional*) – the maximum numbers of iterations of the layout generation algorithm.
- **weight_exp** (*int, optional*) – The exponential decay rate of attraction between vertices. The higher this number, the lesser the attraction between distant vertices.
- **is_stopped** (*function, optional*) – A function to call without arguments, that returns a boolean indicating whether the layout generation process needs to be stopped.
- **preview_available** (*function, optional*) – A function to call after every iteration with two lists, x and y, representing the current layout.
- **layout_available** (*function, optional*) – A function to call after the last iteration with two lists, x and y, representing the final layout.
- **layout_progressed** (*function, optional*) – A function to call after each iteration with the current iteration number.

compute_layout()

Computes the layout using VSGD-MS and returns x and y coordinates for each vertex in the graph.

Returns
x and y coordinates

Return type
tuple(list,list)

spinedb_api.import_functions

Functions for importing data into a Spine database in a standard format. This functionaly is equivalent to the one provided by *DatabaseMapping.add_update_item()*, but the syntax is a little more compact.

Module Contents

Functions

import_data(db_map[, unparse_value, on_conflict])	Imports data into a Spine database using a standard format.
get_data_for_import(db_map[, unparse_value, ...])	Yields data to import into a Spine DB.
import_superclass_subclasses(db_map, data)	Imports superclass_subclasses into a Spine database using a standard format.
import_entity_classes(db_map, data)	Imports entity classes into a Spine database using a standard format.
import_entities(db_map, data)	Imports entities into a Spine database using a standard format.
import_entity_alternatives(db_map, data)	Imports entity alternatives into a Spine database using a standard format.
import_entity_groups(db_map, data)	Imports entity groups into a Spine database using a standard format.
import_parameter_definitions(db_map, data[, unparse_value])	Imports parameter definitions into a Spine database using a standard format.
import_parameter_values(db_map, data[, unparse_value, ...])	Imports parameter values into a Spine database using a standard format.
import_alternatives(db_map, data)	Imports alternatives into a Spine database using a standard format.
import_scenarios(db_map, data)	Imports scenarios into a Spine database using a standard format.
import_scenario_alternatives(db_map, data)	Imports scenario alternatives into a Spine database using a standard format.
import_parameter_value_lists(db_map, data[, unparse_value])	Imports parameter value lists into a Spine database using a standard format.
import_metadata(db_map, data)	Imports metadata into a Spine database using a standard format.

spinedb_api.import_functions.**import_data**(*db_map*, *unparse_value=to_database*, *on_conflict='merge'*, ***kwargs*)

Imports data into a Spine database using a standard format.

Example:

```
entity_classes = [
    ('example_class', ()), ('other_class', ()), ('multi_d_class', ('example_class',
↪'other_class'))
]
alternatives = [('example_alternative', 'An example')]
scenarios = [('example_scenario', 'An example')]
scenario_alternatives = [
    ('example_scenario', 'example_alternative'), ('example_scenario', 'Base',
↪'example_alternative')
]
parameter_value_lists = [("example_list", "value1"), ("example_list", "value2")]
parameter_definitions = [('example_class', 'example_parameter'), ('multi_d_class',
↪'other_parameter')]
entities = [
```

(continues on next page)

(continued from previous page)

```
    ('example_class', 'example_entity'),
    ('example_class', 'example_group'),
    ('example_class', 'example_member'),
    ('other_class', 'other_entity'),
    ('multi_d_class', ('example_entity', 'other_entity')),
]
entity_groups = [
    ('example_class', 'example_group', 'example_member'),
    ('example_class', 'example_group', 'example_entity'),
]
parameter_values = [
    ('example_object_class', 'example_entity', 'example_parameter', 3.14),
    ('multi_d_class', ('example_entity', 'other_entity'), 'rel_parameter', 2.718),
]
entity_alternatives = [
    ('example_class', 'example_entity', "example_alternative", True),
    ('example_class', 'example_entity', "example_alternative", False),
]
import_data(
    db_map,
    entity_classes=entity_classes,
    alternatives=alternatives,
    scenarios=scenarios,
    scenario_alternatives=scenario_alternatives,
    parameter_value_lists=parameter_value_lists,
    parameter_definitions=parameter_definitions,
    entities=entities,
    entity_groups=entity_groups,
    parameter_values=parameter_values,
    entity_alternatives=entity_alternatives,
)
```

Parameters

- **db_map** (*spinedb_api.DiffDatabaseMapping*) – database mapping
- **on_conflict** (*str*) – Conflict resolution strategy for `parameter_value.fix_conflict()`
- **entity_classes** (*list(tuple(str,tuple,str,int)*) – tuples of (name, dimension name tuple, description, display icon integer)
- **parameter_definitions** (*list(tuple(str,str,str,str)*) – tuples of (class name, parameter name, default value, parameter value list name, description)
- **entities** – (list(tuple(str,str or tuple(str)): tuples of (class name, entity name or element name list)
- **entity_alternatives** – (list(tuple(str,str or tuple(str),str,bool): tuples of (class name, entity name or element name list, alternative name, activity)
- **entity_groups** (*list(tuple(str,str,str))*) – tuples of (class name, group entity name, member entity name)
- **parameter_values** (*list(tuple(str,str or tuple(str),str,str|numeric, str]*) – tuples of (class name, entity name or element name list, parameter name, value,

alternative name)

- **alternatives** (*list(str,str)*) – tuples of (name, description)
- **scenarios** (*list(str,str)*) – tuples of (name, description)
- **scenario_alternatives** (*list(str,str,str)*) – tuples of (scenario name, alternative name, preceeding alternative name)
- **parameter_value_lists** (*list(str,str|numeric)*) – tuples of (list name, value)

Returns
number of items imported list: errors

Return type
int

spinedb_api.import_functions.**get_data_for_import**(*db_map*, *unparse_value=to_database*, *on_conflict='merge'*, *entity_classes=()*, *entities=()*, *entity_groups=()*, *entity_alternatives=()*, *parameter_definitions=()*, *parameter_values=()*, *parameter_value_lists=()*, *alternatives=()*, *scenarios=()*, *scenario_alternatives=()*, *metadata=()*, *entity_metadata=()*, *parameter_value_metadata=()*, *superclass_subclasses=()*, *object_classes=()*, *relationship_classes=()*, *object_parameters=()*, *relationship_parameters=()*, *objects=()*, *relationships=()*, *object_groups=()*, *object_parameter_values=()*, *relationship_parameter_values=()*, *object_metadata=()*, *relationship_metadata=()*, *object_parameter_value_metadata=()*, *relationship_parameter_value_metadata=()*, *tools=()*, *features=()*, *tool_features=()*, *tool_feature_methods=()*)

Yields data to import into a Spine DB.

Parameters

- **db_map** (*spinedb_api.DiffDatabaseMapping*) – database mapping
- **on_conflict** (*str*) – Conflict resolution strategy for `fix_conflict()`
- **entity_classes** (*list(tuple(str,tuple,str,int)*) – tuples of (name, dimension name tuple, description, display icon integer)
- **parameter_definitions** (*list(tuple(str,str,str,str)*) – tuples of (class name, parameter name, default value, parameter value list name)
- **entities** – (list(tuple(str,str or tuple(str)): tuples of (class name, entity name or element name list)
- **entity_alternatives** – (list(tuple(str,str or tuple(str),str,bool): tuples of (class name, entity name or element name list, alternative name, activity)
- **entity_groups** (*list(tuple(str,str,str))*) – tuples of (class name, group entity name, member entity name)

- **parameter_values** (*list(tuple(str,str or tuple(str),str,str|numeric, str])*) – tuples of (class name, entity name or element name list, parameter name, value, alternative name)
- **alternatives** (*list(str,str)*) – tuples of (name, description)
- **scenarios** (*list(str,str)*) – tuples of (name, description)
- **scenario_alternatives** (*list(str,str,str)*) – tuples of (scenario name, alternative name, preceeding alternative name)
- **parameter_value_lists** (*list(str,str|numeric)*) – tuples of (list name, value)

Yields
str – item type tuple(list,list,list): tuple of (items to add, items to update, errors)

spinedb_api.import_functions.**import_superclass_subclasses**(*db_map*, *data*)

Imports superclass_subclasses into a Spine database using a standard format.

Parameters
- **db_map** (*spinedb_api.DiffDatabaseMapping*) – database mapping
- **data** (*list(tuple(str,tuple,str,int)*) – tuples of (superclass name, subclass name)

Returns
number of items imported list: errors

Return type
int

spinedb_api.import_functions.**import_entity_classes**(*db_map*, *data*)

Imports entity classes into a Spine database using a standard format.

Parameters
- **db_map** (*spinedb_api.DiffDatabaseMapping*) – database mapping
- **data** (*list(tuple(str,tuple,str,int,bool)*) – tuples of (name, dimension name tuple, description, display icon integer, active by default flag)

Returns
number of items imported list: errors

Return type
int

spinedb_api.import_functions.**import_entities**(*db_map*, *data*)

Imports entities into a Spine database using a standard format.

Parameters
- **db_map** (*spinedb_api.DiffDatabaseMapping*) – database mapping
- **data** – (list(tuple(str,str or tuple(str)): tuples of (class name, entity name or element name list)

Returns
number of items imported list: errors

Return type
int

spinedb_api.import_functions.**import_entity_alternatives**(*db_map*, *data*)

Imports entity alternatives into a Spine database using a standard format.

Parameters

- **db_map** (*spinedb_api.DiffDatabaseMapping*) – database mapping
- **data** – (list(tuple(str,str or tuple(str),str,bool): tuples of (class name, entity name or element name list, alternative name, activity)

Returns
number of items imported list: errors

Return type
int

spinedb_api.import_functions.**import_entity_groups**(*db_map*, *data*)

Imports entity groups into a Spine database using a standard format.

Parameters

- **db_map** (*spinedb_api.DiffDatabaseMapping*) – database mapping
- **data** (*list(tuple(str,str,str))*) – tuples of (class name, group entity name, member entity name)

Returns
number of items imported list: errors

Return type
int

spinedb_api.import_functions.**import_parameter_definitions**(*db_map*, *data*, *unparse_value=to_database*)

Imports parameter definitions into a Spine database using a standard format.

Parameters

- **db_map** (*spinedb_api.DiffDatabaseMapping*) – database mapping
- **data** (*list(tuple(str,str,str,str))*) – tuples of (class name, parameter name, default value, parameter value list name)

Returns
number of items imported list: errors

Return type
int

spinedb_api.import_functions.**import_parameter_values**(*db_map*, *data*, *unparse_value=to_database*, *on_conflict='merge'*)

Imports parameter values into a Spine database using a standard format.

Parameters

- **db_map** (*spinedb_api.DiffDatabaseMapping*) – database mapping
- **data** (*list(tuple(str,str or tuple(str),str,str|numeric,str]*) – tuples of (class name, entity name or element name list, parameter name, value, alternative name)
- **on_conflict** (*str*) – Conflict resolution strategy for `fix_conflict()`

Returns
number of items imported list: errors

Return type
int

spinedb_api.import_functions.**import_alternatives**(*db_map*, *data*)

Imports alternatives into a Spine database using a standard format.

Parameters

- **db_map** (*spinedb_api.DiffDatabaseMapping*) – database mapping
- **data** (*list(str,str)*) – tuples of (name, description)

Returns
number of items imported list: errors

Return type
int

spinedb_api.import_functions.**import_scenarios**(*db_map*, *data*)

Imports scenarios into a Spine database using a standard format.

Parameters

- **db_map** (*spinedb_api.DiffDatabaseMapping*) – database mapping
- **data** (*list(str, bool, str)*) – tuples of (name, <unused_bool>, description)

Returns
number of items imported list: errors

Return type
int

spinedb_api.import_functions.**import_scenario_alternatives**(*db_map*, *data*)

Imports scenario alternatives into a Spine database using a standard format.

Parameters

- **db_map** (*spinedb_api.DiffDatabaseMapping*) – database mapping
- **data** (*list(str,str,str)*) – tuples of (scenario name, alternative name, preceeding alternative name)

Returns
number of items imported list: errors

Return type
int

spinedb_api.import_functions.**import_parameter_value_lists**(*db_map*, *data*, *unparse_value=to_database*)

Imports parameter value lists into a Spine database using a standard format.

Parameters

- **db_map** (*spinedb_api.DiffDatabaseMapping*) – database mapping
- **data** (*list(str,str|numeric)*) – tuples of (list name, value)

Returns
number of items imported list: errors

Return type
int

spinedb_api.import_functions.**import_metadata**(*db_map*, *data*)

Imports metadata into a Spine database using a standard format.

Parameters

- **db_map** (*spinedb_api.DiffDatabaseMapping*) – database mapping
- **data** (*list(tuple(str,str))*) – tuples of (entry name, value)

Returns

number of items imported list: errors

Return type

int

spinedb_api.parameter_value

Parameter values in a Spine DB can be of different types (see *Parameter value format*). For each of these types, this module provides a Python class to represent values of that type.

Table 1: Parameter value type and Python class

type	Python class
date_time	*DateTime*
duration	*Duration*
array	*Array*
time_pattern	*TimePattern*
time_series	*TimeSeriesFixedResolution* and *TimeSeriesVariableResolution*
map	*Map*

The module also provides the functions *to_database()* and *from_database()* to translate between instances of the above classes and their DB representation (namely, the *value* and *type* fields that would go in the `parameter_value` table).

For example, to write a Python object into a parameter value in the DB:

```
# Create the Python object
parsed_value = TimeSeriesFixedResolution(
    "2023-01-01T00:00",             # start
    "1D",                           # resolution
    [9, 8, 7, 6, 5, 4, 3, 2, 1, 0], # values
    ignore_year=False,
    repeat=False,
)
# Translate it to value and type
value, type_ = to_database(parsed_value)
# Add a parameter_value to the DB with that value and type
with DatabaseMapping(url) as db_map:
    db_map.add_parameter_value_item(
        entity_class_name="cat",
        entity_byname=("Tom",),
        parameter_definition_name="number_of_lives",
        alternative_name="Base",
        value=value,
        type=type_,
```

(continues on next page)

(continued from previous page)

```
    )
    db_map.commit_session("Tom is living one day at a time")
```

The value can be accessed as a Python object using the `parsed_value` field:

```
# Get the parameter_value from the DB
with DatabaseMapping(url) as db_map:
    pval_item = db_map.get_parameter_value_item(
        entity_class_name="cat",
        entity_byname=("Tom",),
        parameter_definition_name="number_of_lives",
        alternative_name="Base",
    )
value = pval_item["parsed_value"]
```

In the rare case where a manual conversion from a DB value to Python object is needed, use *from_database()*:

```
# Obtain value and type
value, type_ = pval_item["value"], pval_item["type"]
# Translate value and type to a Python object manually
parsed_value = from_database(value, type_)
```

Module Contents

Classes

ParameterValue	Base for all classes representing parameter values.
DateTime	A parameter value of type 'date_time'. A point in time.
Duration	A parameter value of type 'duration'. An extension of time.
IndexedValue	Base for all classes representing indexed parameter values.
Array	A parameter value of type 'array'. A one dimensional array with zero based indexing.
TimePattern	A parameter value of type 'time_pattern'.
TimeSeries	Base for all classes representing 'time_series' parameter values.
TimeSeriesFixedResolution	A parameter value of type 'time_series'.
TimeSeriesVariableResolution	A parameter value of type 'time_series'.
Map	A parameter value of type 'map'. A mapping from key to value, where the values can be other instances

Functions

from_database(value[, type_])	Converts a parameter value from the DB into a Python object.
to_database(parsed_value)	Converts a Python object representing a parameter value into their DB representation.

spinedb_api.parameter_value.**from_database**(*value*, *type_=None*)

Converts a parameter value from the DB into a Python object.

Parameters

- **value** (*bytes or None*) – the *value* field from the `parameter_value` table.
- **type** (*str, optional*) – the *type* field from the `parameter_value` table.

Returns
a Python object representing the parameter value.

Return type
ParameterValue, float, str, bool or None

spinedb_api.parameter_value.**to_database**(*parsed_value*)

Converts a Python object representing a parameter value into their DB representation.

Parameters
parsed_value (*any*) – the Python object.

Returns
the *value* and *type* fields that would go in the `parameter_value` table.

Return type
tuple(bytes,str)

class spinedb_api.parameter_value.**ParameterValue**

Base for all classes representing parameter values.

abstract static type_()

Returns the type of the parameter value represented by this object.

Returns
str

to_database()

Returns the DB representation of this object. Equivalent to calling *to_database()* with it.

Returns
the *value* and *type* fields that would go in the `parameter_value` table.

Return type
tuple(bytes,str)

class spinedb_api.parameter_value.**DateTime**(*value=None*)

Bases: *ParameterValue*

A parameter value of type 'date_time'. A point in time.

Parameters
value (*DateTime* or str or `datetime`) – the *date_time* value.

property value

The value.

Returns
datetime

class spinedb_api.parameter_value.**Duration**(*value=None*)

Bases: *ParameterValue*

A parameter value of type 'duration'. An extension of time.

Parameters
value (str or *Duration* or relativedelta) – the *duration* value.

property value

The value.

Returns
relativedelta

class spinedb_api.parameter_value.**IndexedValue**(*values*, *value_type=None*, *index_name=''*)

Bases: *ParameterValue*

Base for all classes representing indexed parameter values.

property indexes

The indexes.

Returns
ndarray

property values

The values.

Returns
ndarray

property value_type

The type of the values.

Return type
type

get_nearest(*index*)

Returns the value at the nearest index to the given one.

Parameters
index (*any*) – The index.

Returns
The value.

Return type
any

get_value(*index*)

Returns the value at the given index.

Parameters
index (*any*) – The index.

Returns
The value.

Return type
any

`set_value`(*index*, *value*)
Sets the value at the given index.

Parameters

- **index** (*any*) – The index.
- **value** (*any*) – The value.

`class spinedb_api.parameter_value.Array`(*values*, *value_type=None*, *index_name=''*)
Bases: *IndexedValue*

A parameter value of type 'array'. A one dimensional array with zero based indexing.

Parameters

- **values** (*Sequence*) – the array values.
- **value_type** (*type, optional*) – the type of the values; if not given, it will be deduced from *values*. Defaults to float if *values* is empty.
- **index_name** (*str*) – the name you would give to the array index in your application.

`class spinedb_api.parameter_value.TimePattern`(*indexes*, *values*, *index_name=''*)
Bases: *IndexedValue*

A parameter value of type 'time_pattern'. A mapping from time patterns strings to numerical values.

Parameters

- **indexes** (*list*) – the time pattern strings.
- **values** (*Sequence*) – the values associated to different patterns.
- **index_name** (*str*) – index name

`class spinedb_api.parameter_value.TimeSeries`(*values*, *ignore_year*, *repeat*, *index_name=''*)
Bases: *IndexedValue*

Base for all classes representing 'time_series' parameter values.

`property ignore_year`
Whether the year should be ignored.

Return type
bool

`property repeat`
Whether the series is repeating.

Return type
bool

`class spinedb_api.parameter_value.TimeSeriesFixedResolution`(*start*, *resolution*, *values*, *ignore_year*, *repeat*, *index_name=''*)
Bases: *TimeSeries*

A parameter value of type 'time_series'. A mapping from time stamps to numerical values, with fixed durations between the time stamps.

When getting the indexes the durations are applied cyclically.

Currently, there is no support for the *ignore_year* and *repeat* options other than having getters for their values.

Parameters

- **start** (str or datetime or datetime64) – the first time stamp
- **resolution** (str, relativedelta, list) – duration(s) between the time stamps.
- **values** (*Sequence*) – the values in the time-series.
- **ignore_year** (*bool*) – True if the year should be ignored.
- **repeat** (*bool*) – True if the series is repeating.
- **index_name** (*str*) – index name.

property indexes

The indexes.

Returns
ndarray

property start

Returns the start index.

Return type
datetime64

property resolution

Returns the resolution as list of durations.

Return type
list(*Duration*)

class spinedb_api.parameter_value.**TimeSeriesVariableResolution**(*indexes*, *values*, *ignore_year*, *repeat*, *index_name=''*)

Bases: *TimeSeries*

A parameter value of type 'time_series'. A mapping from time stamps to numerical values with arbitrary time steps.

Parameters

- **indexes** (Sequence(datetime64)) – the time stamps.
- **values** (*Sequence*) – the value for each time stamp.
- **ignore_year** (*bool*) – True if the year should be ignored.
- **repeat** (*bool*) – True if the series is repeating.
- **index_name** (*str*) – index name.

class spinedb_api.parameter_value.**Map**(*indexes*, *values*, *index_type=None*, *index_name=''*)

Bases: *IndexedValue*

A parameter value of type 'map'. A mapping from key to value, where the values can be other instances of *ParameterValue*.

Parameters

- **indexes** (*Sequence*) – the indexes in the map.
- **values** (*Sequence*) – the value for each index.
- **index_type** (*type or NoneType*) – index type or None to deduce from indexes.
- **index_name** (*str*) – index name.

`is_nested`()

Whether any of the values is also a map.

Return type
bool

`value_to_database_data`()

Returns map's database representation's 'data' dictionary.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

A

H

I

M

P

Q

R

S

T

U

V