

---

# Track

*Release 0.0.1-c92247e2*

Jan 10, 2020



<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Documentation</b>	<b>5</b>
<b>3</b>	<b>Overview</b>	<b>7</b>
	<b>Python Module Index</b>	<b>55</b>
	<b>Index</b>	<b>57</b>







# CHAPTER 1

---

## Installation

---

```
pip install -r requirements
python setup.py install
```





## CHAPTER 2

---

### Documentation

---

```
sphinx-build -W --color -c docs/src/ -b html docs/src/ docs/build/html
(cd docs/build/html && python -m http.server 8000 --bind 127.0.0.1)
```



# CHAPTER 3

---

## Overview

---

```
from track import TrackClient

client = TrackClient('file://client_test.json')
client.set_project(name='test_client')

trial = client.new_trial()
trial.log_arguments(batch_size=256)

with trial:
    trial.log_metrics(step=1, epoch_loss=1)
    trial.log_metrics(accuracy=0.98)

client.save()
client.report()
```

## 3.1 Overview

Track as 3 kind of objects, Project, Trial Group and Trial.

- **Project** is a top level object that holds all of its trials and groups
- **TrialGroup** is a set of trials. They are used to order trials together. trials can belong to multiple groups
- **Trial** is the object holding all the information about a given training session. the trial object is the backbone of track and it is the object you will have to deal with the most often

### 3.1.1 Overview

```
from track import TrackClient

client = TrackClient('file://client_test.json')
```

(continues on next page)

(continued from previous page)

```

project = client.set_project(name='paper_78997')
group = client.set_group(name='idea_4573')

trial = client.new_trial(name='final_trial_2', description='almost graduating')
trial.log_arguments(batch_size=256, lr=0.01, momentum=0.99)
trial.log_metadata(gpu='V100')

# start the trial explicitly
with trial:

    for e in range(epochs):

        for batch in dataset:

            # trial helper that compute elapsed time inside a block
            with trial.chrono('batch_time'):
                ...
                loss += ...

            trial.log_metrics(step=e, epoch_loss=loss)

        trial.log_metrics(accuracy=0.98)

client.report()

```

You can find the sample of a report below

```

{
  "revision": 1,
  "name": "final_trial_2",
  "description": "almost graduating",
  "version": "a8c3",
  "tags": {
    "workers": 8,
    "hpo": "byopt"
  },
  "parameters": {
    "batch_size": 32,
    "cuda": true,
    "workers": 0,
    "seed": 0,
    "epochs": 2,
    "arch": "convnet",
    "lr": 0.1,
    "momentum": 0.9,
    "opt_level": "O0",
    "break_after": null,
    "data": "mnist",
    "backend": null
  },
  "metadata": {},
  "metrics": {
    "epoch_loss": {
      "0": 2.306920262972514,
      "1": 2.307889754740397
    }
  }
}

```

(continues on next page)

(continued from previous page)

```

},
"chronos": {
  "runtime": 3142.5199086666107,
  "batch_time": {
    "avg": 0.6737696465350126,
    "min": 0.019209623336791992,
    "max": 445.9658739566803,
    "sd": 12.500646799505962,
    "count": 3751,
    "unit": "s"
  }
},
"errors": [],
"status": {
  "value": 302,
  "name": "Completed"
}
}

```

### 3.1.2 Log Metrics

User can log metrics with a step or without. *step* is used as key in a dictionary and should be unique

```

trial.log_metrics(step=e, epoch_loss=loss, metric2=value)

trial.log_metrics(cost=val)

```

### 3.1.3 Time things

You can easily time things with *chrono*. Do not forget if you are measuring GPU compute time should should synchronize to make sure the computation are done before computing the elapsed time.

```

with trial.chrono('long_compute'):
    sleep(100)

```

### 3.1.4 Save arbitrary data

You can use metadata to save information on a specific trial that might not be reflected by its parameters

```

trial.log_metadata(had_short_hair_when_running_this_trial=False)

```

### 3.1.5 Experiment Report

Get a quick overview of all the data that was saved up during training

```

trial.report()

```

## 3.2 Backends

Track was made to support different backends, you can even implement your own!

### 3.2.1 Local Backend

Track implements a local storage backend for quick and simple experiments

```
client = TrackClient(f'file://report.json')
```

### 3.2.2 CockroachDB backend

Track implements a backend that can use a running cockroachdb instance as storage.

```
address = '127.0.0.1'
port = 8123
client = TrackClient(f'cockroach://{address}:{port}')
```

### 3.2.3 Socket backend

Track implements a backend that uses sockets to forward request to a remote server

#### Server

Simple servers that receive request from the client and forwards all request to another backend. The example below forwards all request to the local backend, allowing to have a single process modifying the file.

```
from track.persistence.socketed import start_track_server

address = '127.0.0.1'
port = 8123
layer = 'AES'
start_track_server('file:server_test.json', address, port, backend=layer)
```

#### Client

Start a client that forwards all request to a remote server

```
username = ...
password = ...
address = '127.0.0.1'
port = 8123
layer = 'AES' # supported AES or (None, i.e put nothing)
client = TrackClient(f'socket://{username}:{password}@{address}:{port}?security_layer=
→{layer}')
```

### 3.2.4 Bring Your Own Backend

To implement your own you can simply extend `track.persistence.protocol.Protocol`

```
from track.persistence import register
from track.persistence.protocol import Protocol

class MyOwnBackend(Protocol):
    ....

register('byob', MyOwnBackend)
```

You can then use it naturally

```
client = TrackClient('byob://....')
```

## 3.3 Simple example

### 3.3.1 Installation and setup

In this tutorial you will run a very simple MNIST example in pytorch using Track. First, install Track, then install pytorch, torchvision and clone the PyTorch [examples repository](#):

```
$ pip3 install torch torchvision
$ git clone git@github.com:pytorch/examples.git
```

### 3.3.2 Adapting the code of MNIST example

After cloning pytorch examples repository, cd to mnist folder:

```
$ cd examples/mnist
```

In main, just after parsing the arguments, you can initialize the track client and create a trial. The client specifies how will the data be saved on your computer, different methods are supported. Once the client is initialized, you can create a new trial.

A trial is a set of data retrieved for a set of arguments.

```
$ ....
$ args = parser.parse_args()
$ client = TrackClient('file:mnist_example.json')
$ trial = client.new_trial(arguments=args)
```

Then you can store any kind of data that you think will be useful. In our example we decided to save the error rate on the test set

```
$ def test(args, model, device, test_loader, trial):
$     ...
$     trial.log_metrics(error_rate=1 - (correct / len(test_loader.dataset)))
```

At the end of training file `mnist_example.json` will be generated holding all the data you saved during training.

## 3.4 track

### 3.4.1 track package

#### Subpackages

#### track.aggregators package

#### Submodules

#### track.aggregators.aggregator module

**class** track.aggregators.aggregator.**Aggregator**  
Bases: object

#### Attributes

**val** Return the last observed value

#### Methods

---

<i>lazy</i> (aggregator_t, \*\*kwargs)	Lazily instantiate the underlying aggregator
--	--

---

<b>append</b>	
<b>to_json</b>	

**append** (*self*, *other*)

**static lazy** (*aggregator\_t*, \\*\\*kwargs)  
Lazily instantiate the underlying aggregator

**to\_json** (*self*, *short=False*)

**val**  
Return the last observed value

**class** track.aggregators.aggregator.**RingAggregator** (*n*, *dtype='f'*)  
Bases: *track.aggregators.aggregator.Aggregator*

Saves the *n* last elements. Start overriding the elements once *n* elements is reached

#### Attributes

**val** Return the last observed value

#### Methods

---

<i>lazy</i> ( <i>n</i> , <i>dtype</i> )	Lazily instantiate the underlying aggregator
---	--

---



<b>append</b>	
<b>to_json</b>	

**append** (*self*, *other*)

**static lazy** (*n*, *dtype*)

Lazily instantiate the underlying aggregator

**to\_json** (*self*, *short=False*)

**val**

Return the last observed value

**class** `track.aggregators.aggregator.StatAggregator` (*skip\_obs=10*)

Bases: `track.aggregators.aggregator.Aggregator`

Compute mean, sd, min, max; does not keep the entire history. This is useful if you are worried about memory usage and the values should not vary much. i.e keeping the entire history is not useful.

#### Attributes

**avg**

**max**

**min**

**sd**

**sum**

**total**

**val** Return the last observed value

#### Methods

---

*lazy*(*skip*)

Lazily instantiate the underlying aggregator

---

<b>append</b>	
<b>from_json</b>	
<b>to_json</b>	

**append** (*self*, *other*)

**avg**

**static from\_json** (*data*)

**static lazy** (*skip*)

Lazily instantiate the underlying aggregator

**max**

**min**

**sd**

**sum**

**to\_json** (*self*, *short=False*)

**total**

**val**

Return the last observed value

**class** `track.aggregators.aggregator.TimeSeriesAggregator`

Bases: `track.aggregators.aggregator.Aggregator`

Keeps the entire history of the metric

**Attributes**

**val** Return the last observed value

## Methods

---

*lazy*()

Lazily instantiate the underlying aggregator

---

<b>append</b>	
<b>to_json</b>	

**append** (*self*, *other*)

**static lazy** ()

Lazily instantiate the underlying aggregator

**to\_json** (*self*, *short=False*)

**val**

Return the last observed value

**class** `track.aggregators.aggregator.ValueAggregator` (*val=None*)

Bases: `track.aggregators.aggregator.Aggregator`

Does not Aggregate only keeps the latest value

**Attributes**

**val** Return the last observed value

## Methods

---

*lazy*()

Lazily instantiate the underlying aggregator

---

<b>append</b>	
<b>to_json</b>	

**append** (*self*, *other*)

**static lazy** ()

Lazily instantiate the underlying aggregator

**to\_json** (*self*, *short=False*)

## Module contents

## Submodules

## Methods

```
append(self, item)
last(self)
to_list(self)
```

## Module contents

## Submodules

## Module contents

## Submodules

[illegible]

cockroach db is a highly resilient database that allow us to remove the Master in a traditional distributed setup.  
This spawn a cockroach node that will store its data in *location*

#### Attributes

**build**  
**client\_flags**  
**node\_id**  
**sql**  
**status**  
**webui**

#### Methods

<b>parse</b>	
<b>start</b>	
<b>stop</b>	
<b>wait</b>	

**build**  
**client\_flags**  
**node\_id**  
**parse** (*self*, *properties*, *line*)  
**sql**  
**start** (*self*, *wait=True*)  
**status**  
**stop** (*self*)  
**wait** (*self*)  
**webui**

#### Module contents

**track.persistence package**

#### Submodules

**track.persistence.cockroach module**

**track.persistence.cometml module**

## track.persistence.local module

**exception** track.persistence.local.**ConcurrentWrite** (*msg*)

Bases: `Exception`

**class** track.persistence.local.**FileProtocol** (*uri, strict=True, eager=True*)

Bases: `track.persistence.protocol.Protocol`

Local File storage to manage experiments

### Parameters

**uri: str** resource to use to store the experiment *file://my\_file.json*

**strict: bool** forces the storage to be correct. if we use the file protocol as an in-memory storage we might get some inconsistencies we can use this flag to ignore them

**eager: bool** eagerly update the underlying files. This is necessary if multiple processes are reading from the file

### Methods

---

`commit(self[, file_name_override])`

Forces to persist the change

---

<b>add_group_trial</b>	
<b>add_project_trial</b>	
<b>add_trial_tags</b>	
<b>fetch_and_update_group</b>	
<b>fetch_and_update_trial</b>	
<b>fetch_groups</b>	
<b>fetch_projects</b>	
<b>fetch_trials</b>	
<b>get_project</b>	
<b>get_trial</b>	
<b>get_trial_group</b>	
<b>log_trial_arguments</b>	
<b>log_trial_chrono_finish</b>	
<b>log_trial_chrono_start</b>	
<b>log_trial_finish</b>	
<b>log_trial_metadata</b>	
<b>log_trial_metrics</b>	
<b>log_trial_start</b>	
<b>new_project</b>	
<b>new_trial</b>	
<b>new_trial_group</b>	
<b>set_group_metadata</b>	
<b>set_trial_status</b>	

**add\_group\_trial** (*self, \*args, \*\*kwargs*)

**add\_project\_trial** (*self, \*args, \*\*kwargs*)

**add\_trial\_tags** (*self, \*args, \*\*kwargs*)

```
commit (self, file_name_override=None, **kwargs)
    Forces to persist the change

fetch_and_update_group (self, *args, **kwargs)

fetch_and_update_trial (self, *args, **kwargs)

fetch_groups (self, *args, **kwargs)

fetch_projects (self, *args, **kwargs)

fetch_trials (self, *args, **kwargs)

get_project (self, *args, **kwargs)

get_trial (self, *args, **kwargs)

get_trial_group (self, *args, **kwargs)

log_trial_arguments (self, *args, **kwargs)

log_trial_chrono_finish (self, *args, **kwargs)

log_trial_chrono_start (self, *args, **kwargs)

log_trial_finish (self, *args, **kwargs)

log_trial_metadata (self, *args, **kwargs)

log_trial_metrics (self, *args, **kwargs)

log_trial_start (self, *args, **kwargs)

new_project (self, *args, **kwargs)

new_trial (self, *args, **kwargs)

new_trial_group (self, *args, **kwargs)

set_group_metadata (self, *args, **kwargs)

set_trial_status (self, *args, **kwargs)
```

```
class track.persistence.local.LockFileRemover (filename)
    Bases: track.utils.signal.SignalHandler
```

## Methods

<b>atexit</b>	
<b>remove</b>	
<b>sigint</b>	
<b>sigterm</b>	

```
atexit (self)

remove (self)

sigint (self, signum, frame)

sigterm (self, signum, frame)
```

```
class track.persistence.local.MultiLock (obj)
    Bases: object
```

`track.persistence.local.execute_query(obj, query)`

Check if the object *obj* matches the query.

The query is a dictionary specifying constraint on each of the object attributes

`track.persistence.local.lock_atomic_write(fun)`

`track.persistence.local.lock_guard(readonly, atomic=False)`

Protect a function call with a lock. reload the database before the action and save it afterwards

`track.persistence.local.lock_read(fun)`

`track.persistence.local.lock_write(fun)`

`track.persistence.local.make_lock(name, eager)`

`track.persistence.local.query_gt(obj, attrs, val)`

`track.persistence.local.query_in(obj, attrs, choices)`

`track.persistence.local.query_lte(obj, attrs, val)`

`track.persistence.local.query_ne(obj, attrs, val)`

### **track.persistence.multiplexer module**

**class** `track.persistence.multiplexer.ProtocolMultiplexer(*backends)`

Bases: `object`

#### **Methods**

---

`get_project(self, *args, **kwargs)`

---

<b>add_group_trial</b>	
<b>add_project_trial</b>	
<b>add_trial_tags</b>	
<b>commit</b>	
<b>fetch_and_update_group</b>	
<b>fetch_and_update_trial</b>	
<b>fetch_groups</b>	
<b>fetch_projects</b>	
<b>fetch_trials</b>	
<b>get_trial</b>	
<b>get_trial_group</b>	
<b>log_trial_arguments</b>	
<b>log_trial_chrono_finish</b>	
<b>log_trial_chrono_start</b>	
<b>log_trial_finish</b>	
<b>log_trial_metadata</b>	
<b>log_trial_metrics</b>	
<b>log_trial_start</b>	
<b>new_project</b>	
<b>new_trial</b>	
<b>new_trial_group</b>	
<b>set_trial_status</b>	

```
add_group_trial (self, *args, **kwargs)
add_project_trial (self, *args, **kwargs)
add_trial_tags (self, *args, **kwargs)
commit (self, *args, **kwargs)
fetch_and_update_group (self, *args, **kwargs)
fetch_and_update_trial (self, *args, **kwargs)
fetch_groups (self, *args, **kwargs)
fetch_projects (self, *args, **kwargs)
fetch_trials (self, *args, **kwargs)
get_project (self, *args, **kwargs)
get_trial (self, *args, **kwargs)
get_trial_group (self, *args, **kwargs)
log_trial_arguments (self, *args, **kwargs)
log_trial_chrono_finish (self, *args, **kwargs)
log_trial_chrono_start (self, *args, **kwargs)
log_trial_finish (self, *args, **kwargs)
log_trial_metadata (self, *args, **kwargs)
log_trial_metrics (self, *args, **kwargs)
log_trial_start (self, *args, **kwargs)
new_project (self, *args, **kwargs)
```



```

new_trial (self, *args, **kwargs)
new_trial_group (self, *args, **kwargs)
set_trial_status (self, *args, **kwargs)

```

## track.persistence.protocol module

```

class track.persistence.protocol.Protocol
    Bases: object

```

### Methods

<code>add_group_trial(self, group, trial)</code>	Add a trial to a group
<code>add_project_trial(self, project, trial)</code>	Add a trial to a project
<code>add_trial_tags(self, trial, \**kwargs)</code>	Add tags to a trial
<code>commit(self, \**kwargs)</code>	Forces to persist the change
<code>fetch_and_update_group(self, query, attr, ...)</code>	Fetch and update a single group
<code>fetch_and_update_trial(self, query, attr, ...)</code>	Fetch and update a single trial
<code>fetch_groups(self, query)</code>	Fetch groups according to a given query
<code>fetch_projects(self, query)</code>	Fetch projects according to a given query
<code>fetch_trials(self, query)</code>	Fetch trials according to a given query
<code>get_project(self, project)</code>	Fetch a project according to the given definition
<code>get_trial(self, trial)</code>	Fetch trials according to a given definition
<code>get_trial_group(self, group)</code>	Fetch a group according to a given definition
<code>log_trial_arguments(self, trial, \**kwargs)</code>	Save the arguments a trail
<code>log_trial_chrono_finish(self, trial, name, ...)</code>	Send the end signal for an event
<code>log_trial_chrono_start(self, trial, name, ...)</code>	Send the start signal for an event
<code>log_trial_finish(self, trial, exc_type, ...)</code>	Send the trial end signal
<code>log_trial_metadata(self, trial, aggregator, ...)</code>	Save metadata for a given trials
<code>log_trial_metrics(self, trial, step, ...)</code>	Save metrics for a given trials
<code>log_trial_start(self, trial)</code>	Send the trial start signal
<code>new_project(self, project)</code>	Insert a new project
<code>new_trial(self, trial[, auto_increment])</code>	Insert a new trial
<code>new_trial_group(self, group)</code>	Create a new group
<code>set_trial_status(self, trial, status[, error])</code>	Change trial status

```

add_group_trial (self, group: track.structure.TrialGroup, trial: track.structure.Trial)
    Add a trial to a group

```

```

add_project_trial (self, project: track.structure.Project, trial: track.structure.Trial)
    Add a trial to a project

```

```

add_trial_tags (self, trial, **kwargs)
    Add tags to a trial

```

### Parameters

**trial:** **Trial** trial reference

**kwargs:** key value pair of the data to save

```

commit (self, **kwargs)

```

Forces to persist the change

**fetch\_and\_update\_group** (*self, query, attr, \*args, \*\*kwargs*)

Fetch and update a single group

**Parameters**

**query:** **Dict** dictionary to fetch groups

**attr:** **str** name of the update function to call on each selected group

**\*args:** additional positional arguments for the attr function

**\*\*kwargs:** additional keyword arguments for the attr function

**Returns**

**returns the modified group**

**fetch\_and\_update\_trial** (*self, query, attr, \*args, \*\*kwargs*)

Fetch and update a single trial

**Parameters**

**query:** **Dict** dictionary to fetch trials

**attr:** **str** name of the update function to call on each selected trials

**\*args:** additional positional arguments for the attr function

**\*\*kwargs:** additional keyword arguments for the attr function

**Returns**

**returns the modified trial**

**fetch\_groups** (*self, query*)

Fetch groups according to a given query

**fetch\_projects** (*self, query*)

Fetch projects according to a given query

**fetch\_trials** (*self, query*) → List[track.structure.Trial]

Fetch trials according to a given query

**get\_project** (*self, project: track.structure.Project*) → Union[track.structure.Project, NoneType]

Fetch a project according to the given definition

**Parameters**

**project:** **Project** project definition used for the lookup

**Returns**

**returns a project object or None**

**get\_trial** (*self, trial: track.structure.Trial*) → List[track.structure.Trial]

Fetch trials according to a given definition

**Parameters**

**trial:** **Trial** trial definition used for the lookup

**get\_trial\_group** (*self, group: track.structure.TrialGroup*) → Union[track.structure.TrialGroup, NoneType]

Fetch a group according to a given definition

**Parameters**

**group: TrialGroup** group definition used for the lookup

#### Returns

**returns a grouo**

**log\_trial\_arguments** (*self*, *trial*: *track.structure.Trial*, *\*\*kwargs*)

Save the arguments a trail

#### Parameters

**trial: Trial** trial for which the arguments are for

**kwargs:** key value pair of arguments

**log\_trial\_chrono\_finish** (*self*, *trial*, *name*, *exc\_type*, *exc\_val*, *exc\_tb*)

Send the end signal for an event

#### Parameters

**trial: Trial** trial sending the event

**name: str** name of the event

**exc\_type:** Exception object

**exc\_val** Exception value

**exc\_tb:** Traceback

**log\_trial\_chrono\_start** (*self*, *trial*, *name*: *str*, *aggregator*: *Callable[[], track.aggregators.aggregator.Aggregator]* = *<function StatAggregator.lazy.<locals>.<lambda> at 0x7ff802abcf28>*, *start\_callback*=*None*, *end\_callback*=*None*)

Send the start signal for an event

#### Parameters

**trial: Trial** trial sending the event

**name: str** name of the event

**aggregator: Aggregator** container used to accumulate elapsed time

**start\_callback: Callable** function called at start time

**end\_callback: Callable** function called at the end

**log\_trial\_finish** (*self*, *trial*, *exc\_type*, *exc\_val*, *exc\_tb*)

Send the trial end signal

#### Parameters

**trial: Trial** reference to the trial that finished

**log\_trial\_metadata** (*self*, *trial*: *track.structure.Trial*, *aggregator*: *Callable[[], track.aggregators.aggregator.Aggregator]* = *<function ValueAggregator.lazy.<locals>.<lambda> at 0x7ff802aabd90>*, *\*\*kwargs*)

Save metadata for a given trials

#### Parameters

**trial: Trial** trial reference

**kwargs:** key value pair of the data to save

**log\_trial\_metrics** (*self*, *trial*: *track.structure.Trial*, *step*: *<built-in function any>* = *None*,  
                          *aggregator*: *Callable[[], track.aggregators.aggregator.Aggregator]* = *None*,  
                          \*\**kwargs*)

Save metrics for a given trials

**Parameters**

**trial: Trial** trial reference

**kwargs:** key value pair of the data to save

**log\_trial\_start** (*self*, *trial*)

Send the trial start signal

**Parameters**

**trial: Trial** reference to the trial being started

**new\_project** (*self*, *project*: *track.structure.Project*)

Insert a new project

**Parameters**

**project: Project** project definition used for the insert

**new\_trial** (*self*, *trial*: *track.structure.Trial*, *auto\_increment=False*)

Insert a new trial

**Parameters**

**trial: Trial** trial definition used for the insert

**auto\_increment: bool** If trial exist increment revision number

**Returns**

Returns None if Trial already exists and auto\_increment is False

**new\_trial\_group** (*self*, *group*: *track.structure.TrialGroup*)

Create a new group

**Parameters**

**group: TrialGroup** group definition used for the insert

**set\_trial\_status** (*self*, *trial*: *track.structure.Trial*, *status*, *error=None*)

Change trial status

**Parameters**

**trial: Trial** trial reference

**status:** new status to update the trial too

**error:** in case the user is changing to a state representing an error it can also provide an error identification string

## track.persistence.socketed module

**Implement a Remote Logger.** Client forwards all the user's request down to the server that executes them one by one.

**exception** `track.persistence.socketed.RPCCallFailure` (*message*, *trace=None*)

Bases: `Exception`

**class** `track.persistence.socketed.ServerSignalHandler` (*server*)

Bases: `track.utils.signal.SignalHandler`

## Methods

<b>atexit</b>	
<b>sigint</b>	
<b>sigterm</b>	

**sigint** (*self, signum, frame*)

**sigterm** (*self, signum, frame*)

**class** `track.persistence.socketed.SocketClient` (*uri*)

Bases: `track.persistence.protocol.Protocol`

Forwards all the local track requests to the track server that execute the requests and send back the results

Clients can provide a username and password for authentication

## Methods

<code>add_group_trial(self, group, trial)</code>	Add a trial to a group
<code>add_project_trial(self, project, trial)</code>	Add a trial to a project
<code>add_trial_tags(self, trial, <i>\**kwargs</i>)</code>	Add tags to a trial
<code>authenticate(self, uri)</code>	returns the username and password used for authentication purposes you can override this function to implement a custom authentication method
<code>commit(self, <i>\**kwargs</i>)</code>	Forces to persist the change
<code>fetch_and_update_group(self, query, attr, ...)</code>	Fetch and update a single group
<code>fetch_and_update_trial(self, query, attr, ...)</code>	Fetch and update a single trial
<code>fetch_groups(self, query)</code>	Fetch groups according to a given query
<code>fetch_projects(self, query)</code>	Fetch projects according to a given query
<code>fetch_trials(self, query)</code>	Fetch trials according to a given query
<code>get_project(self, project)</code>	Fetch a project according to the given definition
<code>get_trial(self, trial)</code>	Fetch trials according to a given definition
<code>get_trial_group(self, group)</code>	Fetch a group according to a given definition
<code>log_trial_arguments(self, trial, <i>\**kwargs</i>)</code>	Save the arguments a trail
<code>log_trial_chrono_finish(self, trial, name, ...)</code>	Send the end signal for an event
<code>log_trial_chrono_start(self, trial, name, ...)</code>	Send the start signal for an event
<code>log_trial_finish(self, trial, exc_type, ...)</code>	Send the trial end signal
<code>log_trial_metadata(self, trial, aggregator, ...)</code>	Save metadata for a given trials
<code>log_trial_metrics(self, trial, step, ...)</code>	Save metrics for a given trials
<code>log_trial_start(self, trial)</code>	Send the trial start signal
<code>new_project(self, project)</code>	Insert a new project
<code>new_trial(self, trial)</code>	Insert a new trial
<code>new_trial_group(self, group)</code>	Create a new group
<code>set_trial_status(self, trial, status[, error])</code>	Change trial status

**add\_group\_trial** (*self*, *group*: *track.structure.TrialGroup*, *trial*: *track.structure.Trial*)

Add a trial to a group

**add\_project\_trial** (*self*, *project*: *track.structure.Project*, *trial*: *track.structure.Trial*)

Add a trial to a project

**add\_trial\_tags** (*self*, *trial*, *\*\*kwargs*)

Add tags to a trial

#### Parameters

**trial: Trial** trial reference

**kwargs:** key value pair of the data to save

**authenticate** (*self*, *uri*)

returns the username and password used for authentication purposes you can override this function to implement a custom authentication method

**commit** (*self*, *\*\*kwargs*)

Forces to persist the change

**get\_project** (*self*, *project*: *track.structure.Project*)

Fetch a project according to the given definition

#### Parameters

**project: Project** project definition used for the lookup

#### Returns

**returns a project object or None**

**get\_trial** (*self*, *trial*: *track.structure.Trial*)

Fetch trials according to a given definition

#### Parameters

**trial: Trial** trial definition used for the lookup

**get\_trial\_group** (*self*, *group*: *track.structure.TrialGroup*)

Fetch a group according to a given definition

#### Parameters

**group: TrialGroup** group definition used for the lookup

#### Returns

**returns a grouo**

**log\_trial\_arguments** (*self*, *trial*: *track.structure.Trial*, *\*\*kwargs*)

Save the arguments a trail

#### Parameters

**trial: Trial** trial for which the arguments are for

**kwargs:** key value pair of arguments

**log\_trial\_chrono\_finish** (*self*, *trial*, *name*, *exc\_type*, *exc\_val*, *exc\_tb*)

Send the end signal for an event

#### Parameters

**trial: Trial** trial sending the event

**name: str** name of the event

**exc\_type:** Exception object

**exec\_val** Exception value

**exc\_tb:** Traceback

```
log_trial_chrono_start (self, trial, name: str, aggregator: Callable[[],
                        track.aggregators.aggregator.Aggregator] = <function StatAg-
                        gregator.lazy.<locals>.<lambda> at 0x7ff800c1a7b8>,
                        start_callback=None, end_callback=None)
```

Send the start signal for an event

#### Parameters

**trial: Trial** trial sending the event

**name: str** name of the event

**aggregator: Aggregator** container used to accumulate elapsed time

**start\_callback: Callable** function called at start time

**end\_callback: Callable** function called at the end

```
log_trial_finish (self, trial, exc_type, exc_val, exc_tb)
```

Send the trial end signal

#### Parameters

**trial: Trial** reference to the trial that finished

```
log_trial_metadata (self, trial: track.structure.Trial, aggregator: Callable[[],
                        track.aggregators.aggregator.Aggregator] = None, **kwargs)
```

Save metadata for a given trials

#### Parameters

**trial: Trial** trial reference

**kwargs:** key value pair of the data to save

```
log_trial_metrics (self, trial: track.structure.Trial, step: <built-in function any> = None,
                    aggregator: Callable[[], track.aggregators.aggregator.Aggregator] = None,
                    **kwargs)
```

Save metrics for a given trials

#### Parameters

**trial: Trial** trial reference

**kwargs:** key value pair of the data to save

```
log_trial_start (self, trial)
```

Send the trial start signal

#### Parameters

**trial: Trial** reference to the trial being started

```
new_project (self, project: track.structure.Project)
```

Insert a new project

#### Parameters

**project: Project** project definition used for the insert

```
new_trial (self, trial: track.structure.Trial)
```

Insert a new trial

**Parameters**

**trial: Trial** trial definition used for the insert

**auto\_increment: bool** If trial exist increment revision number

**Returns**

**Returns None if Trial already exists and auto\_increment is False**

**new\_trial\_group** (*self*, *group*: *track.structure.TrialGroup*)

Create a new group

**Parameters**

**group: TrialGroup** group definition used for the insert

**set\_trial\_status** (*self*, *trial*: *track.structure.Trial*, *status*, *error=None*)

Change trial status

**Parameters**

**trial: Trial** trial reference

**status:** new status to update the trial too

**error:** in case the user is changing to a state representing an error it can also provide an error identification string

**class** *track.persistence.socketed.SocketServer* (*uri*)

Bases: *track.persistence.protocol.Protocol*

Start a track server inside a asyncio loop

**Parameters**

**uri: str** socket://{hostname}:{port}?security\_layer={ }&backend={protocol} with

**Users inherit this class to implement their own custom authentication**

**Methods**

<code>add_group_trial(self, group, trial)</code>	Add a trial to a group
<code>add_project_trial(self, project, trial)</code>	Add a trial to a project
<code>add_trial_tags(self, trial, \**kwargs)</code>	Add tags to a trial
<code>authenticate(self, reader, username, password)</code>	User defined authentication function
<code>commit(self, \**kwargs)</code>	Forces to persist the change
<code>fetch_and_update_group(self, query, attr, ...)</code>	Fetch and update a single group
<code>fetch_and_update_trial(self, query, attr, ...)</code>	Fetch and update a single trial
<code>fetch_groups(self, query)</code>	Fetch groups according to a given query
<code>fetch_projects(self, query)</code>	Fetch projects according to a given query
<code>fetch_trials(self, query)</code>	Fetch trials according to a given query
<code>get_project(self, project)</code>	Fetch a project according to the given definition
<code>get_trial(self, trial)</code>	Fetch trials according to a given definition
<code>get_trial_group(self, group)</code>	Fetch a group according to a given definition
<code>log_trial_arguments(self, trial, \**kwargs)</code>	Save the arguments a trail
<code>log_trial_chrono_finish(self, trial, name, ...)</code>	Send the end signal for an event
<code>log_trial_chrono_start(self, trial, name, ...)</code>	Send the start signal for an event

Continued on next page



Table 10 – continued from previous page

<code>log_trial_finish(self, trial, exc_type, ...)</code>	Send the trial end signal
<code>log_trial_metadata(self, trial, aggregator, ...)</code>	Save metadata for a given trials
<code>log_trial_metrics(self, trial, step, ...)</code>	Save metrics for a given trials
<code>log_trial_start(self, trial)</code>	Send the trial start signal
<code>new_project(self, project)</code>	Insert a new project
<code>new_trial(self, trial[, auto_increment])</code>	Insert a new trial
<code>new_trial_group(self, group)</code>	Create a new group
<code>process_args(self, args[, cache])</code>	replace ids by their object reference so the backend modifies the objects and not a copy
<code>run_server(self)</code>	
<code>set_trial_status(self, trial, status[, error])</code>	Change trial status

<b>close</b>	
<b>close_connection</b>	
<b>exec</b>	
<b>get_username</b>	
<b>handle_client</b>	
<b>is_authenticated</b>	
<b>wait_closed</b>	

**authenticate** (*self*, *reader*, *username*, *password*)

User defined authentication function

#### Parameters

**reader:** **StreamReader** client socket / reader, can be used to link client socket -> username

**username:** **str** client username

**password:** **str** client password

**close** (*self*)

**static close\_connection** (*writer*)

**commit** (*self*, *\*\*kwargs*)

Forces to persist the change

**exec** (*self*, *reader*, *writer*, *proc\_name*, *proc*, *args*, *cache=None*)

**get\_username** (*self*, *reader*)

**handle\_client** (*self*, *reader*, *writer*)

**is\_authenticated** (*self*, *reader*)

**process\_args** (*self*, *args*, *cache=None*)

replace ids by their object reference so the backend modifies the objects and not a copy

**run\_server** (*self*)

**static wait\_closed** (*writer*)

`track.persistence.socketed.read(reader, timeout=None)`

`track.persistence.socketed.recv(socket, timeout=None)`

`track.persistence.socketed.send(socket, msg)`

```
track.persistence.socketed.start_track_server(protocol, hostname, port, security_layer=None)
```

Start a track server inside a asyncio loop

#### Parameters

**protocol: str** URI that defines which backend to forward the request to

**hostname: str** server host name

**port: int** server port to listen to

**security\_layer: str** backend used for encryption (only AES is supported)

```
track.persistence.socketed.to_bytes(message) → bytes
```

```
track.persistence.socketed.to_obj(message: bytes) → <built-in function any>
```

```
track.persistence.socketed.write(writer, msg)
```

### track.persistence.storage module

```
class track.persistence.storage.LocalStorage(target_file: str = None, _objects: Dict[uuid.UUID, <built-in function any>] = <factory>, _projects: Set[uuid.UUID] = <factory>, _groups: Set[uuid.UUID] = <factory>, _trials: Set[uuid.UUID] = <factory>, _project_names: Dict[str, uuid.UUID] = <factory>, _group_names: Dict[str, uuid.UUID] = <factory>, _trial_names: Dict[str, uuid.UUID] = <factory>, _old_rev_tags: Dict[str, int] = <factory>)
```

Bases: object

#### Attributes

**group\_names**

**groups**

**objects**

**project\_names**

**projects**

**target\_file**

**trials**

#### Methods

---

<code>reload(self[, filename])</code>	Reload storage and discard current objects
<code>smart_reload(self[, filename])</code>	Updates current objects with new data

---

<b>commit</b>	
<b>get_current_version_tag</b>	
<b>get_previous_version_tag</b>	

```

commit (self, file_name_override=None, **kwargs)
get_current_version_tag (self, obj)
get_previous_version_tag (self, obj)
group_names
groups
objects
project_names
projects
reload (self, filename=None)
    Reload storage and discard current objects
smart_reload (self, filename=None)
    Updates current objects with new data
target_file = None
trials
track.persistence.storage.load_database (json_name)

```

### track.persistence.utils module

```

track.persistence.utils.parse_options (options)
track.persistence.utils.parse_uri (uri)
    Parse a URI and returns a dictionary from it
    scheme:[//authority]path[?query][#fragment]          with          authority =
    [userinfo@]host[:port]

```

### Module contents

```

track.persistence.get_protocol (backend_name)
    proto://arg
track.persistence.make_cockroach_protocol (uri)
track.persistence.make_comet_ml (uri)
track.persistence.make_ephemeral_protocol (uri)
track.persistence.make_local (uri, strict=True, eager=True)
track.persistence.make_mongodb_protocol (uri)
track.persistence.make_pickled_protocol (uri)
track.persistence.make_socket_protocol (uri)
track.persistence.register (name, proto)

```

## track.utils package

### Submodules

#### track.utils.debug module

`track.utils.debug.print_stack(msg='-')`

#### track.utils.delay module

**class** `track.utils.delay.DelayedCall` (*fun, kwargs*)

Bases: `object`

Delay a call until later

#### Methods

---

<code>__call__(self, *args, **kwargs)</code>	Call self as a function.
--	--------------------------

---

<code>add_arguments</code>	
<code>get_future</code>	

`add_arguments` (*self, \*\*kwargs*)

`get_future` (*self*)

**class** `track.utils.delay.Future` (*promise*)

Bases: `object`

#### Methods

<code>get</code>	
<code>is_ready</code>	

`get` (*self*)

`is_ready` (*self*)

**exception** `track.utils.delay.FutureIsNotReady`

Bases: `Exception`

`track.utils.delay.delay_call` (*fun, \*\*kwargs*)

`track.utils.delay.is_delayed_call` (*obj*)

#### track.utils.encrypted module

**class** `track.utils.encrypted.EncryptedSocket` (*\*args, \*\*kwargs*)

Bases: `socket.socket`

Socket with an encrypted layer

### Attributes

**family** Read-only access to the address family for this socket.

**proto** the socket protocol

**timeout** the socket timeout

**type** Read-only access to the socket type.

### Methods

<code>accept(self)</code>	Accept an incoming connection & initialize the encryption layer for that client
<code>bind(address)</code>	Bind the socket to a local address.
<code>close()</code>	Close the socket.
<code>connect(address)</code>	Connect the socket to a remote address.
<code>connect_ex()</code>	This is like <code>connect(address)</code> , but returns an error code (the <code>errno</code> value) instead of raising an exception when an error occurs.
<code>detach(self)</code>	Close the socket object without closing the underlying file descriptor.
<code>dup(self)</code>	Duplicate the socket.
<code>fileno()</code>	Return the integer file descriptor of the socket.
<code>get_inheritable(self)</code>	Get the inheritable flag of the socket
<code>getblocking()</code>	Returns True if socket is in blocking mode, or False if it is in non-blocking mode.
<code>getpeername()</code>	Return the address of the remote endpoint.
<code>getsockname()</code>	Return the address of the local endpoint.
<code>getsockopt()</code>	Get a socket option.
<code>gettimeout()</code>	Returns the timeout in seconds (float) associated with socket operations.
<code>listen([backlog])</code>	Enable a server to accept connections.
<code>makefile(self[, mode, buffering, encoding, ...])</code>	The arguments are as for <code>io.open()</code> after the filename, except the only supported mode values are 'r' (default), 'w' and 'b'.
<code>recv(self, buffersize, flags[, context])</code>	Receive up to <code>buffersize</code> bytes from the socket.
<code>recv_into()</code>	A version of <code>recv()</code> that stores its data into a buffer rather than creating a new string.
<code>recvfrom(buffersize[, flags])</code>	Like <code>recv(buffersize, flags)</code> but also return the sender's address info.
<code>recvfrom_into(buffer[, nbytes[, flags]])</code>	Like <code>recv_into(buffer[, nbytes[, flags]])</code> but also return the sender's address info.
<code>recvmsg(bufsize[, ancbufsize[, flags]])</code>	Receive normal data (up to <code>bufsize</code> bytes) and ancillary data from the socket.
<code>recvmsg_into(buffers[, ancbufsize[, flags]])</code>	Receive normal data and ancillary data from the socket, scattering the non-ancillary data into a series of buffers.
<code>send(self, data, flags)</code>	Send a data string to the socket.
<code>sendall(data[, flags])</code>	Send a data string to the socket.

Continued on next page

Table 13 – continued from previous page

<code>sendfile(self, file[, offset, count])</code>	Send a file until EOF is reached by using high-performance <code>os.sendfile()</code> and return the total number of bytes which were sent.
<code>sendmsg()</code>	Send normal and ancillary data to the socket, gathering the non-ancillary data from a series of buffers and concatenating it into a single message.
<code>sendmsg_afalg([msg], *, op[, iv[, assoclen]])</code>	Set operation mode, IV and length of associated data for an AF_ALG operation socket.
<code>sendto()</code>	Like <code>send(data, flags)</code> but allows specifying the destination address.
<code>set_inheritable(self, inheritable)</code>	Set the inheritable flag of the socket
<code>setblocking(flag)</code>	Set the socket to blocking (flag is true) or non-blocking (false).
<code>setsockopt(level, option, value, option, ...)</code>	Set a socket option.
<code>settimeout(timeout)</code>	Set a timeout on socket operations.
<code>shutdown(flag)</code>	Shut down the reading side of the socket (flag == SHUT_RD), the writing side of the socket (flag == SHUT_WR), or both ends (flag == SHUT_RDWR).

readsize	
----------	--

**accept** (*self*)

Accept an incoming connection &amp; initialize the encryption layer for that client

**Returns****returns (socket, addr) of the client****readsize** (*self*)**recv** (*self*, *bufferize*, *flags*: *int* = 0, *context*=None)

Receive up to *bufferize* bytes from the socket. For the optional *flags* argument, see the Unix manual. When no data is available, block until at least one byte is available or until the remote end is closed. When the remote end is closed and all data is read, return the empty string.

**send** (*self*, *data*: *bytes*, *flags*: *int* = 0) → *int*

Send a data string to the socket. For the optional *flags* argument, see the Unix manual. Return the number of bytes sent; this may be less than `len(data)` if the network is busy.

**sendall** (*data*[, *flags* ])

Send a data string to the socket. For the optional *flags* argument, see the Unix manual. This calls `send()` repeatedly until all data is sent. If an error occurs, it's impossible to tell how much data has been sent.

`track.utils.encrypted.wrap_socket (sock, server_side=False, handshake=False)`**track.utils.eta module**

**class** `track.utils.eta.EstimatedTime` (*stat\_timer*: `track.utils.stat.StatStream`, *total*: `Union[int, List[int]]`, *start*: *int* = 0, *name*: *str* = None)

Bases: `object`

Compute estimated time to arrival given average time and remaining steps

## Examples

```
>>> timer = StatStream()
>>> total = (10, 1000)
>>> eta = EstimatedTime(timer, total)
>>> eta.estimate_time((1, 2))
```

## Attributes

**total**

## Methods

<code>count(item[, offset])</code>	Return the current iteration it given the completion of each steps
<code>elapsed(self, unit)</code>	Return the elapsed time since the class was created
<code>estimated_time(self, step, unit)</code>	Estimate the time remaining before the end of the computation
<code>set_totals(self, t)</code>	Set the total number of iteration for each step
<code>show_eta(self, step[, msg, show])</code>	Print the estimate time until the processing is done

**static count** (*item*, *offset=0*)

Return the current iteration it given the completion of each steps

**elapsed** (*self*, *unit: int = 60*)

Return the elapsed time since the class was created

**estimated\_time** (*self*, *step: int*, *unit: int = 60*)

Estimate the time remaining before the end of the computation

**set\_totals** (*self*, *t*)

Set the total number of iteration for each step

**show\_eta** (*self*, *step*, *msg=""*, *show=True*)

Print the estimate time until the processing is done

**total**

`track.utils.eta.get_time` (*time: track.utils.stat.StatStream*)

`track.utils.eta.to_list` (*item*)

## track.utils.log module

`track.utils.log.get_log_record_constructor` ()

`track.utils.log.make_logger` (*name*)

`track.utils.log.set_log_level` (*level=20*)

## track.utils.out module

**class** `track.utils.out.RingOutputDecorator` (*file=None*, *n\_entries=50*)

Bases: `object`

## Methods

<b>flush</b>	
<b>out</b>	
<b>output</b>	
<b>raw</b>	
<b>write</b>	

**flush** (*self*)  
**out** (*self*)  
**output** (*self*)  
**raw** (*self*)  
**write** (*self*, *string*)

## track.utils.signal module

**class** track.utils.signal.**SignalHandler**  
Bases: object

## Methods

<b>atexit</b>	
<b>sigint</b>	
<b>sigterm</b>	

**atexit** (*self*)  
**sigint** (*self*, *signum*, *frame*)  
**sigterm** (*self*, *signum*, *frame*)

## track.utils.stat module

**class** track.utils.stat.**StatStream** (*drop\_first\_obs=10*)  
Bases: object

Sharable object

Store the sum of the observations and the the sum of the observations squared The first few observations are discarded (usually slower than the rest)

The average and the standard deviation is computed at the user's request

In order to make the computation stable we store the first observation and subtract it to every other observations. The idea is if  $x \sim N(\mu, \sigma)$   $x - x_0$  and the sum of  $x - x_0$  should be close(r) to 0 allowing for greater precision; without that trick `var` was getting negative on some iteration.

## Attributes

**avg**



count  
current\_count  
current\_obs  
drop\_obs  
first\_obs  
max  
min  
sd  
sum  
sum\_sqr  
total  
val  
var

Methods

from_dict	
state_dict	
to_array	
to_dict	
to_json	
update	

avg  
count  
current\_count  
current\_obs  
drop\_obs  
first\_obs  
static from\_dict (data)  
max  
min  
sd  
state\_dict (self)  
sum  
sum\_sqr  
to\_array (self, transform=None)  
to\_dict (self)

```
    to_json (self)
    total
    update (self, val, weight=1)
    val
    var
class track.utils.stat.StatStreamStruct
    Bases: _ctypes.Structure
        Attributes
            current_count Structure/Union member
            current_obs Structure/Union member
            drop_obs Structure/Union member
            first_obs Structure/Union member
            max Structure/Union member
            min Structure/Union member
            sum Structure/Union member
            sum_sqr Structure/Union member
current_count
    Structure/Union member
current_obs
    Structure/Union member
drop_obs
    Structure/Union member
first_obs
    Structure/Union member
max
    Structure/Union member
min
    Structure/Union member
sum
    Structure/Union member
sum_sqr
    Structure/Union member
```

### track.utils.system module

```
track.utils.system.get_gpu_name()
```

## track.utils.throttle module

**class** track.utils.throttle.**ThrottleRepeatedCalls** (*fun: Callable[[A], R], every=10*)

Bases: object

Limit how often the function *fun* is called in number of times called

### Methods

<code>__call__(self, \*args, \*\*kwargs)</code>	Call self as a function.
---	--------------------------

**class** track.utils.throttle.**Throttler** (*fun: Callable[[A], R], throttle=1*)

Bases: object

Limit how often the function *fun* is called by calling it only every *throttle* time it has been called

### Methods

<code>__call__(self, \*args, \*\*kwargs)</code>	Call self as a function.
---	--------------------------

**class** track.utils.throttle.**TimeThrottler** (*fun: Callable[[A], R], every=10*)

Bases: object

Limit how often the function *fun* is called in seconds

### Methods

<code>__call__(self, \*args, \*\*kwargs)</code>	Call self as a function.
---	--------------------------

track.utils.throttle.**is\_throttled** (*fun: Callable[[~A], ~R]*) → bool

track.utils.throttle.**throttle\_repeated** (*fun: Callable[[~A], ~R], every=None*) →  
Callable[[~A], Union[~R, NoneType]]

track.utils.throttle.**throttled** (*fun: Callable[[~A], ~R], throttle=None, every=None*) →  
Callable[[~A], Union[~R, NoneType]]

## Module contents

**exception** track.utils.**ItemNotFound**

Bases: Exception

track.utils.**listen\_socket** (*add, port, backend=None*)

track.utils.**open\_socket** (*add, port, backend=None*)

## Submodules

## track.chrono module

```
class track.chrono.ChronoContext (acc: track.aggregators.aggregator.Aggregator,  
                                start_callback: Callable = None, end_callback: Callable =  
                                None)
```

Bases: object

Sync is a function that can be set to make the timer wait before ending. This is useful when timing async calls like cuda calls

## track.client module

```
class track.client.TrackClient (backend='none')
```

Bases: object

TrackClient. A client tracks a single Trial being ran

### Parameters

**backend:** str Storage backend to use

### Methods

<code>add_tags(self, \**kwargs)</code>	Insert tags to current trials
<code>get_arguments(self, args, ...[, show])</code>	See <code>log_arguments()</code> for possible arguments
<code>get_device()</code>	Helper function that returns a cuda device if available else a cpu
<code>log_arguments(self, args, ...[, show])</code>	Store the arguments that was used to run the trial.
<code>new_trial(self[, force])</code>	Create a new trial
<code>report(self[, short])</code>	Print a digest of the logged metrics
<code>save(self[, file_name_override])</code>	Saved logged metrics into a json file
<code>set_group(self, group, NoneType] = None, ...)</code>	Set or create a new group
<code>set_project(self, project, NoneType] = None, ...)</code>	Set or create a new project
<code>set_trial(self, trial, NoneType] = None, ...)</code>	Set a new trial
<code>set_version(self[, version])</code>	Compute the version tag from the function call stack.

<b>finish</b>	
<b>start</b>	

**add\_tags** (*self*, *\*\*kwargs*)

Insert tags to current trials

**finish** (*self*, *exc\_type=None*, *exc\_val=None*, *exc\_tb=None*)

**get\_arguments** (*self*, *args: Union[argparse.ArgumentParser, argparse.Namespace, Dict] = None*,  
*show=False*, *\*\*kwargs*) → *argparse.Namespace*  
See `log_arguments()` for possible arguments

**static get\_device** ()

Helper function that returns a cuda device if available else a cpu

**log\_arguments** (*self*, *args: Union[argparse.ArgumentParser, argparse.Namespace, Dict] = None*,  
*show=False*, *\*\*kwargs*) → *argparse.Namespace*  
Store the arguments that was used to run the trial.

**Parameters**

**args:** `Union[ArgumentParser, Namespace, Dict]` save up the trial's arguments

**show:** `bool` print the arguments on the command line

**kwargs** more trial's arguments

**Returns**

**returns the trial's arguments**

**new\_trial** (*self*, *force=False*, *\*\*kwargs*)  
Create a new trial

**Parameters**

**force:** `bool` by default once the trial is set it cannot be changed. use force to override this behaviour.

**kwargs:** See `Trial()` for possible arguments

**Returns**

**returns a trial logger**

**report** (*self*, *short=True*)  
Print a digest of the logged metrics

**save** (*self*, *file\_name\_override=None*)  
Saved logged metrics into a json file

**set\_group** (*self*, *group: Union[track.structure.TrialGroup, NoneType] = None*, *force: bool = False*, *get\_only: bool = False*, *\*\*kwargs*)  
Set or create a new group

**Parameters**

**group:** `Optional[TrialGroup]` project definition you can use to create or set the project

**force:** `bool` by default once the trial group is set it cannot be changed. use force to override this behaviour.

**get\_only:** `bool` if true does not insert the group if missing. default to false

**kwargs** arguments used to create a `TrialGroup` object if no `TrialGroup` object were provided. See `TrialGroup()` for possible arguments

**Returns**

**returns created trial group**

**set\_project** (*self*, *project: Union[track.structure.Project, NoneType] = None*, *force: bool = False*, *get\_only: bool = False*, *\*\*kwargs*)  
Set or create a new project

**Parameters**

**project:** `Optional[Project]` project definition you can use to create or set the project

**force:** `bool` by default once the project is set it cannot be changed. use force to override this behaviour.

**get\_only:** `bool` if true does not insert the project if missing. default to false

**kwargs** arguments used to create a `Project` object if no project object were provided See `Project()` for possible arguments

**Returns**

**returns created project**

**set\_trial** (*self*, *trial*: Union[track.structure.Trial, NoneType] = None, *force*: bool = False, *\*\*kwargs*)  
Set a new trial

**Parameters**

**trial**: Optional[Trial] project definition you can use to create or set the project

**force**: bool by default once the trial is set it cannot be changed. use force to override this behaviour.

**kwargs**: {uid, hash, revision} arguments used to create a Trial object if no Trial object were provided. You should specify *uid* or the pair (*hash*, *revision*). See [Trial\(\)](#) for possible arguments

**Returns**

**returns a trial logger**

**set\_version** (*self*, *version*=None, *version\_fun*: Callable[[], str] = None)  
Compute the version tag from the function call stack. Defaults to compute the hash of the executed file

**Parameters**

**version**: str version string you want to use for the trial

**version\_fun**: Callable[[], str] version function to call to set the trial version

**start** (*self*)

**exception** track.client.TrialDoesNotExist  
Bases: Exception

**track.configuration module**

track.configuration.**find\_configuration** (*file*=None)

track.configuration.**options** (*key*, *default*=<track.configuration.\_DefaultNone object at 0x7ff802a9de80>)

track.configuration.**reset\_configuration** ()

**track.logger module**

**class** track.logger.**LogSignalHandler** (*logger*)  
Bases: track.utils.signal.SignalHandler

**Methods**

<b>atexit</b>	
<b>sigint</b>	
<b>sigterm</b>	

**atexit** (*self*)

**sigint** (*self*, *signum*, *frame*)

**sigterm** (*self*, *signum*, *frame*)

**class** `track.logger.LoggerChronoContext` (*protocol*, *trial*, *acc=s*<{'avg': 0.0, 'min': inf, 'max': -inf, 'sd': 0.0, 'count': 1, 'unit': 's'}>, *name=None*, *\*\*kwargs*)

Bases: object

**class** `track.logger.TrialLogger` (*trial*: `track.structure.Trial`, *protocol*: `track.persistence.protocol.Protocol`)

Bases: object

Unified logger interface. This object should be created through the *TrackClient* interface

### Parameters

**trial: Trial** the trial that the logger modifies

**protocol: Protocol** the storage protocol used to persist the log calls

### Methods

<code>capture_output(self[, output_size])</code>	capture standard output
<code>chrono(self, name, aggregator, ...[, ...])</code>	Start a timer to measure the time spent in that block
<code>finish(self[, exc_type, exc_val, exc_tb])</code>	finish trial, record end time and set the trial status to completed or interrupted
<code>log_arguments(self, \**kwargs)</code>	log the trial arguments.
<code>log_metadata(self, aggregator, ...)</code>	insert metadata value inside a trial
<code>log_metrics(self, step, aggregator, ...)</code>	insert metrics values inside a trial
<code>set_status(self, status[, error])</code>	update trial status
<code>start(self)</code>	Start trial, records start time and set the trial status to running

<b>add_tags</b>	
<b>log_code</b>	
<b>log_directory</b>	
<b>log_file</b>	
<b>set_eta_total</b>	
<b>show_eta</b>	

**add\_tags** (*self*, *\*\*kwargs*)

**capture\_output** (*self*, *output\_size=50*)  
capture standard output

**chrono** (*self*, *name: str*, *aggregator: Callable[[], track.aggregators.aggregator.Aggregator]* = `<function StatAggregator.lazy.<locals>.<lambda> at 0x7ff802ac9ea0>`, *start\_callback=None*, *end\_callback=None*)

Start a timer to measure the time spent in that block

### Parameters

**name: str** name of the timer

**aggregator:** how to save the values, by default it uses the `StatAggregator` and only the mean, sd, max, min values are kept once the training is done

**start\_callback: Callable** function that is called once the timer starts

**end\_callback:** Callable function that is called once the timer ends

### Returns

returns a context manager that represents the timer

**finish** (*self*, *exc\_type=None*, *exc\_val=None*, *exc\_tb=None*)

finish trial, record end time and set the trial status to completed or interrupted

**log\_arguments** (*self*, *\*\*kwargs*)

log the trial arguments. This function has not effect if the trial was already created.

**log\_code** (*self*)

**log\_directory** (*self*, *name*, *recursive=False*)

**log\_file** (*self*, *file\_name*)

**log\_metadata** (*self*, *aggregator: Callable[[], track.aggregators.aggregator.Aggregator] = None*, *\*\*kwargs*)

insert metadata value inside a trial

### Parameters

**kwargs:** dictionary of metrics (metadata\_name: value)

**log\_metrics** (*self*, *step: <built-in function any> = None*, *aggregator: Callable[[], track.aggregators.aggregator.Aggregator] = None*, *\*\*kwargs*)

insert metrics values inside a trial

### Parameters

**step:** any a value representing a training step (could be epoch, timestamp, ...)

**kwargs:** dictionary of metrics (metric\_name: value)

**aggregator:** Optional[Callable[[], Aggregator]] how to store the values locally

**set\_eta\_total** (*self*, *t*)

**set\_status** (*self*, *status*, *error=None*)

update trial status

**show\_eta** (*self*, *step: int*, *timer: track.utils.stat.StatStream*, *msg: str = "**throttle=None*, *every=None*, *no\_print=False*)

**start** (*self*)

Start trial, records start time and set the trial status to running

## track.serialization module

**class** track.serialization.SerializerAspect

Bases: object

### Methods

<b>from_json</b>	
<b>to_json</b>	

**from\_json** (*self*, *obj*)

**to\_json** (*self*, *obj: <built-in function any>*, *short=False*)



```
class track.serialization.SerializerChronoContext
    Bases: track.serialization.SerializerAspect
```

### Methods

<b>from_json</b>	
<b>to_json</b>	

**to\_json** (*self*, *obj*: <built-in function any>, *short*=False)

```
class track.serialization.SerializerDatetime
    Bases: track.serialization.SerializerAspect
```

### Methods

<b>from_json</b>	
<b>to_json</b>	

**to\_json** (*self*, *obj*: *datetime.datetime*, *short*=False)

```
class track.serialization.SerializerProject
    Bases: track.serialization.SerializerAspect
```

### Methods

<b>from_json</b>	
<b>to_json</b>	

**from\_json** (*self*, *obj*)

**to\_json** (*self*, *obj*: *track.structure.Project*, *short*=False)

```
class track.serialization.SerializerStatStream
    Bases: track.serialization.SerializerAspect
```

### Methods

<b>from_json</b>	
<b>to_json</b>	

**from\_json** (*self*, *obj*, *short*=False)

```
class track.serialization.SerializerStatus
    Bases: track.serialization.SerializerAspect
```

## Methods

<b>from_json</b>	
<b>to_json</b>	

**to\_json** (*self*, *obj*: *track.structure.Status*, *short=False*)

**class** *track.serialization.SerializerTrial*  
Bases: *track.serialization.SerializerAspect*

## Methods

<b>from_json</b>	
<b>to_json</b>	

**from\_json** (*self*, *obj*)

**ignore\_meta** = {'\_last\_change', '\_update\_count', 'heartbeat'}

**ignore\_short** = {'uid', 'hash', 'dtype', 'project\_id', 'group\_id'}

**to\_json** (*self*, *obj*: *track.structure.Trial*, *short=False*)

**class** *track.serialization.SerializerTrialGroup*  
Bases: *track.serialization.SerializerAspect*

## Methods

<b>from_json</b>	
<b>maybe_unflatten</b>	
<b>to_json</b>	

**from\_json** (*self*, *obj*)

**static maybe\_unflatten** (*v*)

**to\_json** (*self*, *obj*: *track.structure.TrialGroup*, *short=False*)

**class** *track.serialization.SerializerUUID*  
Bases: *track.serialization.SerializerAspect*

## Methods

<b>from_json</b>	
<b>to_json</b>	

**to\_json** (*self*, *obj*: *uuid.UUID*, *short=False*)

*track.serialization.from\_json* (*obj*: *Dict[str, <built-in function any>]*, *dtype=None*) → *<built-in function any>*

*track.serialization.to\_json* (*k*: *<built-in function any>*, *short=False*)

## track.structure module

hold basic data type classes that all backends need to implement

```
class track.structure.CustomStatus (name, value)
```

Bases: object

### Attributes

**name**

**value**

**name**

**value**

```
class track.structure.Project (_uid: str = None, name: Union[str, NoneType] = None, descrip-
                             tion: Union[str, NoneType] = None, metadata: Dict[str, any] =
                             <factory>, groups: Set[track.structure.TrialGroup] = <factory>,
                             trials: Set[track.structure.Trial] = <factory>) → None
```

Bases: object

Set of Trial Groups & trials If projects define tags than all children inherit those tags. children cannot override the tag of a parent

### Attributes

**description**

**name**

**uid**

## Methods

compute_uid	
-------------	--

```
compute_uid (self) → str
```

```
description = None
```

```
name = None
```

```
uid
```

```
class track.structure.Status
```

Bases: enum.Enum

An enumeration.

```
Broken = 203
```

```
Completed = 302
```

```
CreatedGroup = 0
```

```
ErrorGroup = 200
```

```
Exception = 202
```

```
FinishedGroup = 300
```

```
Interrupted = 201
```

Running = 101

RunningGroup = 100

Suspended = 301

```
class track.structure.Trial(_hash: str = None, revision: int = 0, name: Union[str, NoneType]
    = None, description: Union[str, NoneType] = None, tags: Dict[str,
    any] = <factory>, version: Union[str, NoneType] = None, group_id:
    Union[int, NoneType] = None, project_id: Union[int, NoneType] =
    None, parameters: Dict[str, any] = <factory>, metadata: Dict[str,
    any] = <factory>, metrics: Dict[str, any] = <factory>, chronos:
    Dict[str, any] = <factory>, status: Union[track.structure.Status,
    NoneType] = <Status.CreatedGroup: 0>, errors: List[str] = <fac-
    tory>) → None
```

Bases: object

A single training run

#### Attributes

description

group\_id

hash

name

project\_id

uid

version

#### Methods

compute_hash	
--------------	--

`compute_hash(self) → str`

description = None

group\_id = None

hash

name = None

project\_id = None

revision = 0

status = 0

uid

version = None

```
class track.structure.TrialGroup (_uid: str = None, name: Union[str, NoneType] = None, de-
                                scription: Union[str, NoneType] = None, metadata: Dict[str,
                                any] = <factory>, trials: Set[track.structure.Trial] = <fac-
                                tory>, project_id: Union[int, NoneType] = None) → None
```

Bases: object

Namespace / Set of trials

#### Attributes

**description**

**name**

**project\_id**

**uid**

#### Methods

<b>compute_uid</b>	
--------------------	--

**compute\_uid** (*self*) → str

**description** = None

**name** = None

**project\_id** = None

**uid**

track.structure.get\_current\_project ()

track.structure.get\_current\_trial ()

track.structure.set\_current\_project (project)

track.structure.set\_current\_trial (trial)

track.structure.status (name=None, value=None)

#### track.versioning module

track.versioning.compute\_hash (\*args, \*\*kwargs)

track.versioning.compute\_version (files: List[str]) → str

track.versioning.default\_version\_hash ()

get the current stack frames and from the file compute the version

track.versioning.get\_file\_version (file\_name: str) → str

hash the file using sha256, used in combination with get\_git\_version to version non committed modifications

track.versioning.get\_git\_version (module) → Tuple[str, str]

track.versioning.is\_iterable (iterable)

## Module contents

**class** `track.TrackClient` (*backend='none'*)

Bases: `object`

`TrackClient`. A client tracks a single Trial being ran

### Parameters

**backend:** `str` Storage backend to use

### Methods

<code>add_tags(self, \*\*kwargs)</code>	Insert tags to current trials
<code>get_arguments(self, args, ...[, show])</code>	See <code>log_arguments()</code> for possible arguments
<code>get_device()</code>	Helper function that returns a cuda device if available else a cpu
<code>log_arguments(self, args, ...[, show])</code>	Store the arguments that was used to run the trial.
<code>new_trial(self[, force])</code>	Create a new trial
<code>report(self[, short])</code>	Print a digest of the logged metrics
<code>save(self[, file_name_override])</code>	Saved logged metrics into a json file
<code>set_group(self, group, NoneType] = None, ...)</code>	Set or create a new group
<code>set_project(self, project, NoneType] = None, ...)</code>	Set or create a new project
<code>set_trial(self, trial, NoneType] = None, ...)</code>	Set a new trial
<code>set_version(self[, version])</code>	Compute the version tag from the function call stack.

<b>finish</b>	
<b>start</b>	

**add\_tags** (*self*, *\\*\\*kwargs*)

Insert tags to current trials

**finish** (*self*, *exc\_type=None*, *exc\_val=None*, *exc\_tb=None*)

**get\_arguments** (*self*, *args: Union[argparse.ArgumentParser, argparse.Namespace, Dict] = None*, *show=False*, *\\*\\*kwargs*) → `argparse.Namespace`

See `log_arguments()` for possible arguments

**static get\_device** ()

Helper function that returns a cuda device if available else a cpu

**log\_arguments** (*self*, *args: Union[argparse.ArgumentParser, argparse.Namespace, Dict] = None*, *show=False*, *\\*\\*kwargs*) → `argparse.Namespace`

Store the arguments that was used to run the trial.

### Parameters

**args:** `Union[ArgumentParser, Namespace, Dict]` save up the trial's arguments

**show:** `bool` print the arguments on the command line

**kwargs** more trial's arguments

### Returns

returns the trial's arguments

**new\_trial** (*self*, *force=False*, *\*\*kwargs*)

Create a new trial

#### Parameters

**force: bool** by default once the trial is set it cannot be changed. use force to override this behaviour.

**kwargs:** See [Trial\(\)](#) for possible arguments

#### Returns

returns a trial logger

**report** (*self*, *short=True*)

Print a digest of the logged metrics

**save** (*self*, *file\_name\_override=None*)

Saved logged metrics into a json file

**set\_group** (*self*, *group: Union[track.structure.TrialGroup, NoneType] = None*, *force: bool = False*, *get\_only: bool = False*, *\*\*kwargs*)

Set or create a new group

#### Parameters

**group: Optional[TrialGroup]** project definition you can use to create or set the project

**force: bool** by default once the trial group is set it cannot be changed. use force to override this behaviour.

**get\_only: bool** if true does not insert the group if missing. default to false

**kwargs** arguments used to create a [TrialGroup](#) object if no TrialGroup object were provided. See [TrialGroup\(\)](#) for possible arguments

#### Returns

returns created trial group

**set\_project** (*self*, *project: Union[track.structure.Project, NoneType] = None*, *force: bool = False*, *get\_only: bool = False*, *\*\*kwargs*)

Set or create a new project

#### Parameters

**project: Optional[Project]** project definition you can use to create or set the project

**force: bool** by default once the project is set it cannot be changed. use force to override this behaviour.

**get\_only: bool** if true does not insert the project if missing. default to false

**kwargs** arguments used to create a [Project](#) object if no project object were provided See [Project\(\)](#) for possible arguments

#### Returns

returns created project

**set\_trial** (*self*, *trial: Union[track.structure.Trial, NoneType] = None*, *force: bool = False*, *\*\*kwargs*)

Set a new trial

#### Parameters

**trial: Optional[Trial]** project definition you can use to create or set the project

**force: bool** by default once the trial is set it cannot be changed. use force to override this behaviour.

**kwargs: {uid, hash, revision}** arguments used to create a *Trial* object if no *Trial* object were provided. You should specify *uid* or the pair (*hash*, *revision*). See [Trial\(\)](#) for possible arguments

### Returns

returns a trial logger

**set\_version** (*self*, *version=None*, *version\_fun: Callable[[], str] = None*)

Compute the version tag from the function call stack. Defaults to compute the hash of the executed file

### Parameters

**version: str** version string you want to use for the trial

**version\_fun: Callable[[], str]** version function to call to set the trial version

**start** (*self*)

```
class track.Project(_uid: str = None, name: Union[str, NoneType] = None, description:
                    Union[str, NoneType] = None, metadata: Dict[str, any] = <factory>, groups:
                    Set[track.structure.TrialGroup] = <factory>, trials: Set[track.structure.Trial] =
                    <factory>) → None
```

Bases: object

Set of Trial Groups & trials If projects define tags than all children inherit those tags. children cannot override the tag of a parent

### Attributes

**description**

**name**

**uid**

### Methods

compute_uid	
-------------	--

**compute\_uid** (*self*) → str

**description** = None

**name** = None

**uid**

```
class track.TrialGroup(_uid: str = None, name: Union[str, NoneType] = None, description:
                      Union[str, NoneType] = None, metadata: Dict[str, any] = <factory>, trials:
                      Set[track.structure.Trial] = <factory>, project_id: Union[int, NoneType] =
                      None) → None
```

Bases: object

Namespace / Set of trials

### Attributes

**description**



**name**  
**project\_id**  
**uid**

### Methods

<b>compute_uid</b>	
--------------------	--

**compute\_uid**(*self*) → str  
**description** = None  
**name** = None  
**project\_id** = None  
**uid**



### t

- [track](#), 50
- [track.aggregators](#), 15
- [track.aggregators.aggregator](#), 12
- [track.chrono](#), 40
- [track.client](#), 40
- [track.configuration](#), 42
- [track.containers](#), 15
- [track.containers.ring](#), 15
- [track.containers.types](#), 15
- [track.distributed](#), 16
- [track.distributed.cockroachdb](#), 15
- [track.logger](#), 42
- [track.persistence](#), 31
- [track.persistence.local](#), 17
- [track.persistence.multiplexer](#), 19
- [track.persistence.protocol](#), 21
- [track.persistence.socketed](#), 24
- [track.persistence.storage](#), 30
- [track.persistence.utils](#), 31
- [track.serialization](#), 44
- [track.structure](#), 47
- [track.utils](#), 39
- [track.utils.debug](#), 32
- [track.utils.delay](#), 32
- [track.utils.encrypted](#), 32
- [track.utils.eta](#), 34
- [track.utils.log](#), 35
- [track.utils.out](#), 35
- [track.utils.signal](#), 36
- [track.utils.stat](#), 36
- [track.utils.system](#), 38
- [track.utils.throttle](#), 39
- [track.versioning](#), 49



## A

- accept() (track.utils.encrypted.EncryptedSocket method), 34
- add\_arguments() (track.utils.delay.DelayedCall method), 32
- add\_group\_trial() (track.persistence.local.FileProtocol method), 17
- add\_group\_trial() (track.persistence.multiplexer.ProtocolMultiplexer method), 20
- add\_group\_trial() (track.persistence.protocol.Protocol method), 21
- add\_group\_trial() (track.persistence.socketed.SocketClient method), 26
- add\_project\_trial() (track.persistence.local.FileProtocol method), 17
- add\_project\_trial() (track.persistence.multiplexer.ProtocolMultiplexer method), 20
- add\_project\_trial() (track.persistence.protocol.Protocol method), 21
- add\_project\_trial() (track.persistence.socketed.SocketClient method), 26
- add\_tags() (track.client.TrackClient method), 40
- add\_tags() (track.logger.TrialLogger method), 43
- add\_tags() (track.TrackClient method), 50
- add\_trial\_tags() (track.persistence.local.FileProtocol method), 17
- add\_trial\_tags() (track.persistence.multiplexer.ProtocolMultiplexer method), 20
- add\_trial\_tags() (track.persistence.protocol.Protocol method), 21
- add\_trial\_tags() (track.persistence.socketed.SocketClient method), 26
- Aggregator (class in track.aggregators.aggregator), 12
- append() (track.aggregators.aggregator.Aggregator method), 12
- append() (track.aggregators.aggregator.RingAggregator method), 13
- append() (track.aggregators.aggregator.StatAggregator method), 13
- append() (track.aggregators.aggregator.TimeSeriesAggregator method), 14
- append() (track.aggregators.aggregator.ValueAggregator method), 14
- append() (track.containers.ring.RingBuffer method), 15
- atexit() (track.logger.LogSignalHandler method), 42
- atexit() (track.persistence.local.LockFileRemover method), 18
- atexit() (track.utils.signal.SignalHandler method), 36
- authenticate() (track.persistence.socketed.SocketClient method), 26
- authenticate() (track.persistence.socketed.SocketServer method), 29
- avg (track.aggregators.aggregator.StatAggregator attribute), 13
- avg (track.utils.stat.StatStream attribute), 37

## B

- Broken (track.structure.Status attribute), 47
- build (track.distributed.cockroachdb.CockroachDB attribute), 16

## C

- capture\_output() (track.logger.TrialLogger method), 43
- chrono() (track.logger.TrialLogger method), 43
- ChronoContext (class in track.chrono), 40
- client\_flags (track.distributed.cockroachdb.CockroachDB attribute), 16
- close() (track.persistence.socketed.SocketServer method), 29
- close\_connection() (track.persistence.socketed.SocketServer static method), 29
- CockroachDB (class in track.distributed.cockroachdb), 15
- commit() (track.persistence.local.FileProtocol method), 17
- commit() (track.persistence.multiplexer.ProtocolMultiplexer method), 20
- commit() (track.persistence.protocol.Protocol method), 21

`commit()` (track.persistence.socketed.SocketClient method), 26  
`commit()` (track.persistence.socketed.SocketServer method), 29  
`commit()` (track.persistence.storage.LocalStorage method), 31  
`Completed` (track.structure.Status attribute), 47  
`compute_hash()` (in module track.versioning), 49  
`compute_hash()` (track.structure.Trial method), 48  
`compute_uid()` (track.Project method), 52  
`compute_uid()` (track.structure.Project method), 47  
`compute_uid()` (track.structure.TrialGroup method), 49  
`compute_uid()` (track.TrialGroup method), 53  
`compute_version()` (in module track.versioning), 49  
`ConcurrentWrite`, 17  
`count` (track.utils.stat.StatStream attribute), 37  
`count()` (track.utils.eta.EstimatedTime static method), 35  
`CreatedGroup` (track.structure.Status attribute), 47  
`current_count` (track.utils.stat.StatStream attribute), 37  
`current_count` (track.utils.stat.StatStreamStruct attribute), 38  
`current_obs` (track.utils.stat.StatStream attribute), 37  
`current_obs` (track.utils.stat.StatStreamStruct attribute), 38  
`CustomStatus` (class in track.structure), 47

## D

`default_version_hash()` (in module track.versioning), 49  
`delay_call()` (in module track.utils.delay), 32  
`DelayedCall` (class in track.utils.delay), 32  
`description` (track.Project attribute), 52  
`description` (track.structure.Project attribute), 47  
`description` (track.structure.Trial attribute), 48  
`description` (track.structure.TrialGroup attribute), 49  
`description` (track.TrialGroup attribute), 53  
`drop_obs` (track.utils.stat.StatStream attribute), 37  
`drop_obs` (track.utils.stat.StatStreamStruct attribute), 38

## E

`elapsed()` (track.utils.eta.EstimatedTime method), 35  
`EncryptedSocket` (class in track.utils.encrypted), 32  
`ErrorGroup` (track.structure.Status attribute), 47  
`estimated_time()` (track.utils.eta.EstimatedTime method), 35  
`EstimatedTime` (class in track.utils.eta), 34  
`Exception` (track.structure.Status attribute), 47  
`exec()` (track.persistence.socketed.SocketServer method), 29  
`execute_query()` (in module track.persistence.local), 18

## F

`fetch_and_update_group()` (track.persistence.local.FileProtocol method), 18  
`fetch_and_update_group()` (track.persistence.multiplexer.ProtocolMultiplexer method), 20  
`fetch_and_update_group()` (track.persistence.protocol.Protocol method), 22  
`fetch_and_update_trial()` (track.persistence.local.FileProtocol method), 18  
`fetch_and_update_trial()` (track.persistence.multiplexer.ProtocolMultiplexer method), 20  
`fetch_and_update_trial()` (track.persistence.protocol.Protocol method), 22  
`fetch_groups()` (track.persistence.local.FileProtocol method), 18  
`fetch_groups()` (track.persistence.multiplexer.ProtocolMultiplexer method), 20  
`fetch_groups()` (track.persistence.protocol.Protocol method), 22  
`fetch_projects()` (track.persistence.local.FileProtocol method), 18  
`fetch_projects()` (track.persistence.multiplexer.ProtocolMultiplexer method), 20  
`fetch_projects()` (track.persistence.protocol.Protocol method), 22  
`fetch_trials()` (track.persistence.local.FileProtocol method), 18  
`fetch_trials()` (track.persistence.multiplexer.ProtocolMultiplexer method), 20  
`fetch_trials()` (track.persistence.protocol.Protocol method), 22  
`FileProtocol` (class in track.persistence.local), 17  
`find_configuration()` (in module track.configuration), 42  
`finish()` (track.client.TrackClient method), 40  
`finish()` (track.logger.TrialLogger method), 44  
`finish()` (track.TrackClient method), 50  
`FinishedGroup` (track.structure.Status attribute), 47  
`first_obs` (track.utils.stat.StatStream attribute), 37  
`first_obs` (track.utils.stat.StatStreamStruct attribute), 38  
`flush()` (track.utils.out.RingOutputDecorator method), 36  
`from_dict()` (track.utils.stat.StatStream static method), 37  
`from_json()` (in module track.serialization), 46  
`from_json()` (track.aggregators.aggregator.StatAggregator static method), 13  
`from_json()` (track.serialization.SerializerAspect method), 44  
`from_json()` (track.serialization.SerializerProject method), 45  
`from_json()` (track.serialization.SerializerStatStream method), 45  
`from_json()` (track.serialization.SerializerTrial method), 46  
`from_json()` (track.serialization.SerializerTrialGroup method), 46  
`Future` (class in track.utils.delay), 32

FutureIsNotReady, 32

## G

get() (track.utils.delay.Future method), 32  
 get\_arguments() (track.client.TrackClient method), 40  
 get\_arguments() (track.TrackClient method), 50  
 get\_current\_project() (in module track.structure), 49  
 get\_current\_trial() (in module track.structure), 49  
 get\_current\_version\_tag()  
   (track.persistence.storage.LocalStorage  
   method), 31  
 get\_device() (track.client.TrackClient static method), 40  
 get\_device() (track.TrackClient static method), 50  
 get\_file\_version() (in module track.versioning), 49  
 get\_future() (track.utils.delay.DelayedCall method), 32  
 get\_git\_version() (in module track.versioning), 49  
 get\_gpu\_name() (in module track.utils.system), 38  
 get\_log\_record\_constructor() (in module track.utils.log),  
   35  
 get\_previous\_version\_tag()  
   (track.persistence.storage.LocalStorage  
   method), 31  
 get\_project() (track.persistence.local.FileProtocol  
   method), 18  
 get\_project() (track.persistence.multiplexer.ProtocolMultiplexer  
   method), 20  
 get\_project() (track.persistence.protocol.Protocol  
   method), 22  
 get\_project() (track.persistence.socketed.SocketClient  
   method), 26  
 get\_protocol() (in module track.persistence), 31  
 get\_time() (in module track.utils.eta), 35  
 get\_trial() (track.persistence.local.FileProtocol method),  
   18  
 get\_trial() (track.persistence.multiplexer.ProtocolMultiplexer  
   method), 20  
 get\_trial() (track.persistence.protocol.Protocol method),  
   22  
 get\_trial() (track.persistence.socketed.SocketClient  
   method), 26  
 get\_trial\_group() (track.persistence.local.FileProtocol  
   method), 18  
 get\_trial\_group() (track.persistence.multiplexer.ProtocolMultiplexer  
   method), 20  
 get\_trial\_group() (track.persistence.protocol.Protocol  
   method), 22  
 get\_trial\_group() (track.persistence.socketed.SocketClient  
   method), 26  
 get\_username() (track.persistence.socketed.SocketServer  
   method), 29  
 group\_id (track.structure.Trial attribute), 48  
 group\_names (track.persistence.storage.LocalStorage at-  
   tribute), 31

groups (track.persistence.storage.LocalStorage attribute),  
 31

## H

handle\_client() (track.persistence.socketed.SocketServer  
 method), 29  
 hash (track.structure.Trial attribute), 48

## I

ignore\_meta (track.serialization.SerializerTrial attribute),  
 46  
 ignore\_short (track.serialization.SerializerTrial attribute),  
 46  
 Interrupted (track.structure.Status attribute), 47  
 is\_authenticated() (track.persistence.socketed.SocketServer  
   method), 29  
 is\_delayed\_call() (in module track.utils.delay), 32  
 is\_iterable() (in module track.versioning), 49  
 is\_ready() (track.utils.delay.Future method), 32  
 is\_throttled() (in module track.utils.throttle), 39  
 ItemNotFound, 39

## L

last() (track.containers.ring.RingBuffer method), 15  
 lazy() (track.aggregators.aggregator.Aggregator static  
   method), 12  
 lazy() (track.aggregators.aggregator.RingAggregator  
   static method), 13  
 lazy() (track.aggregators.aggregator.StatAggregator static  
   method), 13  
 lazy() (track.aggregators.aggregator.TimeSeriesAggregator  
   static method), 14  
 lazy() (track.aggregators.aggregator.ValueAggregator  
   static method), 14  
 listen\_socket() (in module track.utils), 39  
 load\_database() (in module track.persistence.storage), 31  
 LocalStorage (class in track.persistence.storage), 30  
 lock\_atomic\_write() (in module track.persistence.local),  
   19  
 lock\_guard() (in module track.persistence.local), 19  
 lock\_read() (in module track.persistence.local), 19  
 lock\_write() (in module track.persistence.local), 19  
 LocalFileRemover (class in track.persistence.local), 18  
 log\_arguments() (track.client.TrackClient method), 40  
 log\_arguments() (track.logger.TrialLogger method), 44  
 log\_arguments() (track.TrackClient method), 50  
 log\_code() (track.logger.TrialLogger method), 44  
 log\_directory() (track.logger.TrialLogger method), 44  
 log\_file() (track.logger.TrialLogger method), 44  
 log\_metadata() (track.logger.TrialLogger method), 44  
 log\_metrics() (track.logger.TrialLogger method), 44  
 log\_trial\_arguments() (track.persistence.local.FileProtocol  
   method), 18

[log\\_trial\\_arguments\(\)](#) (track.persistence.multiplexer.ProtocolMultiplexer method), 20  
[log\\_trial\\_arguments\(\)](#) (track.persistence.protocol.ProtocolMultiplexer method), 23  
[log\\_trial\\_arguments\(\)](#) (track.persistence.socketed.SocketClient method), 26  
[log\\_trial\\_chrono\\_finish\(\)](#) (track.persistence.local.FileProtocol method), 18  
[log\\_trial\\_chrono\\_finish\(\)](#) (track.persistence.multiplexer.ProtocolMultiplexer method), 20  
[log\\_trial\\_chrono\\_finish\(\)](#) (track.persistence.protocol.Protocol method), 23  
[log\\_trial\\_chrono\\_finish\(\)](#) (track.persistence.socketed.SocketClient method), 26  
[log\\_trial\\_chrono\\_start\(\)](#) (track.persistence.local.FileProtocol method), 18  
[log\\_trial\\_chrono\\_start\(\)](#) (track.persistence.multiplexer.ProtocolMultiplexer method), 20  
[log\\_trial\\_chrono\\_start\(\)](#) (track.persistence.protocol.Protocol method), 23  
[log\\_trial\\_chrono\\_start\(\)](#) (track.persistence.socketed.SocketClient method), 27  
[log\\_trial\\_finish\(\)](#) (track.persistence.local.FileProtocol method), 18  
[log\\_trial\\_finish\(\)](#) (track.persistence.multiplexer.ProtocolMultiplexer method), 20  
[log\\_trial\\_finish\(\)](#) (track.persistence.protocol.Protocol method), 23  
[log\\_trial\\_finish\(\)](#) (track.persistence.socketed.SocketClient method), 27  
[log\\_trial\\_metadata\(\)](#) (track.persistence.local.FileProtocol method), 18  
[log\\_trial\\_metadata\(\)](#) (track.persistence.multiplexer.ProtocolMultiplexer method), 20  
[log\\_trial\\_metadata\(\)](#) (track.persistence.protocol.Protocol method), 23  
[log\\_trial\\_metadata\(\)](#) (track.persistence.socketed.SocketClient method), 27  
[log\\_trial\\_metrics\(\)](#) (track.persistence.local.FileProtocol method), 18  
[log\\_trial\\_metrics\(\)](#) (track.persistence.multiplexer.ProtocolMultiplexer method), 20  
[log\\_trial\\_metrics\(\)](#) (track.persistence.protocol.Protocol method), 23  
[log\\_trial\\_metrics\(\)](#) (track.persistence.socketed.SocketClient method), 27  
[log\\_trial\\_start\(\)](#) (track.persistence.local.FileProtocol method), 18  
[log\\_trial\\_start\(\)](#) (track.persistence.multiplexer.ProtocolMultiplexer method), 20

[MultiplexStart](#) (track.persistence.protocol.Protocol method), 24  
[log\\_trial\\_start\(\)](#) (track.persistence.socketed.SocketClient method), 27  
[LoggerChronoContext](#) (class in track.logger), 43  
[LogSignalHandler](#) (class in track.logger), 42

## M

[make\\_cockroach\\_protocol\(\)](#) (in module track.persistence), 31  
[make\\_comet\\_ml\(\)](#) (in module track.persistence), 31  
[make\\_ephemeral\\_protocol\(\)](#) (in module track.persistence), 31  
[make\\_local\(\)](#) (in module track.persistence), 31  
[make\\_lock\(\)](#) (in module track.persistence.local), 19  
[make\\_logger\(\)](#) (in module track.utils.log), 35  
[make\\_mongodb\\_protocol\(\)](#) (in module track.persistence), 31  
[make\\_pickled\\_protocol\(\)](#) (in module track.persistence), 31  
[make\\_socket\\_protocol\(\)](#) (in module track.persistence), 31  
[max](#) (track.aggregators.aggregator.StatAggregator attribute), 13  
[max](#) (track.utils.stat.StatStream attribute), 37  
[max](#) (track.utils.stat.StatStreamStruct attribute), 38  
[maybe\\_unflatten\(\)](#) (track.serialization.SerializerTrialGroup static method), 46  
[min](#) (track.aggregators.aggregator.StatAggregator attribute), 13  
[min](#) (track.utils.stat.StatStream attribute), 37  
[min](#) (track.utils.stat.StatStreamStruct attribute), 38  
[MultiLock](#) (class in track.persistence.local), 18

## N

[name](#) (track.Project attribute), 52  
[name](#) (track.structure.CustomStatus attribute), 47  
[name](#) (track.structure.Project attribute), 47  
[name](#) (track.structure.Trial attribute), 48  
[name](#) (track.structure.TrialGroup attribute), 49  
[name](#) (track.TrialGroup attribute), 53  
[new\\_project\(\)](#) (track.persistence.local.FileProtocol method), 18  
[new\\_project\(\)](#) (track.persistence.multiplexer.ProtocolMultiplexer method), 20  
[new\\_project\(\)](#) (track.persistence.protocol.Protocol method), 24  
[new\\_project\(\)](#) (track.persistence.socketed.SocketClient method), 27  
[new\\_trial\(\)](#) (track.client.TrackClient method), 41  
[new\\_trial\(\)](#) (track.persistence.local.FileProtocol method), 18  
[new\\_trial\(\)](#) (track.persistence.multiplexer.ProtocolMultiplexer method), 21



- new\_trial() (track.persistence.protocol.Protocol method), 24
- new\_trial() (track.persistence.socketed.SocketClient method), 27
- new\_trial() (track.TrackClient method), 50
- new\_trial\_group() (track.persistence.local.FileProtocol method), 18
- new\_trial\_group() (track.persistence.multiplexer.ProtocolMultiplexer method), 21
- new\_trial\_group() (track.persistence.protocol.Protocol method), 24
- new\_trial\_group() (track.persistence.socketed.SocketClient method), 28
- node\_id (track.distributed.cockroachdb.CockroachDB attribute), 16
- ## O
- objects (track.persistence.storage.LocalStorage attribute), 31
- open\_socket() (in module track.utils), 39
- options() (in module track.configuration), 42
- out() (track.utils.out.RingOutputDecorator method), 36
- output() (track.utils.out.RingOutputDecorator method), 36
- ## P
- parse() (track.distributed.cockroachdb.CockroachDB method), 16
- parse\_options() (in module track.persistence.utils), 31
- parse\_uri() (in module track.persistence.utils), 31
- print\_stack() (in module track.utils.debug), 32
- process\_args() (track.persistence.socketed.SocketServer method), 29
- Project (class in track), 52
- Project (class in track.structure), 47
- project\_id (track.structure.Trial attribute), 48
- project\_id (track.structure.TrialGroup attribute), 49
- project\_id (track.TrialGroup attribute), 53
- project\_names (track.persistence.storage.LocalStorage attribute), 31
- projects (track.persistence.storage.LocalStorage attribute), 31
- Protocol (class in track.persistence.protocol), 21
- ProtocolMultiplexer (class in track.persistence.multiplexer), 19
- ## Q
- query\_gt() (in module track.persistence.local), 19
- query\_in() (in module track.persistence.local), 19
- query\_lte() (in module track.persistence.local), 19
- query\_ne() (in module track.persistence.local), 19
- ## R
- raw() (track.utils.out.RingOutputDecorator method), 36
- read() (in module track.persistence.socketed), 29
- readsize() (track.utils.encrypted.EncryptedSocket method), 34
- recv() (in module track.persistence.socketed), 29
- recv() (track.utils.encrypted.EncryptedSocket method), 34
- register() (in module track.persistence), 31
- remove() (track.persistence.storage.LocalStorage method), 31
- remove() (track.persistence.local.LockFileRemover method), 18
- report() (track.client.TrackClient method), 41
- report() (track.TrackClient method), 51
- reset\_configuration() (in module track.configuration), 42
- revision (track.structure.Trial attribute), 48
- RingAggregator (class in track.aggregators.aggregator), 12
- RingBuffer (class in track.containers.ring), 15
- RingOutputDecorator (class in track.utils.out), 35
- RPCCallFailure, 24
- run\_server() (track.persistence.socketed.SocketServer method), 29
- Running (track.structure.Status attribute), 47
- RunningGroup (track.structure.Status attribute), 48
- ## S
- save() (track.client.TrackClient method), 41
- save() (track.TrackClient method), 51
- sd (track.aggregators.aggregator.StatAggregator attribute), 13
- sd (track.utils.stat.StatStream attribute), 37
- send() (in module track.persistence.socketed), 29
- send() (track.utils.encrypted.EncryptedSocket method), 34
- sendall() (track.utils.encrypted.EncryptedSocket method), 34
- SerializerAspect (class in track.serialization), 44
- SerializerChronoContext (class in track.serialization), 44
- SerializerDatetime (class in track.serialization), 45
- SerializerProject (class in track.serialization), 45
- SerializerStatStream (class in track.serialization), 45
- SerializerStatus (class in track.serialization), 45
- SerializerTrial (class in track.serialization), 46
- SerializerTrialGroup (class in track.serialization), 46
- SerializerUUID (class in track.serialization), 46
- ServerSignalHandler (class in track.persistence.socketed), 24
- set\_current\_project() (in module track.structure), 49
- set\_current\_trial() (in module track.structure), 49
- set\_eta\_total() (track.logger.TrialLogger method), 44
- set\_group() (track.client.TrackClient method), 41
- set\_group() (track.TrackClient method), 51
- set\_group\_metadata() (track.persistence.local.FileProtocol method), 18

- set\_log\_level() (in module track.utils.log), 35
  - set\_project() (track.client.TrackClient method), 41
  - set\_project() (track.TrackClient method), 51
  - set\_status() (track.logger.TrialLogger method), 44
  - set\_totals() (track.utils.eta.EstimatedTime method), 35
  - set\_trial() (track.client.TrackClient method), 42
  - set\_trial() (track.TrackClient method), 51
  - set\_trial\_status() (track.persistence.local.FileProtocol method), 18
  - set\_trial\_status() (track.persistence.multiplexer.ProtocolMultiplexer method), 21
  - set\_trial\_status() (track.persistence.protocol.Protocol method), 24
  - set\_trial\_status() (track.persistence.socketed.SocketClient method), 28
  - set\_version() (track.client.TrackClient method), 42
  - set\_version() (track.TrackClient method), 52
  - show\_eta() (track.logger.TrialLogger method), 44
  - show\_eta() (track.utils.eta.EstimatedTime method), 35
  - sigint() (track.logger.LogSignalHandler method), 42
  - sigint() (track.persistence.local.LockFileRemover method), 18
  - sigint() (track.persistence.socketed.ServerSignalHandler method), 25
  - sigint() (track.utils.signal.SignalHandler method), 36
  - SignalHandler (class in track.utils.signal), 36
  - sigterm() (track.logger.LogSignalHandler method), 42
  - sigterm() (track.persistence.local.LockFileRemover method), 18
  - sigterm() (track.persistence.socketed.ServerSignalHandler method), 25
  - sigterm() (track.utils.signal.SignalHandler method), 36
  - smart\_reload() (track.persistence.storage.LocalStorage method), 31
  - SocketClient (class in track.persistence.socketed), 25
  - SocketServer (class in track.persistence.socketed), 28
  - sql (track.distributed.cockroachdb.CockroachDB attribute), 16
  - start() (track.client.TrackClient method), 42
  - start() (track.distributed.cockroachdb.CockroachDB method), 16
  - start() (track.logger.TrialLogger method), 44
  - start() (track.TrackClient method), 52
  - start\_track\_server() (in module track.persistence.socketed), 29
  - StatAggregator (class in track.aggregators.aggregator), 13
  - state\_dict() (track.utils.stat.StatStream method), 37
  - StatStream (class in track.utils.stat), 36
  - StatStreamStruct (class in track.utils.stat), 38
  - Status (class in track.structure), 47
  - status (track.distributed.cockroachdb.CockroachDB attribute), 16
  - status (track.structure.Trial attribute), 48
  - status() (in module track.structure), 49
  - stop() (track.distributed.cockroachdb.CockroachDB method), 16
  - sum (track.aggregators.aggregator.StatAggregator attribute), 13
  - sum (track.utils.stat.StatStream attribute), 37
  - sum (track.utils.stat.StatStreamStruct attribute), 38
  - sum\_sqr (track.utils.stat.StatStream attribute), 37
  - sum\_sqr (track.utils.stat.StatStreamStruct attribute), 38
  - Suspended (track.structure.Status attribute), 48
- ## T
- target\_file (track.persistence.storage.LocalStorage attribute), 31
  - throttle\_repeated() (in module track.utils.throttle), 39
  - throttled() (in module track.utils.throttle), 39
  - Throttler (class in track.utils.throttle), 39
  - ThrottleRepeatedCalls (class in track.utils.throttle), 39
  - TimeSeriesAggregator (class in track.aggregators.aggregator), 14
  - TimeThrottler (class in track.utils.throttle), 39
  - to\_array() (track.utils.stat.StatStream method), 37
  - to\_bytes() (in module track.persistence.socketed), 30
  - to\_dict() (track.utils.stat.StatStream method), 37
  - to\_json() (in module track.serialization), 46
  - to\_json() (track.aggregators.aggregator.Aggregator method), 12
  - to\_json() (track.aggregators.aggregator.RingAggregator method), 13
  - to\_json() (track.aggregators.aggregator.StatAggregator method), 13
  - to\_json() (track.aggregators.aggregator.TimeSeriesAggregator method), 14
  - to\_json() (track.aggregators.aggregator.ValueAggregator method), 14
  - to\_json() (track.serialization.SerializerAspect method), 44
  - to\_json() (track.serialization.SerializerChronoContext method), 45
  - to\_json() (track.serialization.SerializerDatetime method), 45
  - to\_json() (track.serialization.SerializerProject method), 45
  - to\_json() (track.serialization.SerializerStatus method), 46
  - to\_json() (track.serialization.SerializerTrial method), 46
  - to\_json() (track.serialization.SerializerTrialGroup method), 46
  - to\_json() (track.serialization.SerializerUUID method), 46
  - to\_json() (track.utils.stat.StatStream method), 37
  - to\_list() (in module track.utils.eta), 35
  - to\_list() (track.containers.ring.RingBuffer method), 15
  - to\_obj() (in module track.persistence.socketed), 30
  - total (track.aggregators.aggregator.StatAggregator attribute), 14
  - total (track.utils.eta.EstimatedTime attribute), 35

total (track.utils.stat.StatStream attribute), 38  
 track (module), 50  
 track.aggregators (module), 15  
 track.aggregators.aggregator (module), 12  
 track.chrono (module), 40  
 track.client (module), 40  
 track.configuration (module), 42  
 track.containers (module), 15  
 track.containers.ring (module), 15  
 track.containers.types (module), 15  
 track.distributed (module), 16  
 track.distributed.cockroachdb (module), 15  
 track.logger (module), 42  
 track.persistence (module), 31  
 track.persistence.local (module), 17  
 track.persistence.multiplexer (module), 19  
 track.persistence.protocol (module), 21  
 track.persistence.socketed (module), 24  
 track.persistence.storage (module), 30  
 track.persistence.utils (module), 31  
 track.serialization (module), 44  
 track.structure (module), 47  
 track.utils (module), 39  
 track.utils.debug (module), 32  
 track.utils.delay (module), 32  
 track.utils.encrypted (module), 32  
 track.utils.eta (module), 34  
 track.utils.log (module), 35  
 track.utils.out (module), 35  
 track.utils.signal (module), 36  
 track.utils.stat (module), 36  
 track.utils.system (module), 38  
 track.utils.throttle (module), 39  
 track.versioning (module), 49  
 TrackClient (class in track), 50  
 TrackClient (class in track.client), 40  
 Trial (class in track.structure), 48  
 TrialDoesNotExist, 42  
 TrialGroup (class in track), 52  
 TrialGroup (class in track.structure), 48  
 TrialLogger (class in track.logger), 43  
 trials (track.persistence.storage.LocalStorage attribute), 31

## U

uid (track.Project attribute), 52  
 uid (track.structure.Project attribute), 47  
 uid (track.structure.Trial attribute), 48  
 uid (track.structure.TrialGroup attribute), 49  
 uid (track.TrialGroup attribute), 53  
 update() (track.utils.stat.StatStream method), 38

## V

val (track.aggregators.aggregator.Aggregator attribute),

12  
 val (track.aggregators.aggregator.RingAggregator attribute), 13  
 val (track.aggregators.aggregator.StatAggregator attribute), 14  
 val (track.aggregators.aggregator.TimeSeriesAggregator attribute), 14  
 val (track.aggregators.aggregator.ValueAggregator attribute), 14  
 val (track.utils.stat.StatStream attribute), 38  
 value (track.structure.CustomStatus attribute), 47  
 ValueAggregator (class in track.aggregators.aggregator), 14  
 var (track.utils.stat.StatStream attribute), 38  
 version (track.structure.Trial attribute), 48

## W

wait() (track.distributed.cockroachdb.CockroachDB method), 16  
 wait\_closed() (track.persistence.socketed.SocketServer static method), 29  
 webui (track.distributed.cockroachdb.CockroachDB attribute), 16  
 wrap\_socket() (in module track.utils.encrypted), 34  
 write() (in module track.persistence.socketed), 30  
 write() (track.utils.out.RingOutputDecorator method), 36