pycoalaip Documentation

Release 0.0.3

BigchainDB

Nov 26, 2018

Contents

Important: Development Status: Alpha

Contents:

CHAPTER 1

pycoalaip

Python reference implementation for COALA IP.

- Development Status: Alpha
- Free software: Apache Software License 2.0
- Documentation: https://pycoalaip.readthedocs.io

1.1 Features

- `CoalaIp.generate_user()`: Create a user representation suitable for use with `coalaip`
- `CoalaIp.register_manifestation()`: Registering a `Manifestation` (and along with it, an associated parent `Work` and a `Copyright` of the `Manifestation`)
- `CoalaIp.derive_right()`: Derivation of a `Right` from an allowing source `Right` or `Copyright`
- `CoalaIp.transfer_right()`: Transfer of a `Right` or `Copyright` from the current owner to a new owner
- Querying the ownership history of an COALA IP entity

To learn more about how to use these features, you may be interested in the usage section of the docs.

1.2 TODO

- Host COALA IP JSON-LD definitions and set `<coalaip placeholder>` to the purl for the definitions.
- Support IPLD serialization

1.3 Packaging

Bumping versions:

```
$ bumpversion patch
```

Releasing to pypi:

```
$ make release
$ twine upload dist/*
```

1.4 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install pycoalaip, run this command in your terminal:

```
$ pip install coalaip
```

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

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

2.2 From sources

The sources for pycoalaip can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/bigchaindb/pycoalaip
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use pycoalaip in a project:

```
import coalaip
```

3.1 Quickstart

To get started with `coalaip`, you should first pick a persistence layer (and an accompanying plugin) to use. For a list of available persistence layer plugins, see *here*.

Once you've configured your chosen plugin, the main workflow to follow is:

1. Create an instance of `CoalaIp`; 1. Generate users for yourself and other parties; 1. Register a `Manifestation` entity (and its accompanying `Work` and `Copyright` entities) for your IP; 1. Derive a specific `Right` from your IP's `Copyright` (or another `Right` that pertains to your IP); and 1. If desired, transfer the specific `Right` to another party, to record a legal transaction relating to the `Right` (e.g. a transfer of ownership, a loan, etc).

Note: Each of `CoalaIp.register_manifestation()`, `CoalaIp.derive_right()`, and `CoalaIp.transfer_right()` have optional arguments to cover alternate use cases that are not explained here.

You may be interested in looking at the *library reference* for their complete documentation.

Warning: In the current implementation, operations that use the persistence layer are **NOT** ensured to succeed, and you may find that some operations need to be repeated.

A good example of this is if a storage requiring non-neglible consensus (e.g. BigchainDB) is used: the implementation assumes that everything has succeeded if it was able to write to the storage rather than confirming (later) that what it wrote was actually accepted.

3.1.1 Creating an instance of `CoalaIp`

Let's assume you have an instance of a persistence layer plugin ready.

```
from coalaip import CoalaIp

plugin = Plugin(...)
coalaip = CoalaIp(plugin)
```

3.1.2 Generating users

Representations of users are defined by the persistence layer plugin. You can generate a user compatible with your chosen persistence layer by:

```
# Note that the plugin may dictate that you need to provide extra arguments
# to this function
user = coalaip.generate_user()
```

3.1.3 Registering a `Manifestation`

Upon initial registration of a `Manifestation`, a `Work` (if not provided) and `Copyright` are automatically generated.

```
manifestation_data = {...}
registration_result = coalaip.register_manifestation(manifestation_data,
                                                     copyright_holder=user)
manifestation = registration_result['manifestation']
work = registration_result['work']
copyright = registration_result['copyright']
```

3.1.4 Deriving a specific `Right`

You can create more specific `Rights` from source `Rights` or `Copyrights` if you are the current holder of the source `Right`.

```
copyright = ...
right_data = {...}
right = coalaip.derive_right(right_data, current_holder=user,
                             source_right=copyright)
```

3.1.5 Transferring a `Right`

Transfers of a `Right` will change ownership of the entity from the current holder to a new holder. A `RightsAssignment` entity can also be encoded in a transfer, holding more specific information about the particular details related to the transaction, such as a agreed-upon contract between the two parties, the time of the transaction, and etc.

```
right = ...
current_holder = ... # user representation
new_holder = ... # user representation
```

(continues on next page)

(continued from previous page)

```
rights_assignment_data = {...}
rights_assignment = coalaip.transfer_right(right, rights_assignment_data,
                                           current_holder=current_holder,
                                           to=new_holder)
```

3.1.6 Querying for an `Entity`'s ownership history

Each entity returned has a `.history()` method and `.current_owner` property defined, in case you're interested in finding out the ownership history of the entity.

3.1.7 Obtaining an instance of an `Entity`

If you know you have COALA IP entities persisted, but don't have them in an `Entity` class (e.g. you saved the entities' IDs in a database, and now want to use them), you can load an instance of an `Entity` by using the static `.from_persist_id()` method of that entity type.

```
from coalaip.entities import Manifestation

manifestation_id = '...'
manifestation = Manifestation.from_persist_id(manifestation_id,
                                              plugin=plugin)
```

Doing so will generate a lazy-loaded entity for you to use. Accessing the entity's data for the first time will load the entity from the persistence layer (which may error); if you'd like to load it immediately, you can either call `.load()` or use the `force_load` flag in `.from_persist_id()`:

```
manifestation = Manifestation.from_persist_id(manifestation_id,
                                              plugin=plugin)
manifestation.load()

# Or
manifestation = Manifestation.from_persist_id(manifestation_id,
                                              force_load=True,
                                              plugin=plugin)
```

3.2 Reference

See the *library reference* for a complete reference of all available classes and functions.

CHAPTER 4

Plugins

`pycoalaip` requires a persistence layer plugin to be used in order to persist COALA IP entities to a distributed ledger, database, or file storage system.

4.1 Available Plugins

- BigchainDB

4.2 Writing a Plugin

Writing a plugin for `pycoalaip` is relatively simple. We use the *pycoalaip-{plugin_name}* naming scheme for plugin packages.

A plugin is expected to subclass from *`AbstractPlugin`* and implement all the abstract methods and properties, following the API laid out in the *`AbstractPlugin`*'s documentation.

To make your plugin discoverable by name to `pycoalaip`, you should also set an entry point in your `setup.py` for the `coalaip_plugin` namespace. Taking the BigchainDB plugin as an example, this may look something like:

```
setup(
    ...
    entry_points={
        'coalaip_plugin': 'bigchaindb = coalaip_bigchaindb.plugin:Plugin'
    },
    ...
)
```

CHAPTER 5

Library Reference

5.1 coalaip

High-level functions for interacting with COALA IP entities

class coalaip.coalaip.**CoalaIp**(*plugin*)

High-level, plugin-bound COALA IP functions.

Instantiated with an subclass implementing the ledger plugin interface (*AbstractPlugin*) that will automatically be bound to all top-level functions:

- *generate_user()*
- *register_manifestation()*
- *derive_right()*
- *transfer_right()*

plugin

Plugin – Bound persistence layer plugin.

__init__(*plugin*) → None

Initialize self. See help(type(self)) for accurate signature.

derive_right(*right_data*, *, *current_holder*, *source_right=None*, *right_entity_cls=<class 'coalaip.entities.Right'>*, ***kwargs*)

Derive a new Right from an existing `source_right` (a *Right* or subclass) for the `current_holder` of the `source_right`. The newly registered Right can then be transferred to other Parties.

Parameters

- **right_data** (*dict*) – Model data for the `right_entity_cls`. See the given `right_entity_cls` for requirements. If `source` is provided in the dict, the `source_right` parameter is ignored.
- **current_holder** (*any, keyword*) – The current holder of the `source_right`; must be specified in the format required by the persistence layer

- **source_right** (*Right*, keyword, optional) – An already persisted Right that the new Right is allowed by. Must be using the same plugin that *CoalaIp* was instantiated with. Ignored if `source` is provided in `right_data`.
- **right_entity_cls** (subclass of *Right*, keyword, optional) – The class that must be instantiated for the newly derived right. Defaults to *Right*.
- ****kwargs** – Keyword arguments passed through to the `right_entity_cls`'s `create` method (e.g. *create()*'s `data_format`)

Returns A registered `right_entity_cls` Right (by default a *Right*)

Raises

- *ModelDataError* – If the `right_data` contains invalid or is missing required properties.
- *EntityNotYetPersistedError* – If the `source_right` is not associated with an id on the persistence layer (*persist_id*) yet
- *EntityCreationError* – If the Right fails to be created on the persistence layer
- *PersistenceError* – If any other error occurred with the persistence layer

generate_user(**args*, ***kwargs*)

Generate a new user for the backing persistence layer.

Parameters

- ***args** – Argument list passed to the plugin's `generate_user()`
- ****kwargs** – Keyword arguments passed to the plugin's `generate_user()`

Returns A representation of a user, based on the persistence layer plugin

Raises *PersistenceError* – If a user couldn't be generated on the persistence layer

register_manifestation(*manifestation_data*, *, *copyright_holder*, *existing_work=None*, *work_data=None*, *create_work=True*, *create_copyright=True*, ***kwargs*)

Register a Manifestation and automatically assign its corresponding Copyright to the given `user`.

Unless specified (see `existing_work`), also registers a new Work for the Manifestation.

Parameters

- **manifestation_data** (*dict*) – Model data for the *Manifestation*. See *Manifestation* for requirements. If `manifestationOfWork` is provided in the dict, the `existing_work` and `work_data` parameters are ignored and no Work is registered.
- **copyright_holder** (*any, keyword*) – The user to hold the corresponding Copyright of the registered Manifestation; must be specified in the format required by the persistence layer
- **existing_work** (*Work*, keyword, optional) – An already persisted Work that the Manifestation is derived from. Must be using the same plugin that *CoalaIp* was instantiated with. If specified, the `work_data` parameter is ignored and no Work is registered.
- **work_data** (*dict, keyword, optional*) – Model data for the Work that will automatically generated for the Manifestation if no `existing_work` was specified. See *Work* for requirements. If not specified, the Work will be created using only the name of the Manifestation.

- **create_work** (*bool, keyword, optional*) – To allow for the creation of a Manifestation without attaching a Work. Default is True.
- **create_copyright** (*bool, keyword, optional*) – To allow for the creation of a Manifestation without attaching a Copyright. Default is True.
- ****kwargs** – Keyword arguments passed through to each model's *create()* (e.g. `data_format`).

Returns

A `namedtuple` containing the Coypright of the registered Manifestation, the registered Manifestation, and the Work as named fields:

```
(
    'copyright': (:class:`~.Copyright`),
    'manifestation': (:class:`~.Manifestation`),
    'work': (:class:`~.Work`),
)
```

If `manifestationOfWork` was provided in `manifestation_data`, None will be returned for the Work; otherwise, the given `existing_work` or automatically created Work will be returned.

Return type `RegistrationResult`

Raises

- *ModelDataError* – If the `manifestation_data` or `work_data` contain invalid or are missing required properties.
- *IncompatiblePluginError* – If the `existing_work` is not using a compatible plugin
- *EntityNotYetPersistedError* – If the `existing_work` is not associated with an id on the persistence layer (*persist_id*) yet
- *EntityCreationError* – If the manifestation, its copyright, or the automatically created work (if no existing work is given) fail to be created on the persistence layer
- *PersistenceError* – If any other error occurred with the persistence layer

register_work(*work_data*, *, *copyright_holder*, ***kwargs*)
Register a work

transfer_right(*right*, *rights_assignment_data=None*, *, *current_holder*, *to*, ***kwargs*)
Transfer a Right to another user.

Parameters

- **right** (*Right*) – An already persisted Right to transfer
- **rights_assignment_data** (*dict, optional*) – Model data for the generated *RightsAssignment* that will be associated with the transfer
- **current_holder** (*any, keyword*) – The current holder of the `right`; must be specified in the format required by the persistence layer
- **to** (*any, keyword*) – The new holder of the right; must be specified in the format required by the persistence layer. If the specified user format includes private information (e.g. a private key) but is not required by the persistence layer to identify a transfer recipient, then this information may be omitted in this argument.

- ****kwargs** – keyword arguments passed through to the `right`'s `transfer` method (e.g. *`transfer()`*'s `rights_assignment_format`)

Returns the RightsAssignment entity associated with this transfer

Return type *`RightsAssignment`*

Raises

- *`EntityNotYetPersistedError`* – If the `right` has not been persisted yet
- *`EntityNotFoundError`* – If the `right` was not found on the persistence layer
- *`EntityTransferError`* – If the `right` fails to be transferred on the persistence layer
- *`PersistenceError`* – If any other error occurred with the persistence layer

5.2 entities

Entities mirroring COALA IP's entity model.

Requires usage with a persistence layer plugin (see *`AbstractPlugin`*) for the creation and transfer of entities. JSON, JSON-LD, and IPLD data formats are supported.

Note: This module should not be used directly to generate entities, unless you are extending the built-ins for your own extensions. Instead, use the high-level functions (*`coalaip`*) that return instances of these entities.

Warning: The immutability guarantees given in this module are best-effort. There is no general way to achieve immutability in Python, but we try our hardest to make it so.

5.2.1 Core Entities

Note: Most of these core entity classes have their functionality implemented through *`Entity`*. See *`Entity`* for an overview of the base functionality of each of these core entities.

class coalaip.entities.**Work**(*model*, *plugin*)

COALA IP's Work entity.

A distinct, abstract Creation whose existence is revealed through one or more *`Manifestation`* entities.

`Work` entities are always of @type 'AbstractWork'.

classmethod generate_model(**args*, ***kwargs*)

Generate a Work model.

See *`generate_model()`* for more details.

Ignores the given `ld_type` as *`Work`* entities always have @type 'AbstractWork'.

class coalaip.entities.**Manifestation**(*model*, *plugin*)

COALA IP's Manifestation entity.

A perceivable manifestation of a *`Work`*.

`Manifestation` entities are by default of @type 'CreativeWork'.

classmethod generate_model(**args*, ***kwargs*)
Generate a Manifestation model.

See *generate_model()* for more details.

class coalaip.entities.**Right**(*model*, *plugin*)
COALA IP's Right entity. Transferrable.

A statement of entitlement (i.e. "right") to do something in relation to a *Work* or *Manifestation*.

More specific rights, such as `PlaybackRights`, `StreamRights`, etc should be implemented as subclasses of this class.

By default, `Rights` entities are of @type 'Right' and only include the COALA IP context, as Rights are not dependent on schema.org.

classmethod generate_model(**args*, ***kwargs*)
Generate a Work model.

See *generate_model()* for more details.

transfer(*rights_assignment_data=None*, ***, *from_user*, *to_user*, *rights_assignment_format='jsonld'*)
Transfer this Right to another owner on the backing persistence layer.

Parameters

- **rights_assignment_data** (*dict*) – Model data for the resulting *RightsAssignment*
- **from_user** (*any, keyword*) – A user based on the model specified by the persistence layer
- **to_user** (*any, keyword*) – A user based on the model specified by the persistence layer
- **rights_assignment_format** (*str, keyword, optional*) – Data format of the created entity; must be one of:
  - 'jsonld' (default)
  - 'json'
  - 'ipld'

Returns The RightsAssignment entity created from this transfer

Return type *RightsAssignment*

Raises See *transfer()*

class coalaip.entities.**Copyright**(*model*, *plugin*)
COALA IP's Copyright entity. Transferrable.

The full entitlement of Copyright to a *Work* or *Manifestation*.

Copyright entities are always of @type 'Copyright' and by default only include the COALA IP context, they are not dependent on schema.org.

classmethod generate_model(**args*, ***kwargs*)
Generate a Work model.

See *generate_model()* for more details.

Ignores the given `ld_type` as *Copyright* are always 'Copyright's.

class coalaip.entities.**RightsAssignment**(*model*, *plugin*)
COALA IP's RightsAssignment entity.

The assignment (e.g. transfer) of a *Right* to someone.

RightsAssignment entities may only be persisted in the underlying persistence layer through transfer operations, and hence cannot be created normally through *create()*.

RightsAssignment entities are always of @type 'RightsAssignment' and by default only include the COALA IP context, as Copyrights are not dependent on schema.org.

create(**args*, ***kwargs*)
Removes the ability to persist a *RightsAssignment* normally. Raises *PersistenceError* if called.

classmethod generate_model(**args*, ***kwargs*)
Generate a Work model.

See *generate_model()* for more details.

Ignores the given `ld_type` as *RightsAssignment* entities always have @type 'RightsTransferAction's.

5.2.2 Base Entities

Base functionality for the models above. These should never be instantiated; prefer one of the *Core Entities* instead.

class coalaip.entities.**Entity**(*model*, *plugin*)
Abstract base class of all COALA IP entity models.

Immutable (see :class:'~.PostInitImmutable').

Implements base functionality for all COALA IP entities, including entity creation (*create()*) and status queries (*status*) on the backing persistence layer provided by `plugin`.

Subclasses **must** implement their own *generate_model()*; *generate_model()* determines the semantics behind *model* (its creation and validation).

model
Model or *LazyLoadableModel* – Model of the entity. Holds the data and Linked Data (JSON-LD) specifics.

plugin
subclass of *AbstractPlugin* – Persistence layer plugin used by the Entity

persist_id
str – Id of this entity on the persistence layer, if saved to one. Initially `None`. Not initable. Note that this attribute is only immutable after it's been set once after initialization (e.g. after *create()*).

create(*user*, *data_format=<DataFormat.jsonld: 'jsonld'>*)
Create (i.e. persist) this entity to the backing persistence layer.

Parameters

- **user** (*any*) – A user based on the model specified by the persistence layer
- **data_format** (*DataFormat* or str) – Data format used in persisting the entity; must be a member of *DataFormat* or a string equivalent. Defaults to jsonld.

Returns Id of this entity on the persistence layer

Return type str

Raises

- *EntityCreationError* – If an error occurred during the creation of this entity that caused it to **NOT** be persisted. Contains the original error from the persistence layer, if available.
- *EntityPreviouslyCreatedError* – If the entity has already been persisted. Contains the existing id of the entity on the persistence layer.
- *PersistenceError* – If any other unhandled error in the plugin occurred

current_owner

any – A user based on the model specified by the persistence layer if a current owner exists, otherwise None. In the case where the user model contains secret information, the returned user may omit this information.

Raises

- *EntityNotFoundError* – If the entity is persisted, but could not be found on the persistence layer
- *PersistenceError* – If any other unhandled error in the plugin occurred

data

dict – A copy of the basic data held by this entity model. Does not include any JSON-LD or IPLD specific information.

If the entity was generated through *from_persist_id()*, the first access of this property may also load the entity's data from the persistence layer (see *load()* for potentially raised exceptions)

classmethod from_data(*data*, *, *data_format=<DataFormat.jsonld: 'jsonld'>*, *plugin*)

Generic factory for instantiating `cls` entities from their model data. Entities instantiated from this factory have yet to be created on the backing persistence layer; see *create()* on persisting an entity.

Based on the `data_format`, the following are considered special keys in *data* and will have different behaviour depending on the `data_type` requested in later methods (e.g. *create()*):

- **jsonld:**
  - '@type' denotes the Linked Data type of the entity
  - '@context' denotes the JSON-LD context of the entity
  - '@id' denotes the JSON-LD identity of the entity
- **Otherwise:**
  - 'type' denotes the Linked Data type of the entity

Parameters

- **data** (*dict*) – Model data for the entity
- **data_format** (*DataFormat* or str) – Data format of *data*; must be a member of *DataFormat* or a string equivalent. Defaults to jsonld.
- **plugin** (subclass of *AbstractPlugin*, keyword) – Persistence layer plugin used by generated `cls`

Returns A generated `cls` entity from *data*

Return type `cls`

Raises *ModelDataError* – if *data* fails model validation

classmethod from_persist_id(*persist_id*, *, *force_load=False*, *plugin*)
Generic factory for creating `cls` entity instances from their persisted ids.

Note: by default, instances generated from this factory lazily load their data upon first access (accessing *data()*), which may throw under various conditions. In general, most usages of `Entity` and its subclasses do not require access to their data (including internal methods), and thus the data does not usually need to be loaded unless you expect to explicitly use *data()* or one of the transformation methods, e.g. *to_json()*. If you know you will be using the data and want to avoid raising unexpected exceptions upon access, make sure to set `force_load` or use *load()* on the returned entity before accessing *data()*.

Parameters

- **persist_id** (*str*) – Id of the entity on the persistence layer (see *Entity.plugin*)
- **force_load** (*bool, keyword, optional*) – Whether to load the entity's data immediately from the persistence layer after instantiation. Defaults to false.
- **plugin** (subclass of *AbstractPlugin*, keyword) – Persistence layer plugin used by generated `cls`

Returns A generated entity based on *persist_id*

Return type `cls`

Raises

- If `force_load` is `True`, see *load()* for the
- list of possible exceptions.

classmethod generate_model(*, *data*, *ld_type*, *ld_context*, *model_cls*)
Generate a model instance for use with the current `cls`.

Must be implemented by subclasses of *Entity*.

Parameters

- **data** (*dict, keyword*) – Model data
- **ld_type** (*str, keyword*) – @type of the entity.
- **ld_context** (*str or dict or [str|dict], keyword*) – "@context" for the entity as either a string URL or array of string URLs or dictionaries. See the JSON-LD spec on contexts for more information.
- **model_cls** (*class, keyword*) – Model class to use the generated model. See *models*.

Returns A model instance

Raises *ModelDataError* – if *data* fails model validation

history
list of dict – A list containing the ownership history of this entity. Each item in the list is a dict containing a user based on the model specified by the persistence layer and a reference id for the event (e.g. transfer). The ownership events are sorted starting from the beginning of the entity's history (i.e. creation). In the case where the user model contains secret information, the returned user may omit this information.

Raises

- *EntityNotFoundError* – If the entity is persisted, but could not be found on the persistence layer
- *PersistenceError* – If any other unhandled error in the plugin occurred

load()
Load this entity from the backing persistence layer, if possible.

When used by itself, this method is most useful in ensuring that an entity generated from *from_persist_id()* is actually available on the persistence layer to avoid errors later.

Raises

- *EntityNotYetPersistedError* – If the entity is not associated with an id on the persistence layer (*persist_id*) yet
- *EntityNotFoundError* – If the entity has a *persist_id* but could not be found on the persistence layer
- *PersistenceError* – If any other unhandled error in the plugin occurred
- *ModelDataError* – If the loaded entity's data fails validation or its type or context differs from their expected values

status
The current status of this entity in the backing persistence layer, as defined by *Entity.plugin*. Initially None.

Raises

- *EntityNotFoundError* – If the entity is persisted, but could not be found on the persistence layer
- *PersistenceError* – If any other unhandled error in the plugin occurred

to_ipld()
Output this entity's data as an IPLD-serializable dict.

The entity's @type is represented as 'type' and the @context is ignored.

to_json()
Output this entity as a JSON-serializable dict.

The entity's @type is represented as 'type' and the @context is ignored.

to_jsonld()
Output this entity as a JSON-LD-serializable dict.

Adds the @type, @context, and @id as-is. If no @id was given, an empty @id is used by default to refer to the current *persist_id* document.

class coalaip.entities.**TransferrableEntity**(*model*, *plugin*)
Base class for transferable COALA IP entity models.

Provides functionality for transferrable entities through *transfer()*

transfer(*transfer_payload=None*, ***, *from_user*, *to_user*)
Transfer this entity to another owner on the backing persistence layer

Parameters

- **transfer_payload** (*dict*) – Payload for the transfer
- **from_user** (*any*) – A user based on the model specified by the persistence layer
- **to_user** (*any*) – A user based on the model specified by the persistence layer

Returns Id of the resulting transfer action on the persistence layer

Return type str

Raises

- *EntityNotYetPersistedError* – If the entity being transferred is not associated with an id on the persistence layer (*persist_id*) yet
- *EntityNotFoundError* – If the entity could not be found on the persistence layer
- *EntityTransferError* – If the entity fails to be transferred on the persistence layer
- *PersistenceError* – If any other unhandled error in the plugin occurred

5.3 models

Low level data models for COALA IP entities.

Encapsulates the data modelling of COALA IP entities. Supports model validation and the loading of data from a backing persistence layer.

Note: This module should not be used directly to generate models, unless you are extending the built-ins for your own extensions. Instead, use the models that are contained in the entities (*entities*) returned from the high-level functions (*coalaip*).

Warning: The immutability guarantees given in this module are best-effort. There is no general way to achieve immutability in Python, but we try our hardest to make it so.

class coalaip.models.**Model**(*data*, *ld_type*, *ld_id=''*, *ld_context=NOTHING*, *validator=<instance_of validator for type <class 'mappingproxy'>>*)

Basic data model class for COALA IP entities. Includes Linked Data (JSON-LD) specifics.

Immutable (see :class:'~.PostInitImmutable' and attributes).

Initialization may throw if attribute validation fails.

data
dict – Model data. Uses *validator* for validation.

ld_type
str – @type of the entity

ld_id
str – @id of the entity

ld_context
str or dict or [str|dict], keyword – "@context" for the entity as either a string URL or array of string URLs or dictionaries. See the JSON-LD spec on contexts for more information.

validator
callable – A validator complying to `attr`'s validator API that will validate *data*

__init__(*data*, *ld_type*, *ld_id=''*, *ld_context=NOTHING*, *validator=<instance_of validator for type <class 'mappingproxy'>>*) → None
Initialize self. See help(type(self)) for accurate signature.

class coalaip.models.**LazyLoadableModel**(*ld_type*, *ld_id=None*, *ld_context=None*, *validator=<instance_of validator for type <class 'mappingproxy'>>*, *data=None*)

Lazy loadable data model class for COALA IP entities.

Immutable (see :class:'.PostInitImmutable' and attributes).

Similar to *Model*, except it allows the model data to be lazily loaded afterwards from a backing persistence layer through a plugin.

loaded_model
Model – Loaded model from a backing persistence layer. Initially `None`. Not initable. Note that this attribute is only immutable after it's been set once after initialization (e.g. after *load()*).

ld_type
See *ld_type*

ld_context
See *ld_context*

validator
See *validator*

__init__(*ld_type*, *ld_id=None*, *ld_context=None*, *validator=<instance_of validator for type <class 'mappingproxy'>>*, *data=None*)
Initialize a *LazyLoadableModel* instance.

If a *data* is provided, a *Model* is generated as the instance's *loaded_model* using the given arguments.

Ignores *ld_id*, see the *ld_id()* property instead.

data
dict – Model data.

Raises *ModelNotYetLoadedError* if the data has not been loaded yet.

ld_id
str – @id of the entity.

Raises *ModelNotYetLoadedError* if the data has not been loaded yet.

load(*persist_id*, ***, *plugin*)
Load the *loaded_model* of this instance. Noop if model was already loaded.

Parameters

- **persist_id** (*str*) – Id of this model on the persistence layer
- **plugin** (subclass of *AbstractPlugin*) – Persistence layer plugin to load from

Raises

- *ModelDataError* – If the loaded entity's data fails validation from `validator` or its type or context differs from their expected values
- *EntityNotFoundError* – If the entity could not be found on the persistence layer
- *PersistenceError* – If any other unhandled error in the plugin occurred

5.4 data formats

Utilities for data formats supported by pycoalaip.

class coalaip.data_formats.**DataFormat**
Enum of supported data formats.

5.5 exceptions

Custom exceptions for COALA IP

class coalaip.exceptions.**CoalaIpError**
Base class for all Coala IP errors.

class coalaip.exceptions.**IncompatiblePluginError**
Raised when entities with incompatible plugins are used together. Should contain a list of the incompatible plugins as the first argument.

class coalaip.exceptions.**ModelError**
Base class for all model errors.

class coalaip.exceptions.**ModelDataError**
Raised if there is an error with the model's data.

class coalaip.exceptions.**ModelNotYetLoadedError**
Raised if the lazily loaded model has not been loaded from the backing persistence layer yet.

class coalaip.exceptions.**PersistenceError**(*message=''*, *error=None*)
Base class for all persistence-related errors.

message
str – Message of the error

error
`Exception` – Original exception, if available

class coalaip.exceptions.**EntityCreationError**(*message=''*, *error=None*)
Raised if an error occured during the creation of an entity on the backing persistence layer. Should contain the original error that caused the failure, if available.

class coalaip.exceptions.**EntityNotFoundError**(*message=''*, *error=None*)
Raised if the entity could not be found on the backing persistence layer

class coalaip.exceptions.**EntityNotYetPersistedError**(*message=''*, *error=None*)
Raised when an action requiring an entity to be available on the persistence layer is attempted on an entity that has not been persisted yet.

class coalaip.exceptions.**EntityPreviouslyCreatedError**(*existing_id*, **args*, ***kwargs*)
Raised when attempting to persist an already persisted entity. Should contain the existing id of the entity.

existing_id
str – Currently existing id of the entity on the persistence layer

See :exc:`.PersistenceError` for other attributes.

class coalaip.exceptions.**EntityTransferError**(*message=''*, *error=None*)
Raised if an error occured during the transfer of an entity on the backing persistence layer. Should contain the original error that caused the failure, if available.

5.6 plugin

class coalaip.plugin.**AbstractPlugin**
Abstract interface for all persistence layer plugins.

Expects the following to be defined by the subclass:

- *type* (as a read-only property)

- *generate_user()*
- *get_status()*
- *save()*
- *transfer()*

generate_user(**args*, ***kwargs*)
Generate a new user on the persistence layer.

Parameters

- ***args** – argument list, as necessary
- ****kwargs** – keyword arguments, as necessary

Returns A representation of a user (e.g. a tuple with the user's public and private keypair) on the persistence layer

Raises *PersistenceError* – If any other unhandled error in the plugin occurred

get_history(*persist_id*)
Get the ownership history of an entity on the persistence layer.

Parameters **persist_id** (*str*) – Id of the entity on the persistence layer

Returns

The ownership history of the entity, sorted starting from the beginning of the entity's history (i.e. creation). Each dict is of the form:

```
{
    'user': A representation of a user as specified by the
            persistence layer (may omit secret details, e.g.␣
↪private keys),
    'event_id': A reference id for the ownership event (e.g.␣
↪transfer id)
}
```

Return type list of dict

Raises

- *EntityNotFoundError* – If the entity could not be found on the persistence layer
- *PersistenceError* – If any other unhandled error in the plugin occurred

get_status(*persist_id*)
Get the status of an entity on the persistence layer.

Parameters **persist_id** (*str*) – Id of the entity on the persistence layer

Returns Status of the entity, in any format.

Raises

- *EntityNotFoundError* – If the entity could not be found on the persistence layer
- *PersistenceError* – If any other unhandled error in the plugin occurred

is_same_user(*user_a*, *user_b*)
Compare the given user representations to see if they mean the same user on the persistence layer.

Parameters

- **user_a** (*any*) – User representation

- **user_b** (*any*) – User representation

Returns Whether the given user representations are the same user.

Return type bool

load(*persist_id*)
Load the entity from the persistence layer.

Parameters **persist_id** (*str*) – Id of the entity on the persistence layer

Returns The persisted data of the entity

Return type dict

Raises

- *EntityNotFoundError* – If the entity could not be found on the persistence layer
- *PersistenceError* – If any other unhandled error in the plugin occurred

save(*entity_data*, ***, *user*)
Create the entity on the persistence layer.

Parameters

- **entity_data** (*dict*) – The entity's data
- **user** (*any, keyword*) – The user the entity should be assigned to after creation. The user must be represented in the same format as *generate_user()*'s output.

Returns Id of the created entity on the persistence layer

Return type str

Raises

- EntityCreationError – If the entity failed to be created
- *PersistenceError* – If any other unhandled error in the plugin occurred

transfer(*persist_id*, *transfer_payload*, ***, *from_user*, *to_user*)
Transfer the entity whose id matches persist_id on the persistence layer from the current user to a new owner.

Parameters

- **persist_id** (*str*) – Id of the entity on the persistence layer
- **transfer_payload** (*dict*) – The transfer's payload
- **from_user** (*any, keyword*) – The current owner, represented in the same format as *generate_user()*'s output
- **to_user** (*any, keyword*) – The new owner, represented in the same format as *generate_user()*'s output. If the specified user format includes private information (e.g. a private key) but is not required by the persistence layer to identify a transfer recipient, then this information may be omitted in this argument.

Returns Id of the transfer action on the persistence layer

Return type str

Raises

- *EntityNotFoundError* – If the entity could not be found on the persistence layer
- EntityTransferError – If the entity failed to be transferred

- *PersistenceError* – If any other unhandled error in the plugin occurred

type
A string denoting the type of plugin (e.g. BigchainDB).

CHAPTER 6

About this Documentation

This section contains instructions to build and view the documentation locally.

If you do not have a clone of the repo, you need to get one.

6.1 Building the documentation

To build the docs, simply run

```
$ make docs
```

6.2 Viewing the documentation

You can either start a little web server locally, or open the HTML files with your browser.

To start a web server at http://localhost:5555/

```
# In project root, after making the docs
$ cd docs/_build/html/ && python -m SimpleHTTPServer 5555
```

Alternatively, open the *docs/_build/html/index.html* file in your web browser.

6.3 Making changes

Rebuild the docs and refresh the page on your web browser.

CHAPTER 7

Contributing

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

You can contribute in many ways:

7.1 Types of Contributions

7.1.1 Report Bugs

Report bugs at https://github.com/bigchaindb/pycoalaip/issues.

If you are reporting a bug, please include:

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

7.1.2 Fix Bugs

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

7.1.3 Implement Features

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

7.1.4 Write Documentation

pycoalaip could always use more documentation, whether as part of the official pycoalaip docs, in docstrings, or even on the web in blog posts, articles, and such.

7.1.5 Submit Feedback

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

If you are proposing a feature:

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

7.2 Get Started!

Ready to contribute? Here's how to set up *coalaip* for local development.

1. Fork the *coalaip* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/coalaip.git
```

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

```
$ mkvirtualenv coalaip
$ cd coalaip/
$ pip install -r requirements_dev.txt
```

4. Create a branch for local development:

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

   Now you can make your changes locally.

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

```
$ flake8 coalaip tests
$ pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

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

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

7. Submit a pull request through the GitHub website.

7.3 Pull Request Guidelines

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

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.4 and 3.5. Check https://travis-ci.org/bigchaindb/pycoalaip/pull_requests and make sure that the tests pass for all supported Python versions.

7.4 Tips

To run a subset of tests:

```
$ pytest tests.test_coalaip
```

To run tests with debugging:

```
$ pytest -s
```

To run tests and break on errors:

```
$ pytest --pdb
```

CHAPTER 8

Credits

8.1 Development Lead

- BigchainDB <dev@bigchaindb.com>

8.2 Contributors

None yet. Why not be the first?

CHAPTER 9

History

9.1 0.0.3 (2017-05-06)

Some changes during the OMI hackfest!

- Make creation of Work and Copyright optional when registering a Manifestation.

9.2 0.0.2 (2017-05-05)

Some changes during the OMI hackfest!

Some highlights:

- Add *register_work* method to enable registering a work without necessarily registering a manifestation.

9.3 0.0.1 (2017-02-17)

First alpha release on PyPI.

Additional features added with no backwards-incompatible interface changes. COALA IP models are backwards-incompatible to previous versions due to upgrades related to spec changes.

Some highlights:

- Queryability of an Entity's ownership history and current owner
- Entities can be given a custom `@id`
- Additional sanity checks employed when deriving Rights, to ensure that a correct source Right and current holder are given
- Update COALA IP models to latest spec
- Added usage documentation

9.4 0.0.1.dev3 (2016-12-06)

Lots of changes and revisions from 0.0.1.dev2. Totally incompatible from before.

Some highlights:

- Implemented Rights derivation (from existing Rights and Copyrights)
- Implemented Rights transfers
- Entities are now best-effort immutable
- Support for loading Entities from a connected persistence layer

9.5 0.0.1.dev2 (2016-08-31)

- Fix packaging on PyPI

9.6 0.0.1.dev1 (2016-08-31)

- Development (pre-alpha) release on PyPI.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

Symbols

A

C

D

E

F

G

H

I

L

M

P

R

S

T

V

W