# PROV Database Connector Documentation

***Release 0.3***

**German Aerospace Center (DLR)**

**Dec 06, 2020**

# Contents

CHAPTER 1

# PROV Database Connector

## 1.1 Introduction

This python module provides a general interface to save W3C-PROV documents into databases. Currently we support the Neo4j graph database.

We transform a PROV document into a graph structure and the result looks like this:



Fig. 1: Complex example in Neo4j

See full documentation at: prov-db-connector.readthedocs.io

## 1.2 Installation

### 1.2.1 PyPi

Install it by running:

```
pip install prov-db-connector
```

You can view prov-db-connector on PyPi's package index

### 1.2.2 Source

```
# Clone project
git clone git@github.com:DLR-SC/prov-db-connector.git
cd prov-db-connector

# Setup virtual environment
virtualenv -p /usr/bin/python3.4 env
source env/bin/activate

# Install dependencies and package into virtual enviroment
make setup
```

## 1.3 Usage

### 1.3.1 Save and get prov document example

```
from prov.model import ProvDocument
from provdbconnector import ProvDb
from provdbconnector.db_adapters.in_memory import SimpleInMemoryAdapter

prov_api = ProvDb(adapter=SimpleInMemoryAdapter, auth_info=None)

# create the prov document
prov_document = ProvDocument()
prov_document.add_namespace("ex", "http://example.com")

prov_document.agent("ex:Bob")
prov_document.activity("ex:Alice")

prov_document.association("ex:Alice", "ex:Bob")

document_id = prov_api.save_document(prov_document)

print(prov_api.get_document_as_provn(document_id))

# Output:
#
# document
# prefix
# ex < http: // example.com >
#
# agent(ex:Bob)
# activity(ex:Alice, -, -)
# wasAssociatedWith(ex:Alice, ex:Bob, -)
# endDocument
```

### 1.3.2 File Buffer example

```
from provdbconnector import ProvDb
from provdbconnector.db_adapters.in_memory import SimpleInMemoryAdapter
import pkg_resources
```

(continues on next page)

(continued from previous page)

```
# create the api
prov_api = ProvDb(adapter=SimpleInMemoryAdapter, auth_info=None)

# create the prov document from examples
prov_document_buffer = pkg_resources.resource_stream("examples", "file_buffer_example_
↪primer.json")

# Save document
document_id = prov_api.save_document(prov_document_buffer)
# This is similar to:
# prov_api.create_document_from_json(prov_document_buffer)

# get document
print(prov_api.get_document_as_provn(document_id))

# Output:

# document
# prefix
# foaf < http: // xmlns.com / foaf / 0.1 / >
# prefix
# dcterms < http: // purl.org / dc / terms / >
# prefix
# ex < http: // example / >
#
# specializationOf(ex:articleV2, ex:article)
# specializationOf(ex:articleV1, ex:article)
# wasDerivedFrom(ex:blogEntry, ex:article, -, -, -, [prov:type = 'prov:Quotation'])
# alternateOf(ex:articleV2, ex:articleV1)
# wasDerivedFrom(ex:articleV1, ex:dataSet1, -, -, -)
# wasDerivedFrom(ex:articleV2, ex:dataSet2, -, -, -)
# wasDerivedFrom(ex:dataSet2, ex:dataSet1, -, -, -, [prov:type = 'prov:Revision'])
# used(ex:correct, ex:dataSet1, -)
# used(ex:compose, ex:dataSet1, -, [prov:role = "ex:dataToCompose"])
# wasDerivedFrom(ex:chart2, ex:dataSet2, -, -, -)
# wasGeneratedBy(ex:dataSet2, ex:correct, -)
# used(ex:compose, ex:regionList, -, [prov:role = "ex:regionsToAggregateBy"])
# used(ex:illustrate, ex:composition, -)
# wasGeneratedBy(ex:composition, ex:compose, -)
# wasAttributedTo(ex:chart1, ex:derek)
# wasGeneratedBy(ex:chart1, ex:compile, 2012 - 03 - 02
# T10:30:00)
# wasGeneratedBy(ex:chart1, ex:illustrate, -)
# wasAssociatedWith(ex:compose, ex:derek, -)
# wasAssociatedWith(ex:illustrate, ex:derek, -)
# actedOnBehalfOf(ex:derek, ex:chartgen, ex:compose)
# entity(ex:article, [dcterms:title = "Crime rises in cities"])
# entity(ex:articleV1)
# entity(ex:articleV2)
# entity(ex:dataSet1)
# entity(ex:dataSet2)
# entity(ex:regionList)
# entity(ex:composition)
# entity(ex:chart1)
# entity(ex:chart2)
# entity(ex:blogEntry)
```

(continues on next page)

(continued from previous page)

```
# activity(ex:compile, -, -)
# activity(ex:compile2, -, -)
# activity(ex:compose, -, -)
# activity(ex:correct, 2012 - 03 - 31
# T09:21:00, 2012 - 04 - 01
# T15:21:00)
# activity(ex:illustrate, -, -)
# agent(ex:derek, [foaf:mbox = "<mailto:derek@example.org>", foaf:givenName = "Derek",
↪ prov:type = 'prov:Person'])
# agent(ex:chartgen, [foaf:name = "Chart Generators Inc", prov:type =
↪'prov:Organization'])
# endDocument
```

You find all examples in the examples folder

## 1.4 Release

Create a new release on github, please use the semver standard for the version number

## 1.5 License

See LICENSE file

CHAPTER 2

# Development

## 2.1 Contribute

Please, fork the code on Github and develop your feature in a new branch split from the develop branch. Commit your code to the main project by sending a pull request onto the develop branch

- Issue Tracker: https://github.com/DLR-SC/prov-db-connector/issues
- Source Code: https://github.com/DLR-SC/prov-db-connector

## 2.2 Setup

```
# Clone project
git clone git@github.com:DLR-SC/prov-db-connector.git
cd prov-db-connector

# Setup virtual environment
virtualenv -p /usr/bin/python3.4 env
source env/bin/activate

# Install dependencies
make dev-setup
```

## 2.3 Execute tests

```
make test
```

## 2.4 Coverage report

```
make coverage
```

## 2.5 Compile documentation

```
make docs
```

## 2.6 Create new database adapters

The database adapters are the binding class to the actual database. If you are consider to build your own adapter please keep in mind:

- All adapters **must** enhance the `Baseadapter` class.
- You **must** implement all specified functions in BaseAdapter
- You **should** test it via the *`AdapterTestTemplate`* class template.
- You **should** test it also via the `ProvDbTestTemplate` class template.

### 2.6.1 1. - Create your database adapter

First you must create a class that extend from `Baseadapter` and implement all functions.

```
import logging
from uuid import uuid4

from prov.constants import PROV_ASSOCIATION, PROV_TYPE, PROV_MENTION
from provdbconnector.db_adapters.baseadapter import BaseAdapter, DbRecord, DbRelation, METADATA_KEY_IDENTIFIER, \
    METADATA_KEY_PROV_TYPE
from provdbconnector.exceptions.database import InvalidOptionsException,NotFoundException
from provdbconnector.utils.serializer import encode_dict_values_to_primitive, split_into_formal_and_other_attributes, \
    merge_record

log = logging.getLogger(__name__)


class SimpleInMemoryAdapter(BaseAdapter):
    """
    The simple in memory adapter is a reference implementation for a database adapter to save prov information
    into a graph database


    For exmaple to use the simple db_adapter use the following script

    .. literalinclude:: ../examples/simple_example.py
```

(continues on next page)

(continued from previous page)

```
            :linenos:
            :language: python

    """
    all_nodes = dict()  # separate dict for records only (to get them by id)
    """
    Contains all nodes
    """
```

### 2.6.2 2. - Create test suites

To test your adapter you should create two test suits:

- *SimpleInMemoryAdapterTest* : Unit test for the low level functions in your adapter.
- For further introduction on testing your database adapter have a look at the *Testing Howto*.
- *SimpleInMemoryAdapterProvDbTests* : Integration test for the adapter with the api.

See this example tests for the *SimpleInMemoryAdapter*

```
from provdbconnector.exceptions.database import InvalidOptionsException
from provdbconnector.db_adapters.in_memory import SimpleInMemoryAdapter
from provdbconnector.prov_db import ProvDb
from provdbconnector.tests import AdapterTestTemplate
from provdbconnector.tests import ProvDbTestTemplate


class SimpleInMemoryAdapterTest(AdapterTestTemplate):
    """
    This class implements the AdapterTestTemplate and only override some functions.

    """
    def setUp(self):
        """
        Connect to your database

        """
        self.instance = SimpleInMemoryAdapter()
        self.instance.connect(None)

    def test_connect_invalid_options(self):
        """
        Test your connect function with invalid data

        """
        auth_info = {"invalid": "Invalid"}
        with self.assertRaises(InvalidOptionsException):
            self.instance.connect(auth_info)

    def clear_database(self):
        """
        Clear the database

        """
        self.instance.all_nodes = dict()
```

(continues on next page)

(continued from previous page)

```
        self.instance.all_relations= dict()

    def tearDown(self):
        """
        Delete your instance

        """
        del self.instance


class SimpleInMemoryAdapterProvDbTests(ProvDbTestTemplate):
    """
    This is the high level test for the SimpleInMemoryAdapter

    """
    def setUp(self):
        """
        Setup a ProvDb instance
        """
        self.provapi = ProvDb(api_id=1, adapter=SimpleInMemoryAdapter, auth_info=None)

    def clear_database(self):
        """
        Clear function get called before each test starts

        """
        self.provapi._adapter.all_nodes = dict()
        self.provapi._adapter.all_relations = dict()

    def tearDown(self):
        """
        Delete prov api instance
        """
        del self.provapi
```

### 2.6.3 3. - Implement your adapter logic

The last step is to create your logic inside the *SimpleInMemoryAdapter* for example the save_record and get_record functions:

Now you are ready to implement all other functions.

**Note:** If you don't know where should you start Start with the first test and try to implement functions successively according to the tests and look into the documentation of the *AdapterTestTemplate*

```
    """
    Contains all relation according to the following structure
    `(start_identifier, (end_identifier,attributes, metadata))``
    """

    def __init__(self, *args):
        """
        Init the adapter without any params
```

(continues on next page)

(continued from previous page)

```
        :param args:
        """
        super(SimpleInMemoryAdapter, self).__init__()
        pass

    def connect(self, authentication_info):
        """
        This function setups your database connection (auth / service discover)

        :param authentication_info: The info to connect to the db
        :type authentication_info: dict or None
        :return: The result of the connection attempt
        :rtype: Bool
        """

        if authentication_info is not None:
            raise InvalidOptionsException()
        :type to_node: prov.model.Identifier
        :param attributes: The actual provenance data
        :type attributes: dict
        :param metadata: Some metadata that are not PROV-O related
        :type metadata: dict
        :return: The id of the relation
        :rtype: str
        """

        # save all relation information and return the relation id as string

```

CHAPTER 3

# Changelog

## 3.1 Version 0.3.x

- Upgraded prov to 1.5.3 .. #73: https://github.com/DLR-SC/prov-db-connector/pull/73
- Upgraded neo4j-driver to 1.7.0 .. #70: https://github.com/DLR-SC/prov-db-connector/pull/70

## 3.2 Version 0.3.1

- Upgraded neo4j-driver to 1.6.2 .. #67: https://github.com/DLR-SC/prov-db-connector/pull/67
- Enhanced error handling neo4j-adapater
- Automatic pipi release on git tag

## 3.3 Version 0.3

- **Changed ‘‘provdb.create_*‘‘ to ‘‘provdb.save_*‘‘** because we can’t guarantee that the db-adapter actual create a new node, document, relation. Maybe the adapter merges your properties into existing data, behavior is still the same.
- **Renamed files provDb.py into prov_db.py**
- Enhanced the `prov:Mention` support. If you create a bundle link (`prov:Mention`) the destination bundle entity will be automatically created. For example: ‘‘‘python

from prov.tests.examples import bundles2

doc = bundles2() bundle = list(doc.get_records()).pop() #I know, the get_record function return a set, so it can happen that you get the wrong bundle here (alice:bundle5 is correct) prov_api.save_bundle(bundle) ‘‘‘

- Add ability to save relations between elements that doesn’t exist. For example, on a empty database:

```
doc = ProvDocument()
relation = doc.wasGeneratedBy("ex:Entity", "ex:Activity")

#Works now fine. The ex:entity and ex:Activity elements will be created automatically
provapi.save_relation(relation)
```

- Removed node type "Unknown" for relations with unknown nodes. (The prov-db-adapter now detects which type the relation implicitly mean.

```
doc = ProvDocument()
relation = doc.wasGeneratedBy("ex:Entity", -)

# Creates a Activity with a random identifier as destions for the relation
provapi.save_relation(relation)
```

- Introduced new methods

**prov_db.save_relation(prov_relation)**

```
doc = ProvDocument()

activity    = doc.activity("ex:yourActivity")
entity      = doc.entity("ex:yourEntity")
wasGeneratedBy = entity.wasGeneratedBy("ex:yourAgent")

# Save the elements
rel_id = prov_db.save_relation(wasGeneratedBy)
```

**prov_db.save_element(prov_element, [bundle_id])**

```
doc = ProvDocument()

agent       = doc.agent("ex:yourAgent")
activity    = doc.activity("ex:yourActivity")
entity      = doc.entity("ex:yourEntity")

# Save the elements
agent_id = prov_db.save_element(agent)
activity_id = prov_db.save_element(activity)
entity_id = prov_db.save_element(entity)
```

**prov_db.get_element(identifier)**

```
doc = ProvDocument()

identifier = QualifiedName(doc, "ex:yourAgent")

prov_element = prov_db.get_element(identifier)
```

**prov_db.save_record(prov_record, [bundle_id])**

```
doc = ProvDocument()

agent       = doc.agent("ex:Alice")
ass_rel     = doc.association("ex:Alice", "ex:Bob")

# Save the elements
```

(continues on next page)

(continued from previous page)

```
agent_id = prov_db.save_record(agent)
relation_id = prov_db.save_record(ass_rel)
```

**prov_api.save_bundle(prov_bundle)**

```
doc = ProvDocument()

bundle = doc.bundle("ex:bundle1")
# Save the bundle
prov_db.save_bundle(bundle)
```

**prov_db.get_elements([ProvCLS])**

```
from prov.model import ProvEntity, ProvAgent, ProvActivity

document_with_all_entities = prov_db.get_elements(ProvEntity)
document_with_all_agents = prov_db.get_elements(ProvAgent)
document_with_all_activities = prov_db.get_elements(ProvActivity)

print(document_with_all_entities)
print(document_with_all_agents)
print(document_with_all_activities)
```

**prov_db.get_bundle(identifier)**

```
doc = ProvDocument()
bundle_name = doc.valid_qualified_name("ex:YourBundleName")
# get the bundle
prov_bundle = prov_db.get_bundle(bundle_name)
doc.add_bundle(prov_bundle)
```

# CHAPTER 4

# Testing Howto

To run the test local follow the next steps

## 4.1 1. Setup your env

```
# Clone project
git clone git@github.com:DLR-SC/prov-db-connector.git
cd prov-db-connector

# Setup virtual environment
virtualenv -p /usr/bin/python3.4 env
source env/bin/activate

# Install dependencies
make dev-setup
```

## 4.2 2. Start your neo4j setup

The tests require a running neo4j 3.0+ instance The simples way do start neo4j ist to use the docker image provided by neo4j

```
docker run \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    neo4j:3.0
```

Then open a browser *http://localhost:7474* and set the password to **neo4jneo4j** Alternative you can set the env. variables:

- NEO4J_USERNAME: Default: neo4j
- NEO4J_PASSWORD: Default: neo4jneo4j
- NEO4J_HOST: Default: localhost
- NEO4J_BOLT_PORT: Default: 7687
- NEO4J_HTTP_PORT: Default: 7474

Alternative use docker-compose

```
docker-compose up
```

## 4.3 3. Run your tests

```
# Change env
source env/bin/activate
#Start tests
make test
```

**Note:** If some tests fail because of certificate issues, delete or rename the known_hosts file in ~/.neo4j.

CHAPTER 5

# provdbconnector package

## 5.1 Subpackages

### 5.1.1 provdbconnector.db_adapters package

#### Subpackages

#### provdbconnector.db_adapters.in_memory package

#### Submodules

#### provdbconnector.db_adapters.in_memory.simple_in_memory module

**class** provdbconnector.db_adapters.in_memory.simple_in_memory.**SimpleInMemoryAdapter**(**args*)

Bases: *provdbconnector.db_adapters.baseadapter.BaseAdapter*

The simple in memory adapter is a reference implementation for a database adapter to save prov information into a graph database

For exmaple to use the simple db_adapter use the following script

```
from prov.model import ProvDocument
from provdbconnector import ProvDb
from provdbconnector.db_adapters.in_memory import SimpleInMemoryAdapter

prov_api = ProvDb(adapter=SimpleInMemoryAdapter, auth_info=None)

# create the prov document
prov_document = ProvDocument()
prov_document.add_namespace("ex", "http://example.com")

prov_document.agent("ex:Bob")
```

(continues on next page)

(continued from previous page)

```
prov_document.activity("ex:Alice")

prov_document.association("ex:Alice", "ex:Bob")

document_id = prov_api.save_document(prov_document)

print(prov_api.get_document_as_provn(document_id))

# Output:
#
# document
# prefix
# ex < http: // example.com >
#
# agent(ex:Bob)
# activity(ex:Alice, -, -)
# wasAssociatedWith(ex:Alice, ex:Bob, -)
# endDocument
```

**all_nodes = {}**
Contains all nodes

**all_relations = {}**
Contains all relation according to the following structure *(start_identifier, (end_identifier,attributes, metadata))‘*

**connect** (*authentication_info*)
This function setups your database connection (auth / service discover)

**Parameters** **authentication_info** (*dict or None*) – The info to connect to the db

**Returns** The result of the connection attempt

**Return type** Bool

**save_element** (*attributes*, *metadata*)
Store a single node in the database and if necessary and possible merge the node

**Parameters**

- **attributes** (*dict*) – The actual provenance data
- **metadata** (*dict*) – Some metadata that are not PROV-O related

**Returns** id of the record

**Return type** str

**save_relation** (*from_node*, *to_node*, *attributes*, *metadata*)
Store a relation between 2 nodes in the database. Merge also the relation if necessary and possible

**Parameters**

- **from_node** (*prov.model.Identifier*) – The identifier for the start node
- **to_node** (*prov.model.Identifier*) – The identifier for the end node
- **attributes** (*dict*) – The actual provenance data
- **metadata** (*dict*) – Some metadata that are not PROV-O related

**Returns** The id of the relation

**Return type** str

**get_record**(*record_id*)
Get a ProvDocument from the database based on the document id

**Parameters** **record_id** (*str*) – The id of the node

**Returns** A named tuple with (attributes, metadata)

**Return type** *DbRecord*

**get_relation**(*relation_id*)
Return the relation behind the relation_id

**Parameters** **relation_id** (*str*) – The id of the relation

**Returns** The namedtuple with (attributes, metadata)

**Return type** *DbRelation*

**get_records_by_filter**(*attributes_dict=None*, *metadata_dict=None*)
Filter all nodes based on the provided attributes and metadata dict The filter is currently defined as follows:

- The filter is only applied to the start node
- All connections from the start node are also included in the result set

**Parameters**

- **attributes_dict** (*dict*) – A filter dict with a conjunction of all values in the attributes_dict and metadata_dict
- **metadata_dict** (*dict*) – A filter for the metadata with a conjunction of all values (also in the attributes_dict )

**Returns** The list of matching relations and nodes

**Return type** List(*DbRecord* or Dbrelation)

**get_records_tail**(*attributes_dict=None*, *metadata_dict=None*, *depth=None*)
Return the provenance based on a filter combination. The filter dicts are only relevant for the start nodes. They describe the params to get the start nodes (for example a filter for a specific identifier ) and from there we want all connected nodes

**Parameters**

- **attributes_dict** (*dict*) – A filter dict with a conjunction of all values in the attributes_dict and metadata_dict
- **metadata_dict** (*dict*) – A filter for the metadata with a conjunction of all values (also in the attributes_dict )
- **depth** (*int*) – The level of detail, default to infinite

**Returns** A list of DbRelations and DbRecords

**Return type** list(*DbRelation* or *DbRecord*)

**get_bundle_records**(*bundle_identifier*)
Get the records for a specific bundle identifier

This include all nodes that have a relation of the prov:type = prov:bundleAssociation and also all relation where the start and end node are in the bundle. Also you should add the prov mentionOf relation where the start node is in the bundle. See https://www.w3.org/TR/prov-links/

**Parameters** **bundle_identifier** (*prov.model.Identifier*) – The identifier of the bundle

**Returns** The list with the bundle nodes and all connections where the start node and end node in the bundle.

**Return type** list(*DbRelation* or *DbRecord* )

**delete_records_by_filter**(*attributes_dict=None*, *metadata_dict=None*)
Delete a set of records based on filter conditions

**Parameters**

- **attributes_dict** (*dict*) – A filter dict with a conjunction of all values in the attributes_dict and metadata_dict
- **metadata_dict** (*dict*) – A filter for the metadata with a conjunction of all values (also in the attributes_dict )

**Returns** The result of the operation

**Return type** Bool

**delete_record**(*record_id*)
Delete a single record

**Parameters** **record_id** (*str*) – The node id

**Returns** Result of the delete operation

**Return type** Bool

**delete_relation**(*relation_id*)
Delete the relation

**Parameters** **relation_id** (*str*) – The relation id

**Returns** Result of the delete operation

**Return type** Bool

### Module contents

### provdbconnector.db_adapters.neo4j package

### Submodules

### provdbconnector.db_adapters.neo4j.cypher_commands module

### provdbconnector.db_adapters.neo4j.neo4jadapter module

**class** provdbconnector.db_adapters.neo4j.neo4jadapter.**Neo4jAdapter**(**args*)
Bases: *provdbconnector.db_adapters.baseadapter.BaseAdapter*

This is the neo4j adapter to store prov. data in a neo4j database

**connect**(*authentication_options*)
The connect method to create a new instance of the db_driver

**Parameters** **authentication_options** – Username, password, host and encrypted option

**Returns** None

**Return type** None

**Raises** InvalidOptionsException

**save_element**(*attributes*, *metadata*)
Saves a single record

**Parameters**

- **attributes** (*dict*) – The attributes dict
- **metadata** (*dict*) – The metadata dict

**Returns** The id of the record

**Return type** str

**save_relation**(*from_node*, *to_node*, *attributes*, *metadata*)
Save a single relation

**Parameters**

- **from_node** (*QualifiedName*) – The from node as QualifiedName
- **to_node** (*QualifiedName*) – The to node as QualifiedName
- **attributes** (*dict*) – The attributes dict
- **metadata** (*dict*) – The metadata dict

**Returns** Id of the relation

**Return type** str

**get_records_by_filter**(*attributes_dict=None*, *metadata_dict=None*)
Return the records by a certain filter

**Parameters**

- **attributes_dict** (*dict*) – Filter dict
- **metadata_dict** (*dict*) – Filter dict for metadata

**Returns** list of all nodes and relations that fit the conditions

**Return type** list(DbRecord and DbRelation)

**get_records_tail**(*attributes_dict=None*, *metadata_dict=None*, *depth=None*)
Return all connected nodes form the origin.

**Parameters**

- **attributes_dict** (*dict*) – Filter dict
- **metadata_dict** (*dict*) – Filter dict for metadata
- **depth** – Max steps

**Returns** list of all nodes and relations that fit the conditions

**Return type** list(DbRecord and DbRelation)

**get_bundle_records**(*bundle_identifier*)
Return all records and relations for the bundle

**Parameters** **bundle_identifier** –

**Returns**

**get_record**(*record_id*)
Try to find the record in the database

**Parameters** **record_id** –

**Returns** DbRecord

**Return type** *DbRecord*

**get_relation**(*relation_id*)
Get a relation

**Parameters** **relation_id** –

**Returns** The relation

**Return type** *DbRelation*

**delete_records_by_filter**(*attributes_dict=None*, *metadata_dict=None*)
Delete records and relations by a filter

**Parameters**

- **attributes_dict** –
- **metadata_dict** –

**Returns**

**delete_record**(*record_id*)
Delete a single record

**Parameters** **record_id** –

**Returns**

**delete_relation**(*relation_id*)
Delete a single relation

**Parameters** **relation_id** –

**Returns**

### Module contents

### Submodules

### provdbconnector.db_adapters.baseadapter module

**class** provdbconnector.db_adapters.baseadapter.**DbDocument**(*document*, *bundles*)
Bases: tuple

**bundles**
Alias for field number 1

**document**
Alias for field number 0

**class** provdbconnector.db_adapters.baseadapter.**DbBundle**(*records*, *bundle_record*)
Bases: tuple

**bundle_record**
Alias for field number 1

**records**
Alias for field number 0

**class** provdbconnector.db_adapters.baseadapter.**DbRecord**(*attributes*, *metadata*)
Bases: tuple

**attributes**
Alias for field number 0

**metadata**
Alias for field number 1

**class** provdbconnector.db_adapters.baseadapter.**DbRelation**(*attributes*, *metadata*)
Bases: tuple

**attributes**
Alias for field number 0

**metadata**
Alias for field number 1

**class** provdbconnector.db_adapters.baseadapter.**BaseAdapter**(**args*, ***kwargs*)
Bases: object

Interface class for a prov database adapter

**connect**(*authentication_info*)
Establish the database connection / login into the database

**Parameters** **authentication_info** (*dict*) – a custom dict with credentials

**Returns** Indicate whether the connection was successful

**Return type** boolean

**Raises** ***InvalidOptionsException*** –

**save_element**(*attributes*, *metadata*)
Saves a entity, activity or entity into the database

**Parameters**

- **attributes** (*dict*) – Attributes as dict for the record. Be careful you have to encode the dict
- **metadata** (*dict*) – Metadata as dict for the record. Be careful you have to encode the dict but you can be sure that all meta keys are always there

**Returns** Record id

**Return type** str

**save_relation**(*from_node*, *to_node*, *attributes*, *metadata*)
Create a relation between 2 nodes

**Parameters**

- **from_node** (*str*) – The identifier
- **to_node** – The identifier for the destination node
- **attributes** (*dict*) – Attributes as dict for the record. Be careful you have to encode the dict
- **metadata** (*dict*) – Metadata as dict for the record. Be careful you have to encode the dict but you can be sure that all meta keys are always there

**Type** to_node: str

**Returns** Record id

**Return type** str

**get_records_by_filter**(*attributes_dict=None*, *metadata_dict=None*)
Returns all records (nodes and relations) based on a filter dict. The filter dict's are and AND combination but only the start node must fulfill the conditions. The result should contain all associated relations and nodes together

**Parameters**

- **attributes_dict** (*dict*) –
- **metadata_dict** (*dict*) –

**Returns** list of relations and nodes

**Return type** list

**get_records_tail**(*attributes_dict=None*, *metadata_dict=None*, *depth=None*)
Returns all connected nodes and relations based on a filter. The filter is an AND combination and this describes the filter only for the origin nodes.

**Parameters**

- **attributes_dict** (*dict*) –
- **metadata_dict** (*dict*) –
- **depth** (*int*) –

**Returns** a list of relations and nodes

**Return type** list

**get_bundle_records**(*bundle_identifier*)
Returns the relations and nodes for a specific bundle identifier. Please use the bundle association to get all bundle nodes. Only the relations belongs to the bundle where the start AND end node belong also to the bundle. Except the prov:Mention see: W3C bundle links

**Parameters** **bundle_identifier** (*str*) – The bundle identifier

**Returns** list of nodes and bundles

**Return type** list

**get_record**(*record_id*)
Return a single record

**Parameters** **record_id** (*str*) – The id

**Returns** DbRecord

**Return type** *DbRecord*

**get_relation**(*relation_id*)
Returns a single relation

**Parameters** **relation_id** (*str*) – The id

**Returns** DbRelation

**Return type** *DbRelation*

**delete_records_by_filter**(*attributes_dict*, *metadata_dict*)
Delete records by filter

**Parameters**

- **attributes_dict** (*dict*) –

- **metadata_dict** (*dict*) –

**Returns** Indicates whether the deletion was successful

**Return type** boolean

**Raises** ***NotFoundException*** –

**delete_record**(*record_id*)
Delete a single record

**Parameters** **record_id** (*str*) –

**Returns** Indicates whether the deletion was successful

**Return type** boolean

**Raises** ***NotFoundException*** –

**delete_relation**(*relation_id*)
Delete a single relation

**Parameters** **relation_id** (*str*) –

**Returns** Indicates whether the deletion was successful

**Return type** boolean

**Raises** ***NotFoundException*** –

#### Module contents

### 5.1.2 provdbconnector.exceptions package

#### Submodules

#### provdbconnector.exceptions.database module

**exception** provdbconnector.exceptions.database.**AdapterException**
Bases: *provdbconnector.exceptions.provapi.ProvDbException*

Base exception class for database adapters.

**exception** provdbconnector.exceptions.database.**InvalidOptionsException**
Bases: *provdbconnector.exceptions.database.AdapterException*

Thrown, if passed argument for adapter is invalid.

**exception** provdbconnector.exceptions.database.**AuthException**
Bases: *provdbconnector.exceptions.database.AdapterException*

Thrown, if database adapter could not establish a connection with given credentials to the database.

**exception** provdbconnector.exceptions.database.**DatabaseException**
Bases: *provdbconnector.exceptions.database.AdapterException*

Thrown, if method could not performed on database.

**exception** provdbconnector.exceptions.database.**CreateRecordException**
Bases: *provdbconnector.exceptions.database.DatabaseException*

Thrown, if record could not be saved in database.

**exception** provdbconnector.exceptions.database.**CreateRelationException**
Bases: *provdbconnector.exceptions.database.DatabaseException*

Thrown, if relation could not be saved in database.

**exception** provdbconnector.exceptions.database.**NotFoundException**
Bases: *provdbconnector.exceptions.database.DatabaseException*

Thrown, if record or relation could not be found in database.

**exception** provdbconnector.exceptions.database.**MergeException**
Bases: *provdbconnector.exceptions.database.DatabaseException*

Thrown, if a record or relation can't get merged

### provdbconnector.exceptions.provapi module

**exception** provdbconnector.exceptions.provapi.**ProvDbException**
Bases: Exception

Base exception class for all api exceptions.

**exception** provdbconnector.exceptions.provapi.**NoDataBaseAdapterException**
Bases: *provdbconnector.exceptions.provapi.ProvDbException*

Thrown, if no database adapter argument is passed to the api class.

**exception** provdbconnector.exceptions.provapi.**InvalidArgumentTypeException**
Bases: *provdbconnector.exceptions.provapi.ProvDbException*

Thrown, if an invalid argument is passed to any api method.

**exception** provdbconnector.exceptions.provapi.**InvalidProvRecordException**
Bases: *provdbconnector.exceptions.provapi.ProvDbException*

" Thrown, if an invalid record is passed to any api method.

### provdbconnector.exceptions.utils module

**exception** provdbconnector.exceptions.utils.**ConverterException**
Bases: *provdbconnector.exceptions.provapi.ProvDbException*

Base exception class for document converter.

**exception** provdbconnector.exceptions.utils.**ParseException**
Bases: *provdbconnector.exceptions.utils.ConverterException*

Thrown, if a given statement could not ne parsed.

**exception** provdbconnector.exceptions.utils.**NoDocumentException**
Bases: *provdbconnector.exceptions.utils.ConverterException*

Thrown, if no document argument is passed.

**exception** provdbconnector.exceptions.utils.**SerializerException**
Bases: *provdbconnector.exceptions.provapi.ProvDbException*

Base exception class for serializer.

**exception** provdbconnector.exceptions.utils.**ValidatorException**
Bases: *provdbconnector.exceptions.provapi.ProvDbException*

Base exception class for validator.

#### Module contents

### 5.1.3 provdbconnector.tests package

#### Subpackages

#### provdbconnector.tests.db_adapters package

#### Subpackages

#### provdbconnector.tests.db_adapters.in_memory package

#### Submodules

#### provdbconnector.tests.db_adapters.in_memory.test_simple_in_memory module

**class** provdbconnector.tests.db_adapters.in_memory.test_simple_in_memory.**SimpleInMemoryAdapt**

Bases: *provdbconnector.tests.db_adapters.test_baseadapter.AdapterTestTemplate*

This class implements the AdapterTestTemplate and only override some functions.

**setUp**()
Connect to your database

**test_connect_invalid_options**()
Test your connect function with invalid data

**clear_database**()
Clear the database

**tearDown**()
Delete your instance

**class** provdbconnector.tests.db_adapters.in_memory.test_simple_in_memory.**SimpleInMemoryAdapt**

Bases: *provdbconnector.tests.test_prov_db.ProvDbTestTemplate*

This is the high level test for the SimpleInMemoryAdapter

**setUp**()
Setup a ProvDb instance

**clear_database**()
Clear function get called before each test starts

**tearDown**()
Delete prov api instance

#### Module contents

#### provdbconnector.tests.db_adapters.neo4j package

#### Submodules

### provdbconnector.tests.db_adapters.neo4j.test_neo4jadapter module

**class** provdbconnector.tests.db_adapters.neo4j.test_neo4jadapter.**Neo4jAdapterTests**(**args*, ***kwargs*)

Bases: *provdbconnector.tests.db_adapters.test_baseadapter.AdapterTestTemplate*

This test extends from AdapterTestTemplate and provide a common set for the neo4j adapter

**setUp**()
Setup the test

**test_connect_fails**()
Try to connect with the wrong password

**test_connect_invalid_options**()
Try to connect with some invalid arguments

**tearDown**()
Delete all data on the database :return:

**class** provdbconnector.tests.db_adapters.neo4j.test_neo4jadapter.**Neo4jAdapterProvDbTests**(**ar*, ***k*

Bases: *provdbconnector.tests.test_prov_db.ProvDbTestTemplate*

High level api test for the neo4j adapter

**setUp**()
Use the setup method to create a provapi instance with you adapter

> **Warning:** Override this function if you extend this test! Otherwise the test will fail.

**Returns**

**clear_database**()
This function get called before each test starts

**tearDown**()
Delete all data in the database

### Module contents

### Submodules

### provdbconnector.tests.db_adapters.test_baseadapter module

provdbconnector.tests.db_adapters.test_baseadapter.**json_serial**(*obj*)
JSON serializer for objects not serializable by default json code

provdbconnector.tests.db_adapters.test_baseadapter.**encode_adapter_result_to_excpect**(*dict_vals*)
This function translate a metadata dict to an expected version of this dict

**Parameters** **dict_vals** –

**Returns**

provdbconnector.tests.db_adapters.test_baseadapter.**insert_document_with_bundles**(*instance*, *identifier_prefix=''*)

This function creates a full bundle on your database adapter, to prepare the test data

**Parameters**

- **instance** – The db_adapter instnace
- **identifier_prefix** – A prefix for the identifiers

**Returns** The ids of the records

**class** provdbconnector.tests.db_adapters.test_baseadapter.**AdapterTestTemplate**(**args*, ***kwargs*)

Bases: unittest.case.TestCase

This test class is a template for each database adapter. The following example show how you implement the test for your adapter:

```
from provdbconnector.exceptions.database import InvalidOptionsException
from provdbconnector.db_adapters.in_memory import SimpleInMemoryAdapter
from provdbconnector.prov_db import ProvDb
from provdbconnector.tests import AdapterTestTemplate
from provdbconnector.tests import ProvDbTestTemplate


class SimpleInMemoryAdapterTest(AdapterTestTemplate):
    """
    This class implements the AdapterTestTemplate and only override some functions.

    """
    def setUp(self):
        """
        Connect to your database

        """
        self.instance = SimpleInMemoryAdapter()
        self.instance.connect(None)

    def test_connect_invalid_options(self):
        """
        Test your connect function with invalid data

        """
```

**maxDiff = None**

**setUp**()

**Setup the instnace of your database adapter**

> **Warning:** Override this method otherwise all test will fail

**Returns**

**clear_database**()
This function is to clear your database adapter before each test :return:

**test_1_save_element**()
This test try to save a simple record

**Graph-Strucutre**

**Input-Data**

> **Warning:** This is a json representation of the input data and not the real input data. For example metadata.prov_type is a QualifiedName instance


```
{
   "metadata":{
      "identifier":"<QualifiedName: prov:example_node>",
      "namespaces":{
         "ex":"http://example.com",
         "custom":"http://custom.com"
      },
      "prov_type":"<QualifiedName: prov:Activity>",
      "type_map":{
         "int value":"int",
         "date value":"xds:datetime"
      }
   },
   "attributes":{
      "ex:individual attribute":"Some value",
      "ex:date value":"datetime.datetime(2005, 6, 1, 13, 33)",
      "ex:dict value":{
         "dict":"value"
      },
      "ex:list value":[
         "list",
         "of",
         "strings"
      ],
      "ex:double value":99.33,
      "ex:int value":99
   }
}
```


**Output-Data**

The output is only a id as string

*4d3cdc76-467d-4db8-89bf-9accc7b27777*

**test_2_save_relation**()
This test try to save a simple relation between 2 identifiers

**Graph-Strucutre**

**Input-Data**

**Warning:** This is a json representation of the input data and not the real input data. For example metadata.prov_type is a QualifiedName instance


```
{

   "from_node":"<QualifiedName: ex:Yoda>",
   "to_node":"<QualifiedName: ex:Luke Skywalker>",
   "metadata":{
      "prov_type":"<QualifiedName: prov:Mention>",
      "type_map":{
         "date value":"xds:datetime",
         "int value":"int"
      },
      "namespaces":{
         "custom":"http://custom.com",
         "ex":"http://example.com"
      },
      "identifier":"identifier for the relation"
   },
   "attributes":{
      "ex:list value":[
         "list",
         "of",
         "strings"
      ],
      "ex:int value":99,
      "ex:double value":99.33,
      "ex:individual attribute":"Some value",
      "ex:dict value":{
         "dict":"value"
      },
      "ex:date value":"datetime.datetime(2005, 6, 1, 13, 33)"
   },
}
```


**Output-Data**

The output is only the id of the relation as string

*4d3cdc76-467d-4db8-89bf-9accc7b27778*

**test_3_save_relation_with_unknown_records**()
This test is to test the creation of a relation where the nodes are not in the database :return:

**test_4_get_record**()
Create a record and then try to get it back

**Graph-Strucutre**

**Input-Data**

*"id-333"*

**Output-Data**

The output is all connected nodes and there relations

```
[
   {
      "ex:int value":99,
      "ex:list value":[
         "list",
         "of",
         "strings"
      ],
      "ex:date value":"2005-06-01 13:33:00",
      "ex:individual attribute":"Some value",
      "ex:double value":99.33,
      "ex:dict value":"{"dict": "value"}"
   },
   {
      "identifier":"prov:example_node",
      "prov_type":"prov:Activity",
      "type_map":"{"date value": "xds:datetime", "int value": "int"}",
      "namespaces":"{"custom": "http://custom.com", "ex": "http://example.com
↪"}"
   }
]
```


**test_5_get_record_not_found**()
Try to get a record with a invalid id :return:

**test_6_get_relation**()
create a relation between 2 nodes and try to get the relation back

**Graph-Strucutre**

**Input-Data**

*"id-333"*

**Output-Data**

The output is all connected nodes and there relations


```
[
   {
      "ex:dict value":"{"dict": "value"}",
      "ex:int value":99,
      "ex:individual attribute":"Some value",
      "ex:date value":"2005-06-01 13:33:00",
      "ex:list value":[
         "list",
         "of",
         "strings"
      ],
      "ex:double value":99.33
   },
   {
      "identifier":"identifier for the relation",
      "prov_type":"prov:Mention",
      "namespaces":"{"ex": "http://example.com", "custom": "http://custom.com
↪"}",
      "type_map":"{"date value": "xds:datetime", "int value": "int"}"
```


(continues on next page)

(continued from previous page)

```
    }
]
```

**test_7_get_relation_not_found**()
Try to get a not existing relation

**test_8_get_records_by_filter**()
This test is to get a the whole graph without any filter

**Graph-Strucutre**

**Input-Data**

We have no input data for the filter function because we want to get the whole graph

**Output-Data**

The output is the only node that exist in the database (was also created during the test)

**Warning:** This is a json representation of the input data and not the real input data. For example metadata.prov_type is a QualifiedName instance


```
{
   "metadata":{
      "identifier":"<QualifiedName: prov:example_node>",
      "namespaces":{
         "ex":"http://example.com",
         "custom":"http://custom.com"
      },
      "prov_type":"<QualifiedName: prov:Activity>",
      "type_map":{
         "int value":"int",
         "date value":"xds:datetime"
      }
   },
   "attributes":{
      "ex:individual attribute":"Some value",
      "ex:date value":"datetime.datetime(2005, 6, 1, 13, 33)",
      "ex:dict value":{
         "dict":"value"
      },
      "ex:list value":[
         "list",
         "of",
         "strings"
      ],
      "ex:double value":99.33,
      "ex:int value":99
   }
}
```


**test_9_get_records_by_filter_with_properties**()
This test is to get a specific part of the graph via certain filter criteria

**Graph-Strucutre**

*Get single node*

The first part of the test is to try to get a single node based on a attribut

*Input-Data*

```
{
    "prov:type":"prov:Bundle"
}
```

*Output-Data*

The output is a list of namedtuples wit the following structure: *list(tuple(attributes,metadata))*

```
[
    [
        {
            "prov:type": "prov:Bundle"
        },
        {
            "prov_type": "prov:Entity",
            "identifier": "ex:bundle name",
            "type_map": "{\"date value\": \"xds:datetime\", \"int value\": \
↪"int\"}",
            "namespaces": "{\"ex\": \"http://example.com\"}"
        }
    ]
]
```

*Get other nodes*

The second part tests the other way to get all other node except the bundle node

*Input-Data*

The input is a set of attributes

```
{
   "ex:dict value":{
      "dict":"value"
   },
   "ex:double value":99.33,
   "ex:int value":99,
   "ex:individual attribute":"Some value",
   "ex:list value":[
      "list",
      "of",
      "strings"
   ]
}
```

*Output-Data*

The output is a list of namedtuple with attributes and metadata

```
[
   [
      {
         "ex:dict value":"{'dict': 'value'}",
```

(continues on next page)

(continued from previous page)


```
        "ex:double value":99.33,
        "ex:list value":[
           "list",
           "of",
           "strings"
        ],
        "ex:int value":99,
        "ex:date value":"2005-06-01 13:33:00",
        "ex:individual attribute":"Some value"
     },
     {
        "identifier":"ex:TO NODE",
        "type_map":"{'int value': 'int', 'date value': 'xds:datetime'}",
        "namespaces":"{'ex': 'http://example.com', 'custom': 'http://custom.
↪com'}",
        "prov_type":"prov:Activity"
     }
  ],
  [
     {
        "ex:dict value":"{'dict': 'value'}",
        "ex:double value":99.33,
        "ex:list value":[
           "list",
           "of",
           "strings"
        ],
        "ex:int value":99,
        "ex:date value":"2005-06-01 13:33:00",
        "ex:individual attribute":"Some value"
     },
     {
        "identifier":"ex:FROM NODE",
        "type_map":"{'int value': 'int', 'date value': 'xds:datetime'}",
        "namespaces":"{'ex': 'http://example.com', 'custom': 'http://custom.
↪com'}",
        "prov_type":"prov:Activity"
     }
  ],
  [
     {
        "ex:dict value":"{'dict': 'value'}",
        "ex:double value":99.33,
        "ex:list value":[
           "list",
           "of",
           "strings"
        ],
        "ex:int value":99,
        "ex:date value":"2005-06-01 13:33:00",
        "ex:individual attribute":"Some value"
     },
     {
        "identifier":"ex:prov:example_node",
        "type_map":"{'int value': 'int', 'date value': 'xds:datetime'}",
        "namespaces":"{'ex': 'http://example.com', 'custom': 'http://custom.
↪com'}",
```


(continues on next page)

(continued from previous page)

```
            "prov_type":"prov:Activity"
        }
    ],
    [
        {
            "ex:dict value":"{'dict': 'value'}",
            "ex:double value":99.33,
            "ex:list value":[
                "list",
                "of",
                "strings"
            ],
            "ex:int value":99,
            "ex:date value":"2005-06-01 13:33:00",
            "ex:individual attribute":"Some value"
        },
        {
            "identifier":"identifier for the relation",
            "type_map":"{'int value': 'int', 'date value': 'xds:datetime'}",
            "namespaces":"{'ex': 'http://example.com', 'custom': 'http://custom.
↪com'}",
            "prov_type":"prov:Mention"
        }
    ]
]
```

:return

**test_10_get_records_by_filter_with_metadata**()

Should test also the filter by metadata

@todo implement test for filter by metadata

**Graph-Strucutre**

> **Warning:** This test is not implemented jet

**Returns**

**test_11_get_records_tail**()

This test is to get the whole provenance from a starting point

**Graph-Strucutre**

**Input-Data**

In this case we filter by metadata and by the identifier

```
{
    "identifier":"ex:FROM NODE"
}
```

**Output-Data**

The output is all connected nodes and there relations

```
[
   [
      {
         "ex:list value":[
            "list",
            "of",
            "strings"
         ],
         "ex:double value":99.33,
         "ex:int value":99,
         "ex:individual attribute":"Some value",
         "ex:date value":"datetime.datetime(2005, 6, 1, 13, 33)",
         "ex:dict value":{
            "dict":"value"
         }
      },
      {
         "prov_type":"<QualifiedName: prov:Mention>",
         "identifier":"identifier for the relation",
         "type_map":{
            "date value":"xds:datetime",
            "int value":"int"
         },
         "namespaces":{
            "custom":"http://custom.com",
            "ex":"http://example.com"
         }
      }
   ],
   [
      {
         "ex:list value":[
            "list",
            "of",
            "strings"
         ],
         "ex:double value":99.33,
         "ex:int value":99,
         "ex:individual attribute":"Some value",
         "ex:date value":"datetime.datetime(2005, 6, 1, 13, 33)",
         "ex:dict value":{
            "dict":"value"
         }
      },
      {
         "prov_type":"<QualifiedName: prov:Activity>",
         "identifier":"<QualifiedName: ex:TO NODE>",
         "type_map":{
            "date value":"xds:datetime",
            "int value":"int"
         },
         "namespaces":{
            "custom":"http://custom.com",
            "ex":"http://example.com"
         }
      }
   ]
]
```

**test_12_get_records_tail_recursive**()

Test the same behavior as the *test_get_records_tail* test but with a recursive data structure

**Graph-Strucutre**

**Input-Data**

```
{
   "identifier":"ex:FROM NODE"
}
```

**Output-Data**

The output is all connected nodes and there relations

```
[
   [
      {
         "ex:dict value":{
            "dict":"value"
         },
         "ex:date value":"datetime.datetime(2005, 6, 1, 13, 33)",
         "ex:double value":99.33,
         "ex:individual attribute":"Some value",
         "ex:int value":99,
         "ex:list value":[
            "list",
            "of",
            "strings"
         ]
      },
      {
         "namespaces":{
            "custom":"http://custom.com",
            "ex":"http://example.com"
         },
         "identifier":"identifier for the relation",
         "prov_type":"<QualifiedName: prov:Mention>",
         "type_map":{
            "date value":"xds:datetime",
            "int value":"int"
         }
      }
   ],
   [
      {
         "ex:dict value":{
            "dict":"value"
         },
         "ex:date value":"datetime.datetime(2005, 6, 1, 13, 33)",
         "ex:double value":99.33,
         "ex:individual attribute":"Some value",
         "ex:int value":99,
         "ex:list value":[
            "list",
            "of",
            "strings"
         ]
```

(continues on next page)

(continued from previous page)


```
    },
    {
        "namespaces":{
            "custom":"http://custom.com",
            "ex":"http://example.com"
        },
        "identifier":"identifier for the relation",
        "prov_type":"<QualifiedName: prov:Mention>",
        "type_map":{
            "date value":"xds:datetime",
            "int value":"int"
        }
    }
],
[
    {
        "ex:dict value":{
            "dict":"value"
        },
        "ex:date value":"datetime.datetime(2005, 6, 1, 13, 33)",
        "ex:double value":99.33,
        "ex:individual attribute":"Some value",
        "ex:int value":99,
        "ex:list value":[
            "list",
            "of",
            "strings"
        ]
    },
    {
        "namespaces":{
            "custom":"http://custom.com",
            "ex":"http://example.com"
        },
        "identifier":"<QualifiedName: ex:TO NODE>",
        "prov_type":"<QualifiedName: prov:Activity>",
        "type_map":{
            "date value":"xds:datetime",
            "int value":"int"
        }
    }
],
[
    {
        "ex:dict value":{
            "dict":"value"
        },
        "ex:date value":"datetime.datetime(2005, 6, 1, 13, 33)",
        "ex:double value":99.33,
        "ex:individual attribute":"Some value",
        "ex:int value":99,
        "ex:list value":[
            "list",
            "of",
            "strings"
        ]
    },
```


(continues on next page)

(continued from previous page)


```
    {
        "namespaces":{
            "custom":"http://custom.com",
            "ex":"http://example.com"
        },
        "identifier":"<QualifiedName: ex:second_TO NODE>",
        "prov_type":"<QualifiedName: prov:Activity>",
        "type_map":{
            "date value":"xds:datetime",
            "int value":"int"
        }
    }
],
[
    {
        "ex:dict value":{
            "dict":"value"
        },
        "ex:date value":"datetime.datetime(2005, 6, 1, 13, 33)",
        "ex:double value":99.33,
        "ex:individual attribute":"Some value",
        "ex:int value":99,
        "ex:list value":[
            "list",
            "of",
            "strings"
        ]
    },
    {
        "namespaces":{
            "custom":"http://custom.com",
            "ex":"http://example.com"
        },
        "identifier":"identifier for the relation",
        "prov_type":"<QualifiedName: prov:Mention>",
        "type_map":{
            "date value":"xds:datetime",
            "int value":"int"
        }
    }
],
[
    {
        "ex:dict value":{
            "dict":"value"
        },
        "ex:date value":"datetime.datetime(2005, 6, 1, 13, 33)",
        "ex:double value":99.33,
        "ex:individual attribute":"Some value",
        "ex:int value":99,
        "ex:list value":[
            "list",
            "of",
            "strings"
        ]
    },
    {
```


(continues on next page)

(continued from previous page)


```
        "namespaces":{
           "custom":"http://custom.com",
           "ex":"http://example.com"
        },
        "identifier":"<QualifiedName: ex:FROM NODE>",
        "prov_type":"<QualifiedName: prov:Activity>",
        "type_map":{
           "date value":"xds:datetime",
           "int value":"int"
        }
    }
],
[
    {
        "ex:dict value":{
           "dict":"value"
        },
        "ex:date value":"datetime.datetime(2005, 6, 1, 13, 33)",
        "ex:double value":99.33,
        "ex:individual attribute":"Some value",
        "ex:int value":99,
        "ex:list value":[
           "list",
           "of",
           "strings"
        ]
    },
    {
        "namespaces":{
           "custom":"http://custom.com",
           "ex":"http://example.com"
        },
        "identifier":"identifier for the relation",
        "prov_type":"<QualifiedName: prov:Mention>",
        "type_map":{
           "date value":"xds:datetime",
           "int value":"int"
        }
    }
],
[
    {
        "ex:dict value":{
           "dict":"value"
        },
        "ex:date value":"datetime.datetime(2005, 6, 1, 13, 33)",
        "ex:double value":99.33,
        "ex:individual attribute":"Some value",
        "ex:int value":99,
        "ex:list value":[
           "list",
           "of",
           "strings"
        ]
    },
    {
        "namespaces":{
```


(continues on next page)

(continued from previous page)


```
            "custom":"http://custom.com",
            "ex":"http://example.com"
         },
         "identifier":"<QualifiedName: ex:second_FROM NODE>",
         "prov_type":"<QualifiedName: prov:Activity>",
         "type_map":{
            "date value":"xds:datetime",
            "int value":"int"
         }
      }
   ]
]
```


**test_13_get_bundle_records**()

The get_bundle function is to return the records (relation and nodes) for a bundle identifier

Test the same behavior as the *test_get_records_tail* test but with a recursive data structure

**Graph-Strucutre**

**Input-Data**


```
{
   "identifier":"ex:FROM NODE"
}
```


**Output-Data**

The output is all connected nodes and there relations

> **Warning:** coming soon!

**test_14_delete_by_filter**()

Try to all records of the database

**Returns**

**test_15_delete_by_filter_with_properties**()

Try to delete by filter, same behavior as get_by_filter

**Returns**

**test_16_delete_by_filter_with_metadata**()

Try to delete by metadata, same behavior as get_by_metadata :return:

**test_17_delete_record**()

Delete a single record based on the id

**Returns**

**test_18_delete_relation**()

Delete a singe relation based on the relation id

**Returns**

**test_19_merge_record**()

This function test the merge abbility of your adapter.

#### Graph-Strucutre

#### Input-Data

We try to create the node twice, with the following data

```
{
   "metadata":{
      "prov_type":"<QualifiedName: prov:Activity>",
      "namespaces":{
         "ex":"http://example.com",
         "custom":"http://custom.com"
      },
      "identifier":"<QualifiedName: ex:Yoda>",
      "type_map":{
         "int value":"int",
         "date value":"xds:datetime"
      }
   },
   "attributes":{
      "ex:int value":99,
      "ex:individual attribute":"Some value",
      "ex:list value":[
         "list",
         "of",
         "strings"
      ],
      "ex:date value":"datetime.datetime(2005, 6, 1, 13, 33)",
      "ex:dict value":{
         "dict":"value"
      },
      "ex:double value":99.33
   }
}
```

#### Output-Data

The output is one entry with no change of the data

```
[
   {
      "ex:int value":99,
      "ex:individual attribute":"Some value",
      "ex:list value":[
         "list",
         "of",
         "strings"
      ],
      "ex:date value":"2005-06-01 13:33:00",
      "ex:dict value":"{"dict": "value"}",
      "ex:double value":99.33
   },
   {
      "prov_type":"prov:Activity",
      "namespaces":"{"ex": "http://example.com", "custom": "http://custom.com
↪"}",
      "identifier":"ex:Yoda",
```

(continues on next page)

(continued from previous page)

```
      "type_map":"{"int value": "int", "date value": "xds:datetime"}"
   }
]
```

**test_20_merge_record_complex**()

In this example we test if we merge different attributes into one node

**Graph-Strucutre**

**Input-Data**

This is the attributes used to create the entry

```
{
   "ex:individual attribute":"Some value",
   "ex:dict value":{
      "dict":"value"
   },
   "ex:double value":99.33,
   "ex:list value":[
      "list",
      "of",
      "strings"
   ],
   "ex:int value":99,
   "ex:date value":"datetime.datetime(2005, 6, 1, 13, 33)"
}
```

This are the attributes to alter the existing node

```
{
    "ex:a other attribute":true
}
```

**Output-Data**

The output is one entry with the additional attribute

```
[
   {
      "ex:individual attribute":"Some value",
      "ex:dict value":"{"dict": "value"}",
      "ex:double value":99.33,
      "ex:list value":[
         "list",
         "of",
         "strings"
      ],
      "ex:int value":99,
      "ex:date value":"2005-06-01 13:33:00",
      "ex:a other attribute":true
   },
   {
      "type_map":"{"date value": "xds:datetime", "int value": "int"}",
      "identifier":"ex:Yoda",
      "prov_type":"prov:Activity",
```

(continues on next page)

(continued from previous page)

```
      "namespaces":"{"ex": "http://example.com", "custom": "http://custom.com
↪"}"
   }
]
```

**test_21_merge_record_complex_fail**()

In this example we test if we merge different attributes into one node

**Graph-Strucutre**

Fig. 1: **Input-Data**
This is the attributes used to create the entry

```
{
  "ex:list value":[
     "list",
     "of",
     "strings"
  ],
  "ex:double value":99.33,
  "ex:date value":"datetime.datetime(2005, 6, 1, 13, 33)",
  "ex:dict value":{
     "dict":"value"
  },
  "ex:int value":99,
  "ex:individual attribute":"Some value"
}
```

Try to *override* the existing attribute

```
{
    "ex:int value": 1
}
```

**Output-Data**
Should throw an MergeException

**test_22_merge_record_metadata**()

This test try to merge the metadata. This is important if you add some new attributes that uses other namespaces, so you need to merge the namespaces Same behavior for the type_map

**Graph-Strucutre**

**Input-Data**

The metadata for the initial record:

```
{
  "type_map":{
     "date value":"xsd:datetime",
     "int value":"int"
  },
  "prov_type":"<QualifiedName: prov:Activity>",
  "namespaces":{
     "custom":"http://custom.com",
     "ex":"http://example.com"
```

(continues on next page)

(continued from previous page)

```
    },
    "identifier":"<QualifiedName: ex:Yoda>"
}
```

Try to add a record with some modified namespaces

```
{
    "namespaces":{
        "custom":"http://custom.com",
        "ex":"http://example.com"
    },
    "prov_type":"<QualifiedName: prov:Activity>",
    "identifier":"<QualifiedName: ex:Yoda>",
    "type_map":{
        "custom_attr_1":"xds:some_value"
    }
}
```

**Output-Data**

The output is the merged result of the type map

```
[
    {
        "ex:individual attribute":"Some value",
        "ex:list value":[
            "list",
            "of",
            "strings"
        ],
        "ex:int value":99,
        "ex:date value":"2005-06-01 13:33:00",
        "ex:dict value":"{"dict": "value"}",
        "ex:double value":99.33
    },
    {
        "prov_type":"prov:Activity",
        "type_map":"{"custom_attr_1": "xds:some_value", "date value":
↪"xds:datetime", "int value": "int"}",
        "identifier":"ex:Yoda",
        "namespaces":"{"custom": "http://custom.com", "ex": "http://example.com
↪"}"
    }
]
```

**test_23_merge_relation**()

Merge a relation is pretty similar to merge records. The big difference is the different rules for uniques

**Graph-Strucutre**

A relation is unique if:

- The relation type is the same
- all other formal attributes (see SimpleDbAdapter) are the same

otherwise it is not the same relation.

**test_24_merge_relation_complex**()
Same behavior as the merge_node test

**test_25_merge_relation_complex_fail**()
Same behavior as the merge_node_fail test

**test_26_merge_relation_metadata**()
Same as the merge_record_metadata

**test_27_save_bundle**()

**Returns**

**class** provdbconnector.tests.db_adapters.test_baseadapter.**BaseConnectorTests**(*methodName='runTe*
Bases: unittest.case.TestCase

This class is only to test that the BaseConnector is alright

**test_instance_abstract_class**()
Test that the BaseAdapter is abstract

### Module contents

### provdbconnector.tests.utils package

### Submodules

### provdbconnector.tests.utils.test_converter module

**class** provdbconnector.tests.utils.test_converter.**ConverterTests**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test the convert class

**setUp**()
Hook method for setting up the test fixture before exercising it.

**tearDown**()
Close all files

**test_form_string**()
Test the convert from string

**test_to_json**()
Test the convert to json

**test_from_json**()
Test the convert from json

**test_to_provn**()
Test the convert to prov-n

**test_from_provn**()
Test the convert from prov-n

**test_to_xml**()
Test the convert to xml

**test_from_xml**()
Test the convert from xml

### provdbconnector.tests.utils.test_validator module

**class** provdbconnector.tests.utils.test_validator.**ValidatorTests**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test the validator class

**setUp**()
Setup validator

**tearDown**()
Delete validator

### Module contents

### Submodules

### provdbconnector.tests.examples module

provdbconnector.tests.examples.**prov_db_unknown_prov_typ_example**()

provdbconnector.tests.examples.**prov_default_namespace_example**(*ns_postfix: str*)

provdbconnector.tests.examples.**attributes_dict_example**()
Retuns a example dict with some different attributes

**Returns** dict with attributes

**Return type** dict

provdbconnector.tests.examples.**base_connector_bundle_parameter_example**()
This example returns a dict with example arguments for a db_adapter

**Returns** dict {attributes, metadata}

**Return type** dict

provdbconnector.tests.examples.**base_connector_record_parameter_example**()
Returns a dict with attributes and metadata for a simple node

:return:dict with attributes metadata :rtype: dict

provdbconnector.tests.examples.**base_connector_relation_parameter_example**()
Returns a example with a start nodes (attributes, metadata) and also a relation dict with attributes metadata

**Returns** dict

**Return type** dict

provdbconnector.tests.examples.**base_connector_merge_example**()
This example returns a namedtuple with a from_node relation and to_node to test the merge behavior

**Returns** namedtuple(from_node, relation, to_node)

**Return type** namedtuple

provdbconnector.tests.examples.**prov_api_record_example**()
This is a more complex record example

**Returns**

### provdbconnector.tests.test_prov_db module

**class** provdbconnector.tests.test_prov_db.**ProvDbTestTemplate**(**args*, ***kwargs*)
Bases: unittest.case.TestCase

This abstract test class to test the high level function of you database adapter. To use this unitest Template extend from this class.

```
        """
        auth_info = {"invalid": "Invalid"}
        with self.assertRaises(InvalidOptionsException):
            self.instance.connect(auth_info)

    def clear_database(self):
        """
        Clear the database

        """
        self.instance.all_nodes = dict()
        self.instance.all_relations= dict()

    def tearDown(self):
        """
        Delete your instance
```

**setUp**()
Use the setup method to create a provapi instance with you adapter

> **Warning:** Override this function if you extend this test! Otherwise the test will fail.

**Returns**

**clear_database**()
Override this function to clear your database before each test

**Returns**

**test_prov_primer_example**()
This test try to save and restore a common prov example document

**Returns**

**test_primer_example_alternate**()
This test try to save and restore a common prov example document. But in a more complex way

**Returns**

**test_w3c_publication_1**()
This test try to save and restore a common prov example document.

**Returns**

**test_w3c_publication_2**()
This test try to save and restore a common prov example document.

**Returns**

**test_bundles1**()
This test try to save and restore a common prov example document. With a bundle and some connections inside the bundle. This example is also available via *Provstore <https://provenance.ecs.soton.ac.uk/store/documents/114710/>*

**Returns**

**test_bundles2**()
This test try to save and restore a common prov example document. With a bundle and some connections inside the bundle. This example is also available via *Provstore <https://provenance.ecs.soton.ac.uk/store/documents/114704/>*

The main difference to the bundle_1 is that here we have also a mentionOf connection between bundles. See PROV-Links spec for more information

**Returns**

**test_collections**()
This test try to save and restore a common prov example document.

**Returns**

**test_long_literals**()
This test try to save and restore a common prov example document.

**Returns**

**test_datatypes**()
This test try to save and restore a common prov example document.

**Returns**

**class** provdbconnector.tests.test_prov_db.**ProvDbTests**(*methodName='runTest'*)
Bases: unittest.case.TestCase

This tests are only for the ProvDb itself. You don't have to extend this test in case you want to write your own adapter

**maxDiff = None**

**setUp**()
Loads the test xml json and provn data

**clear_database**()

**tearDown**()
Destroy the prov api and remove all data from neo4j :return:

**test_provapi_instance**()
Try to create a test instnace :return:

**test_save_document_from_json**()
Try to create a document from a json buffer :return:

**test_get_document_as_json**()
try to get the document as json :return:

**test_save_document_from_xml**()
Try to create a document from xml :return:

**test_get_document_as_xml**()
try to get the document as xml :return:

**test_save_document_from_provn**()
Try to create a document from provn :return:

**test_get_document_as_provn**()
Try to get a document in provn :return:

**test_save_document**()
Try to create a document from a prov instnace :return:

**test_save_document_from_prov**()
Try to create a primer example document :return:

**test_save_document_from_prov_alternate**()
Try to create a prov_alternative :return:

**test_save_document_from_prov_bundles**()
Try to create a document with bundles :return:

**test_save_document_from_prov_bundles2**()
Try to create more bundles :return:

**test_save_document_from_prov_invalid_arguments**()
Try to create a prov with some invalid arguments :return:

**test_get_document_as_prov**()
Try to get the document as ProvDocument instnace

**Returns**

**test_get_document_as_prov_invalid_arguments**()
Try to get the prov document with invalid arguments

**Returns**

**test_save_bundle_invalid_arguments**()
Try to create a bundle with invalid arguments :return:

**test_save_element_invalid**()
Test save_element with invalid args

**test_save_record**()
Test to save a record (a element or a relation)

**test_save_record_invalid**()

**test_save_element**()
Try to save a single record without document_di

**test_get_elements**()
Test for the get_elements function

**test_get_element_invalid**()
Test get element with error

**test_get_element**()
Try to save a single record without document_id and get the record back from the db

**test_save_bundle**()
Test the public method to save bundles

**test_save_bundle_invalid**()
Test the public method to save bundles with invalid arguments

**test_get_bundle**()
Test the public method to get bundles

**test_get_bundle_invalid**()
Test with invalid arguemnts

**test_save_relation_with_unknown_nodes**()
Test to create a relation were the start and end node dose not exist This should also work

**test_save_relation_invalid**()

**test_get_metadata_and_attributes_for_record_invalid_arguments**()
Try to get attributes and metadata with invalid arguments :return:

**test_save_unknown_prov_typ**()
Test to prefer non unknown prov type

**test_save_with_override_default_namespace**()
Test to support default namespace overrides

**test_get_metadata_and_attributes_for_record**()
Test the split into metadata / attributes function This function separates the attributes and metadata from a prov record :return:

#### Module contents

provdbconnector.tests.**additional_tests**()

### 5.1.4 provdbconnector.utils package

#### Submodules

#### provdbconnector.utils.converter module

provdbconnector.utils.converter.**form_string**(*content*)
Take a string or BufferedReader as argument and transform the string into a ProvDocument

**Parameters** **content** – Takes a sting or BufferedReader

**Returns** ProvDocument

provdbconnector.utils.converter.**to_json**(*document=None*)
Try to convert a ProvDocument into the json representation

**Parameters** **document** (*prov.model.ProvDocument*) –

**Returns** Json string of the document

**Return type** str

provdbconnector.utils.converter.**from_json**(*json=None*)
Try to convert a json string into a document

**Parameters** **json** (*str*) – The json str

**Returns** Prov Document

**Return type** prov.model.ProvDocument

**Raise** NoDocumentException

provdbconnector.utils.converter.**to_provn**(*document=None*)
Try to convert a document into a provn representation

**Parameters** **document** (*prov.model.ProvDocument*) – Prov document to convert

**Returns** The prov-n str

**Return type** str

**Raise** NoDocumentException

provdbconnector.utils.converter.**from_provn**(*provn_str=None*)
Try to convert a provn string into a ProvDocument

**Parameters** **provn_str** (*str*) – The string to convert

**Returns** The Prov document

**Return type** ProvDocument

**Raises** NoDocumentException

provdbconnector.utils.converter.**to_xml**(*document=None*)
Try to convert a document into an xml string

**Parameters**

- **document** – The ProvDocument to convert
- **document** – ProvDocument

**Returns** The xml string

**Return type** str

provdbconnector.utils.converter.**from_xml**(*xml_str=None*)
Try to convert a xml string into a ProvDocument

**Parameters** **xml_str** (*str*) – The xml string

**Returns** The Prov document

**Return type** ProvDocument

### provdbconnector.utils.serializer module

provdbconnector.utils.serializer.**FormalAndOtherAttributes**
alias of provdbconnector.utils.serializer.formal_and_other_attributes

provdbconnector.utils.serializer.**encode_dict_values_to_primitive**(*dict_values*)
This function transforms a dict with all kind of types into a dict with only

- str
- dict
- book
- str

values

**Parameters** **dict_values** –

**Returns**

provdbconnector.utils.serializer.**encode_string_value_to_primitive**(*value*)
Convert a value into one of the following types:

- dict
- str
- float

- int
- list

**Parameters** **value** –

**Returns**

provdbconnector.utils.serializer.**literal_json_representation**(*literal*)
Some internationalization stuff

**Parameters** **literal** –

**Returns**

provdbconnector.utils.serializer.**encode_json_representation**(*value*)
Get the type of a value

**Parameters** **value** –

**Returns**

provdbconnector.utils.serializer.**add_namespaces_to_bundle**(*prov_bundle*, *metadata*)
Add all namespaces in the metadata_dict to the provided bundle

**Parameters**

- **prov_bundle** –
- **metadata** –

**Returns** None

provdbconnector.utils.serializer.**create_prov_record**(*bundle*, *prov_type*, *prov_id*, *properties*, *type_map*)

**Parameters**

- **bundle** –
- **prov_type** – valid prov type like prov:Entry as string
- **prov_id** – valid id as string like <namespace>:<name>
- **properties** – dict{attr_name:attr_value} dict with all properties (prov and additional)
- **type_map** – dict{attr_name:type_str} Contains the type information for each property (only if type is necessary)

**Returns** ProvRecord

provdbconnector.utils.serializer.**decode_json_representation**(*value*, *type*, *bundle*)
Return the value based on the type see also encode_json_representation

**Parameters**

- **value** –
- **type** –
- **bundle** –

**Returns**

provdbconnector.utils.serializer.**split_into_formal_and_other_attributes**(*attributes*, *metadata*)

This function split the attributes and metadata into formal attributes and other attributes. Helpful for merge operations and searching for duplicate relations

**Parameters**

- **attributes** –
- **metadata** –

**Returns** namedtuple(formal_attributes, other_attributes)

**Return type** FormalAndOtherAttributes

provdbconnector.utils.serializer.**merge_record**(*attributes*, *metadata*, *other_attributes*, *other_metadata*)

Merge 2 records into one

**Parameters**

- **attributes** – The original attributes
- **metadata** – The original metadata
- **other_attributes** – The attributes to merge
- **other_metadata** – The metadata to merge

**Returns** tuple(attributes, metadata)

**Return type** Tuple(attributes,metadata)

provdbconnector.utils.serializer.**serialize_namespace**(*namespace: prov.identifier.Namespace*)

#### provdbconnector.utils.validator module

**class** provdbconnector.utils.validator.**Validator**

Bases: object

Class to do some validation, not implemented yet

#### Module contents

## 5.2 Submodules

## 5.3 provdbconnector.prov_db module

**class** provdbconnector.prov_db.**ProvDb**(*api_id=None*, *adapter=None*, *auth_info=None*, **args*)

Bases: object

The public api class. This class provide methods to save and get documents or part of ProvDocuments

**save_document_from_json**(*content=None*)

Saves a new document in the database

**Parameters** **content** (*str or buffer*) – The content

**Returns** document_id

**Return type** str or buffer

**get_document_as_json**(*document_id=None*)
Get a ProvDocument from the database based on the document_id

**Parameters** **document_id** (*str*) – document id

**Returns** ProvDocument as json string

**Return type** str

**save_document_from_xml**(*content=None*)
Saves a prov document in the database based on the xml file

**Parameters** **content** (*str or buffer*) – The content

**Returns** document_id

**Return type** str

**get_document_as_xml**(*document_id=None*)
Get a ProvDocument from the database based on the document_id

**Parameters** **document_id** (*str*) – The id

**Returns** ProvDocument as XML string

**Return type** str

**save_document_from_provn**(*content=None*)
Saves a prov document in the database based on the provn string or buffer

**Parameters** **content** (*str or buffer*) – provn object

**Returns** Document_id

**Return type** str

**get_document_as_provn**(*document_id=None*)
Get a ProvDocument from the database based on the document_id

**Parameters** **document_id** (*str*) – The id

**Returns** ProvDocument

**Return type** ProvDocument

**save_document_from_prov**(*content=None*)
Saves a prov document in the database based on the prov document

**Parameters** **content** (*ProvDocument*) – Prov document

**Returns** document_id

**Return type** str

**save_document**(*content=None*)
The main method to Save a document in the db

**Parameters** **content** (*str or buffer or ProvDocument*) – The content can be a xml, json or provn string or buffer or a ProvDocument instance

**Returns** Document id

**Return type** str

**get_document_as_prov**(*document_id=None*)
Get a ProvDocument from the database based on the document id

**Parameters** **document_id** (*str*) – The id

**Returns** Prov Document

**Return type** ProvDocument

**save_element**(*prov_element*, *bundle_id=None*)

Saves a activity, entity, agent

```
doc = ProvDocument()

agent       = doc.agent("ex:yourAgent")
activity    = doc.activity("ex:yourActivity")
entity      = doc.entity("ex:yourEntity")

# Save the elements
agent_id = prov_db.save_element(agent)
activity_id = prov_db.save_element(activity)
entity_id = prov_db.save_element(entity)
```

**Parameters**

- **prov_element** (*prov.model.ProvElement*) – The ProvElement
- **bundle_id** (*str*) –

**Returns** Identifier of the element

**Return type** prov.model.QualifiedName

**get_elements**(*prov_element_cls*)

Return a document that contains the requested type

```
from prov.model import ProvEntity, ProvAgent, ProvActivity

document_with_all_entities = prov_db.get_elements(ProvEntity)
document_with_all_agents = prov_db.get_elements(ProvAgent)
document_with_all_activities = prov_db.get_elements(ProvActivity)

print(document_with_all_entities)
print(document_with_all_agents)
print(document_with_all_activities)
```

**Parameters** **prov_element_cls** –

**Returns** Prov document

:rtype prov.model.ProvDocument

**get_element**(*identifier*)

Get a element (activity, agent, entity) from the database

```
doc = ProvDocument()

identifier = QualifiedName(doc, "ex:yourAgent")

prov_element = prov_db.get_element(identifier)
```

**Parameters** **identifier** (*prov.model.QualifiedName*) –

**Returns** A prov Element class

**save_record**(*prov_record*, *bundle_id=None*)
Saves a realtion or a element (Entity, Agent or Activity)

```
doc = ProvDocument()

agent       = doc.agent("ex:Alice")
ass_rel     = doc.association("ex:Alice", "ex:Bob")

# Save the elements
agent_id = prov_db.save_record(agent)
relation_id = prov_db.save_record(ass_rel)
```

**Parameters** **prov_record** – The prov record

:type prov.model.ProvRecord :param bundle_id: The bundle id that you got back if you created a bundle or document :type str :return:

**get_bundle**(*identifier*)
Returns the whole bundle for the provided identifier

**Parameters** **identifier** (*prov.model.QualifiedName*) – The identifier

**Returns** The prov bundle instance

:rtype prov.model.ProvBundle

**save_bundle**(*prov_bundle*)
Public method to save a bundle

```
doc = ProvDocument()

bundle = doc.bundle("ex:bundle1")
# Save the bundle
prov_db.save_bundle(bundle)
```

**Parameters** **prov_bundle** (*prov.model.ProvBundle*) –

**Returns**

**save_relation**(*prov_relation*, *bundle_id=None*)
Saves a relation between 2 nodes that are already in the database.

```
doc = ProvDocument()

activity    = doc.activity("ex:yourActivity")
entity      = doc.entity("ex:yourEntity")
wasGeneratedBy = entity.wasGeneratedBy("ex:yourAgent")

# Save the elements
rel_id = prov_db.save_relation(wasGeneratedBy)
```

**Parameters** **prov_relation** (*ProvRelation*) – The ProvRelation instance

:param bundle_id :type bundle_id: str :return: Relation id :rtype: str

## 5.4 Module contents

CHAPTER 6

# provdbconnector modules

CHAPTER 7

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## p

# Index

## D



## E



## F



## G

## I



## J



## L



## M



## N



## P

## R



## S

## T

## V