Contact Book Documentation

Release 0.0.1

Ninad Page

May 31, 2017

Contents

Contents:

contactbook.**init_contactbook**(*, *sqlite_db_path=None*, *db_connection_string=None*, *logger=None*)

Initializes Contact Book library (database connections, logging, etc). All parameters must be names explicitly. Only one of sqlite_db_path and db_connection_string must be provided.

If the sqlite database given by the path doesn't exist, it is created. The path can be relative or absolute, in which case it must start with a /.

If you want to use any other database engine, you can specify appropriate db_connection_string. Do not use an existing database, you might lose existing tables!

If logger is not provided, it creates a logger which emits to stdout at DEBUG level.

Parameters

- **sqlite_db_path** (*str*) – Path to sqlite database file
- **db_connection_string** (*str*) – A SQLAlchemy Database URL (see http://docs.sqlalchemy.org/en/latest/core/engines.html#database-urls)
- **logger** (*logging.Logger*) – Logger object which will be used by this package for all logging

Returns None

class contactbook.db.**ContactBookDB**

add_address(*person_id*, *house_number=None*, *street_name=None*, *address_line_1=None*, *address_line_2=None*, *city=None*, *postal_code=None*, *country=None*, *address_label=None*)

Adds a new address to person specified by person_id.

Parameters

- **person_id** (*int*) – id of the person
- **house_number** (*str*) – House number
- **street_name** (*str*) – Street name
- **address_line_1** (*str*) – Address line 1
- **address_line_2** (*str*) – Address line 2
- **city** (*str*) – City
- **postal_code** (*str*) – Postal code
- **country** (*str*) – Country
- **address_label** (*str*) – Label for address

Returns Address created

Return type models.Address

add_email_address(*person_id*, *email_address*, *email_label=None*)

Adds a new email address to person specified by person_id.

Parameters

- **person_id** (*int*) – id of the person
- **email_address** (*str*) – Email address

- **email_label** (*str*) – Label for email address

Returns Email address created

Return type models.EmailAddress

add_group_to_person(*person_id*, *group_id*)
Adds group specified by group_id to person specified by person_id.

Parameters

- **person_id** (*int*) – id of the person
- **group_id** (*int*) – id of the group

Returns None

add_phone_number(*person_id*, *phone_number*, *phone_label=None*)
Adds a new phone number to person specified by person_id.

Parameters

- **person_id** (*int*) – id of the person
- **phone_number** (*str*) – Phone number
- **phone_label** (*str*) – Label for phone number

Returns Phone number created

Return type models.PhoneNumber

create_group(*name*)
Adds a new group.

Parameters **name** (*str*) – Name of the group

Returns Group created

Return type models.Group

create_person(*title=None*, *first_name=None*, *middle_name=None*, *last_name=None*, ***, *suffix=None*, *phone_number=None*, *phone_label=None*, *email_address=None*, *email_label=None*, *group_id=None*)
Adds a person and optionally, associated details (phone number, email, group). The group must exist. Street address can be later added manually. Addition details such as other phone numbers or other groups can also be later added manually.

Parameters

- **title** (*str*) – Title
- **first_name** (*str*) – First name
- **middle_name** (*str*) – Middle name
- **last_name** (*str*) – Last name
- **suffix** (*str*) – Suffix
- **phone_number** (*str*) – Phone number
- **phone_label** (*str*) – Label for phone number
- **email_address** (*str*) – Email address
- **email_label** (*str*) – Label for email address
- **group_id** (*int*) – id of the group this person belongs to

Returns Person created

Return type models.Person

delete_address(*address_id*)
Deletes an address.

Parameters **address_id** (*int*) – id of the address to delete

Returns None

delete_email_address(*email_address_id*)
Deletes an email address.

Parameters **email_address_id** (*int*) – id of the email address to delete

Returns None

delete_group(*group_id*)
Deletes a group. The group is also removed from the persons who were part of this group.

Parameters **group_id** (*int*) – id of the group to delete

Returns None

delete_person(*person_id*)
Deletes a person and all associated fields (phone numbers, email addresses, etc). The person is also removed from the groups he/she was part of.

Parameters **person_id** (*int*) – id of the person to delete

Returns None

delete_phone_number(*phone_number_id*)
Deletes a phone number.

Parameters **phone_number_id** (*int*) – id of the phone number to delete

Returns None

classmethod **find_person_details_by_name**(*name*)
Returns short details of a person (person id and full name) given a name. This name can be of any of person's attributes (i.e., first_name, last_name, etc) and can even be a prefix. Lookup is case-insensitive.

If name is space separated, each word is searched individually, and results matching _both_ are returned. This can be used to search for persons with matching first name & last name (both).

This lookup is extremely fast as it is performed on an in-memory trie which caches these details. So this method can be used to implement Auto-Complete for a Contacts app, where you call this every time a character is entered by the user, show the user list of matching full names which is updated instantly, and when the user picks one, use the id to fetch full details of the person.

Example

Assume you have persons with following persons (id: full name):

```
>>> 1: Abcd Hijk
>>> 2: Cdef Abc
>>> 3: Abef Hijk
```

Then

```
>>> find_person_details_by_name('ab')
```

will return `[(1, Abcd Hijk), (2, Abef Abc), (3, Abef Hijk)]`

```
>>> find_person_details_by_name('abc')
```

will return `[(1, Abcd Hijk), (2, Abef Abc)]`

```
>>> find_person_details_by_name('abcd')
```

will return `[(1, Abcd Hijk)]`

```
>>> find_person_details_by_name('ab hi')
```

will return `[(1, Abcd Hijk), (3, Abef Hijk)]`

```
>>> find_person_details_by_name('abe hi')
```

will return `[(3, Abef Hijk)]`

Parameters **name** (*str*) – Prefix of any name attribute to lookup

Returns List of short persons' details (which are namedtuples with fields id and full_name)

Return type <list (fast_lookup.FastLookupValue)>

get_all_persons(*group_id=None*)

Returns all persons. Optionally filters based on given group_id.

Parameters **group_id** (*int*) – Group id to filter persons on

Returns List of persons

Return type <list (models.Person)>

get_group_by_id(*group_id*)

Returns a Group object with given id.

Parameters **group_id** (*int*) – id of the group

Returns Group object

Return type models.Group

get_person_by_id(*person_id*)

Returns a Person object with given id.

Parameters **person_id** (*int*) – id of the person

Returns Person object

Return type models.Person

get_persons_by_email(*email*)

Returns all persons with matching email address. *email* can be prefix of the address. e.g., for a person with email abc@example.com, it will return that person for both email='abc@example.com' and email='abc'.

Since we use SQL LIKE query to find matching persons, it's fairly straightforward to extend it to find match anywhere (instead of just as prefix). i.e., `SELECT when email LIKE '%<substr>%';`

However, such queries (patterns beginning with wildcards) need to do full-table scan as indices won't help for that kind of queries. It can be sped up by using FULLTEXT index in MySQL or using CONTAINS instead of LIKE in SQL Server (which again uses a full-text index). However, expect the performance gain to be limited as full- text indices typically split words.

But in a contact book, looking up by email isn't typically expected to be an instant operation like looking up by name or phone number is, so any of the above solutions would do just fine.

Parameters **email** (*str*) – Email address to filter persons on (can be a prexix)

Returns List of persons

Return type <list (models.Person)>

save_object(*obj*)
Writes in-memory changes done to an SQLAlchemy object to database.

Parameters **obj** (*sqlalchemy.ext.declarative.api.Base*) – Any Contact Book object (Person, Address, PhoneNumber, EmailAddress, Group)

Returns Updated object

Return type sqlalchemy.ext.declarative.api.Base

exception contactbook.exceptions.**NoSuchObjectFound**(*object_type*, *object_id*, **args*, ***kwargs*)
Exception class which is raised if object with given id is not found in the database.

CHAPTER 1

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index