tahweela

Generated by Doxygen 1.8.17

Chapter 1

README

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 client Namespace Reference

Namespaces

- client

6.2 client.client Namespace Reference

Data Structures

- class Client

Functions

- def get_bank_name ()
- def get_branch_number ()
- def get_account_number ()
- def get_name_reference ()
- def get_name ()
- def get_email ()
- def get_balance ()
- def get_credid ()
- def get_rand_pass (L=9)
- def rand_alphanum (L=9)

Variables

- string db_configs = "dbname='demo_client' user='tahweela_client' password='tahweela'"
- filename
- format
- filemode
- logger = logging.getLogger()
- seed = int.from_bytes(os.urandom(2), 'big')
- faker = Faker(seed)
- parser = ArgumentParser()
- type
- args = parser.parse_args()
- cred_id = args.add_contact
- trader1 = Client()
- trader2 = Client()

6.2.1 Function Documentation

6.2.1.1 get_account_number()

```
def client.client.get_account_number ( )
```

Definition at line 46 of file client.py.

```
46 def get_account_number():
47     return int(3333333*random.random())
```

6.2.1.2 get_balance()

```
def client.client.get_balance ( )
```

Definition at line 54 of file client.py.

```
54 def get_balance():
55     return 333*random.random()
```

6.2.1.3 get_bank_name()

```
def client.client.get_bank_name ( )
```

Definition at line 42 of file client.py.

```
42 def get_bank_name():
43     return faker.name().split()[0]
```

6.2.1.4 get_branch_number()

```
def client.client.get_branch_number ( )
```

Definition at line 44 of file client.py.

```
44 def get_branch_number():
45     return int(33*random.random())
```

6.2.1.5 get_credid()

```
def client.client.get_credid ( )
```

Definition at line 56 of file client.py.

```
56 def get_credid():
57     return int(3333333333333*random.random())
```

6.2.1.6 get_email()

```
def client.client.get_email ( )
```

Definition at line 52 of file client.py.

```
52 def get_email():
53     return faker.email()
```

6.2.1.7 get_name()

```
def client.client.get_name ( )
```

Definition at line 50 of file client.py.

```
50 def get_name():
51     return faker.name()
```

6.2.1.8 get_name_reference()

```
def client.client.get_name_reference ( )
```

Definition at line 48 of file client.py.

```
48 def get_name_reference():
49     return faker.name().split()[1]
```

6.2.1.9 get_rand_pass()

```
def client.client.get_rand_pass (
               L = 9 )
```

Definition at line 58 of file client.py.

```
58 def get_rand_pass(L=9):
59     passcode=''.join(random.choice(string.ascii_uppercase+\
60                                    string.ascii_lowercase+\
61                                    string.digits)\
62                      for _ in range(L))
63     return passcode
```

6.2.1.10 rand_alphanum()

```
def client.client.rand_alphanum (
               L = 9 )
```

Definition at line 64 of file client.py.

```
64 def rand_alphanum(L=9):
65     passcode=''.join(random.choice(string.ascii_uppercase+\
66                                    string.ascii_lowercase+\
67                                    string.digits)\
68                      for _ in range(L))
69     return passcode
70
```

6.2.2 Variable Documentation

6.2.2.1 args

```
client.client.args = parser.parse_args()
```

Definition at line 339 of file client.py.

6.2.2.2 cred_id

```
client.client.cred_id = args.add_contact
```

Definition at line 340 of file client.py.

6.2.2.3 db_configs

```
string client.client.db_configs = "dbname='demo_client' user='tahweela_client' password='tahweela'"
```

Definition at line 21 of file client.py.

6.2.2.4 faker

```
client.client.faker = Faker(seed)
```

Definition at line 40 of file client.py.

6.2.2.5 filemode

```
client.client.filemode
```

Definition at line 25 of file client.py.

6.2.2.6 filename

client.client.filename

Definition at line 23 of file client.py.

6.2.2.7 format

client.client.format

Definition at line 24 of file client.py.

Referenced by queries.database.exists.account_byemail(), core.queries.database.exists.account_byemail(), queries.database.exists.account_byname(), core.queries.database.exists.account_byname(), queries.database.inserts.add_bank_account(), core.queries.database.inserts.add_bank_account(), queries.database.inserts.add_client(), core.queries.database.inserts.add_client(), core.queries.database.inserts.add_currency(), queries.inserts.goods.add_good(), core.queries.inserts.goods.add_good(), queries.inserts.owners.add_owner(), core.queries.inserts.owners.add_owner(), client.client.Client.autotract(), queries.database.exists.bank_account_bycid(), core.queries.database.exists.bank_account_bycid(), queries.database.gets.cid2credid(), core.queries.database.gets.cid2credid(), queries.database.exists.client_exists(), core.queries.database.exists.client_exists(), queries.database.exists.contact_exists(), core.queries.database.exists.contact_exists(), queries.database.exists.credential_exists(), core.queries.database.exists.credential_exists(), queries.database.gets.credid2cid(), core.queries.database.gets.credid2cid(), core.queries.database.exists.currency(), core.queries.database.updates.currency_preference(), queries.exists.banking.exists(), core.queries.exists.banking.exists(), queries.exists.clients.exists(), core.queries.exists.clients.exists(), queries.exists.contacts.exists(), core.queries.exists.contacts.exists(), queries.exists.credentials.exists(), core.queries.exists.credentials.exists(), queries.exists.goods.exists(), core.queries.exists.goods.exists(), queries.gets.clients.get(), core.queries.gets.clients.get(), queries.database.gets.get(), core.queries.database.gets.get(), queries.gets.banking.get_balance_by_cid(), core.queries.gets.banking.get_balance_by_cid(), queries.database.gets.get_balance_by_cid(), core.queries.database.gets.get_balance_by_cid(), queries.gets.banking.get_balance_by_credid(), core.queries.gets.banking.get_balance_by_credid(), queries.database.gets.get_balance_by_credid(), core.queries.database.gets.get_balance_by_credid(), queries.gets.contacts.get_banking_id(), core.queries.gets.contacts.get_banking_id(), queries.gets.banking.get_banking_id(), core.queries.gets.banking.get_banking_id(), queries.database.gets.get_banking_id(), core.queries.database.gets.get_banking_id(), queries.gets.banking.get_client_id(), core.queries.gets.banking.get_client_id(), queries.database.gets.get_client_id(), core.queries.database.gets.get_client_id(), queries.database.gets.get_client_id_byemail(), core.queries.database.gets.get_client_id_byemail(), queries.database.gets.get_client_id_byname(), core.queries.database.gets.get_client_id_byname(), queries.gets.goods.get_commodity(), core.queries.gets.goods.get_commodity(), core.queries.gets.credentials.get_credential(), queries.gets.credentials.get_credential(), queries.database.gets.get_credential(), core.queries.database.gets.get_credential(), queries.gets.credentials.get_credid_with_gid(), core.queries.gets.credentials.get_credid_with_gid(), core.queries.database.gets.get_currency_id(), core.queries.database.gets.get_currency_name(), queries.gets.goods.get_good(), core.queries.gets.goods.get_good(), queries.gets.owner.get_good_owner(), core.queries.gets.owner.get_good_owner(), queries.gets.credentials.get_id(), core.queries.gets.credentials.get_id(), core.queries.gets.owner.get_owner_goods(), queries.gets.owner.get_owner_goods(), core.queries.gets.credentials.get_password(), queries.gets.credentials.get_password(), queries.database.gets.get_password(), core.queries.database.gets.get_password(), core.queries.database.gets.get_preference_currency_bycid(), queries.database.gets.get_sells(), core.queries.database.gets.get_sells(), core.queries.gets.ledger.get_transactions(), queries.gets.ledger.get_transactions(), queries.database.gets.get_transactions(), core.queries.database.gets.get_transactions(), queries.database.gets.get_transactions_sum(), core.queries.database.gets.get_transactions_sum(), queries.inserts.banking.insert_banking(), core.queries.inserts.banking.insert_banking(), queries.inserts.clients.insert_client(), core.queries.inserts.clients.insert_client(), queries.inserts.contacts.insert_contact(), core.queries.inserts.contacts.insert_contact(), queries.database.inserts.insert_contact(), core.queries.database.inserts.insert_contact(), queries.inserts.ledger.insert_trx(), core.queries.inserts.ledger.insert_trx(), queries.database.inserts.insert_trx(), core.queries.database.inserts.insert_trx(), queries.database.database.lock_advisory(), core.queries.database.database.lock_advisory(), core.queries.inserts.credentials.new_cred(), queries.inserts.credentials.new_cred(),

queries.inserts.credentials.register(), core.queries.inserts.credentials.register(), client.client.Client.register(), queries.database.inserts.register(), core.queries.database.inserts.register(), queries.gets.currency.to_dollar(), core.queries.gets.currency.to_dollar(), queries.database.gets.to_dollar(), core.queries.database.gets.to_euro(), queries.database.database.unlock_advisory(), core.queries.database.database.unlock_advisory(), queries.↩ updates.banking.update_account(), core.queries.updates.banking.update_account(), queries.database.updates.↩ update_account(), core.queries.database.updates.update_account(), queries.updates.owners.update_owner(), and core.queries.updates.owners.update_owner().

6.2.2.8 logger

```
client.client.logger = logging.getLogger()
```

Definition at line 26 of file client.py.

6.2.2.9 parser

```
client.client.parser = ArgumentParser()
```

Definition at line 337 of file client.py.

6.2.2.10 seed

```
client.client.seed = int.from_bytes(os.urandom(2), 'big')
```

Definition at line 38 of file client.py.

6.2.2.11 trader1

```
client.client.trader1 = Client()
```

Definition at line 341 of file client.py.

6.2.2.12 trader2

```
client.client.trader2 = Client()
```

Definition at line 342 of file client.py.

6.2.2.13 type

```
client.client.type
```

Definition at line 338 of file client.py.

Referenced by client.client.Client.__update_balance(), core_test.RestfulTest.__update_balance(), and core_test.↩
RestfulTest.test_register().

6.3 core Namespace Reference

Namespaces

- client
- connection_cursor
- queries
- server
- utils

6.4 core.client Namespace Reference

Namespaces

- client

6.5 core.client.client Namespace Reference

Data Structures

- class Client

Functions

- def get_bank_name ()
- def get_branch_number ()
- def get_account_number ()
- def get_name_reference ()
- def get_name ()
- def get_email ()
- def get_balance ()
- def get_credid ()
- def get_rand_pass (L=9)
- def rand_alphanum (L=9)

Variables

- string db_configs = "dbname='demo_client' user='tahweela_client' password='tahweela'"
- filename
- format
- filemode
- logger = logging.getLogger()
- seed = int.from_bytes(os.urandom(2), 'big')
- faker = Faker(seed)
- parser = ArgumentParser()
- type
- args = parser.parse_args()
- cred_id = args.add_contact
- trader1 = Client()
- trader2 = Client()

6.5.1 Function Documentation

6.5.1.1 get_account_number()

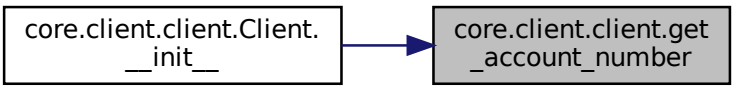
```
def core.client.client.get_account_number ( )
```

Definition at line 46 of file client.py.

```
46 def get_account_number():
47     return int(3333333*random.random())
```

Referenced by core.client.client.Client.__init__().

Here is the caller graph for this function:



6.5.1.2 get_balance()

```
def core.client.client.get_balance ( )
```

Definition at line 54 of file client.py.

```
54 def get_balance():
55     return 333*random.random()
```

6.5.1.3 get_bank_name()

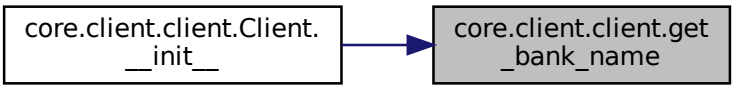
```
def core.client.client.get_bank_name ( )
```

Definition at line 42 of file client.py.

```
42 def get_bank_name():
43     return faker.name().split()[0]
```

Referenced by core.client.client.Client.__init__().

Here is the caller graph for this function:



6.5.1.4 get_branch_number()

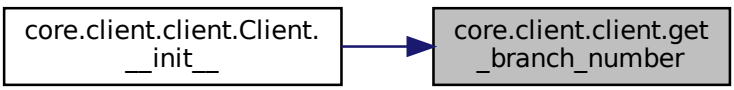
```
def core.client.client.get_branch_number ( )
```

Definition at line 44 of file client.py.

```
44 def get_branch_number():
45     return int(33*random.random())
```

Referenced by core.client.client.Client.__init__().

Here is the caller graph for this function:



6.5.1.5 get_credid()

```
def core.client.client.get_credid ( )
```

Definition at line 56 of file client.py.

```
56 def get_credid():
57     return int(3333333333333*random.random())
```

6.5.1.6 get_email()

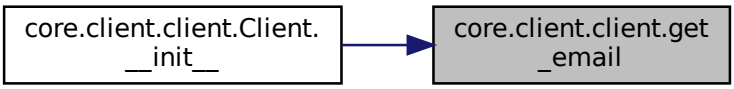
```
def core.client.client.get_email ( )
```

Definition at line 52 of file client.py.

```
52 def get_email():
53     return faker.email()
```

Referenced by core.client.client.Client.__init__().

Here is the caller graph for this function:



6.5.1.7 get_name()

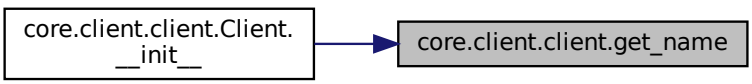
```
def core.client.client.get_name ( )
```

Definition at line 50 of file client.py.

```
50 def get_name():
51     return faker.name()
```

Referenced by core.client.client.Client.__init__().

Here is the caller graph for this function:



6.5.1.8 get_name_reference()

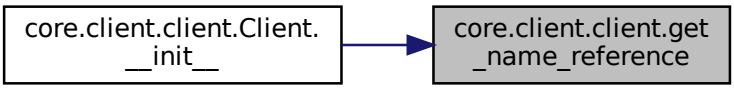
```
def core.client.client.get_name_reference ( )
```

Definition at line 48 of file client.py.

```
48 def get_name_reference():
49     return faker.name().split()[1]
```

Referenced by core.client.client.Client.__init__().

Here is the caller graph for this function:



6.5.1.9 get_rand_pass()

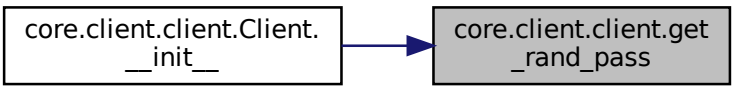
```
def core.client.client.get_rand_pass (
               L = 9 )
```

Definition at line 58 of file client.py.

```
58 def get_rand_pass(L=9):
59     passcode=''.join(random.choice(string.ascii_uppercase+\
60                                    string.ascii_lowercase+\
61                                    string.digits)\
62                      for _ in range(L))
63     return passcode
```

Referenced by core.client.client.Client.__init__().

Here is the caller graph for this function:



6.5.1.10 rand_alphanum()

```
def core.client.client.rand_alphanum (
               L = 9 )
```

Definition at line 64 of file client.py.

```
64 def rand_alphanum(L=9):
65     passcode=''.join(random.choice(string.ascii_uppercase+\
66                                    string.ascii_lowercase+\
67                                    string.digits)\
68                      for _ in range(L))
69     return passcode
70
```

6.5.2 Variable Documentation

6.5.2.1 args

```
core.client.client.args = parser.parse_args()
```

Definition at line 339 of file client.py.

6.5.2.2 cred_id

```
core.client.client.cred_id = args.add_contact
```

Definition at line 340 of file client.py.

6.5.2.3 db_configs

```
string core.client.client.db_configs = "dbname='demo_client' user='tahweela_client' password='tahweela'"
```

Definition at line 21 of file client.py.

6.5.2.4 faker

```
core.client.client.faker = Faker(seed)
```

Definition at line 40 of file client.py.

6.5.2.5 filemode

```
core.client.client.filemode
```

Definition at line 25 of file client.py.

6.5.2.6 filename

```
core.client.client.filename
```

Definition at line 23 of file client.py.

6.5.2.7 format

```
core.client.client.format
```

Definition at line 24 of file client.py.

Referenced by core.client.client.Client.autotract(), and core.client.client.Client.register().

6.5.2.8 logger

```
core.client.client.logger = logging.getLogger()
```

Definition at line 26 of file client.py.

6.5.2.9 parser

```
core.client.client.parser = ArgumentParser()
```

Definition at line 337 of file client.py.

6.5.2.10 seed

```
core.client.client.seed = int.from_bytes(os.urandom(2), 'big')
```

Definition at line 38 of file client.py.

6.5.2.11 trader1

```
core.client.client.trader1 = Client()
```

Definition at line 341 of file client.py.

6.5.2.12 trader2

```
core.client.client.trader2 = Client()
```

Definition at line 342 of file client.py.

6.5.2.13 type

```
core.client.client.type
```

Definition at line 338 of file client.py.

Referenced by core.client.client.Client.__update_balance().

6.6 core.connection_cursor Namespace Reference

6.7 core.queries Namespace Reference

Namespaces

- database
- exists
- gets
- inserts
- updates

6.8 core.queries.database Namespace Reference

Data Structures

- class database
- class exists
- class gets
- class inserts
- class updates

6.9 core.queries.exists Namespace Reference

Namespaces

- banking
- clients
- contacts
- credentials
- goods
- owners

6.10 core.queries.exists.banking Namespace Reference

Functions

- def exists (cid)

6.10.1 Function Documentation

6.10.1.1 exists()

```
def core.queries.exists.banking.exists (
              cid )
```

verify that a banking account with the given client id is available (CALLED AT THE SERVER SIDE)

@param cid: client id
@return boolean wither the banking account for give client exists or note

Definition at line 4 of file banking.py.

```
4 def exists(cid):
5     """verify that a banking account with the given client id is available (CALLED AT THE SERVER SIDE)
6
7     @param cid: client id
8     @return boolean wither the banking account for give client exists or note
9     """
10     stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM banking WHERE client_id={cid});").\
11         format(cid=sql.Literal(cid))
12     cur.execute(stat)
13     return cur.fetchone()[0]
```

References client.client.format.

6.11 core.queries.exists.clients Namespace Reference

Functions

- def exists (cid)

6.11.1 Function Documentation

6.11.1.1 exists()

```
def core.queries.exists.clients.exists (
              cid )
```

verify that a client with given id is available (CALLED AT THE SERVER SIDE)

@param cid: client id
@return boolean wither the client exists or note

Definition at line 4 of file clients.py.

```
4 def exists(cid):
5     """verify that  a client with given id is available (CALLED AT THE SERVER SIDE)
6
7     @param cid: client id
8     @return boolean wither the client exists or note
9     """
10     stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM clients WHERE id={cid});").\
11         format(cid=sql.Literal(cid))
12     cur.execute(stat)
13     return cur.fetchone()[0]
```

References client.client.format.

6.12 core.queries.exists.contacts Namespace Reference

Functions

- def exists (cid)

6.12.1 Function Documentation

6.12.1.1 exists()

```
def core.queries.exists.contacts.exists (
              cid )
```

verify that a contact with given id is available (CALLED AT THE CLIENT SIDE)

@param cid: contact id
@return boolean wither the contact exists or note

Definition at line 4 of file contacts.py.

```
4 def exists(cid):
5     """verify that a contact with given id is available (CALLED AT THE CLIENT SIDE)
6
7     @param cid: contact id
8     @return boolean wither the contact exists or note
9     """
10     stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM contacts WHERE contact_id={cid});").\
11         format(cid=sql.Literal(cid))
12     cur.execute(stat)
13     return cur.fetchone()[0]
```

References client.client.format.

6.13 core.queries.exists.credentials Namespace Reference

Functions

- def exists (cid)

6.13.1 Function Documentation

6.13.1.1 exists()

```
def core.queries.exists.credentials.exists (
              cid )
```

```
verify the credential for the client with given cid(CALLED FROM SERVER SIDE),
or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)

@param cid: client id, or 1 (in case of call from client side for it's own credential)
@return boolean for wither the client (with given cid) is registered or not
```

Definition at line 4 of file credentials.py.

```
4 def exists(cid):
5     """ verify the credential for the client with given cid(CALLED FROM SERVER SIDE),
6     or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)
7
8     @param cid: client id, or 1 (in case of call from client side for it's own credential)
9     @return boolean for wither the client (with given cid) is registered or not
10     """
11     stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM credentials WHERE id={cid})").\
12         format(cid=sql.Literal(cid))
13     cur.execute(stat)
14     return cur.fetchone()[0]
15
16 "'
17 def exists(cred_id, passcode):
18     """ verify the credential for the client with given cid(CALLED FROM SERVER SIDE),
19     or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)
20
21     @param cid: client id, or 1 (in case of call from client side for it's own credential)
22     @return boolean for wither the client (with given cid) is registered or not
23     """
24     stat="SELECT EXISTS (SELECT 1 FROM credentials WHERE cred_id={} AND passcode={})".\
25         format(cred, passcode)
26     cur.execute(cid)
27     return cur.fetchone()[0]
28 "'
```

References client.client.format.

6.14 core.queries.exists.goods Namespace Reference

Functions

- def exists (gid)

6.14.1 Function Documentation

6.14.1.1 exists()

```
def core.queries.exists.goods.exists (
              gid )
```

verify that a good with given id is available

@param gid: good id
@return boolean wither the good exists or note

Definition at line 4 of file goods.py.

```
4 def exists(gid):
5     """verify that a good with given id is available
6
7     @param gid: good id
8     @return boolean wither the good exists or note
9     """
10     stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM goods WHERE id={gid});").format(gid=sql.Literal(gid))
11     cur.execute(stat)
12     return cur.fetchone()[0]
```

References client.client.format.

6.15 core.queries.exists.owners Namespace Reference

Functions

- def exists ()

6.15.1 Function Documentation

6.15.1.1 exists()

```
def core.queries.exists.owners.exists ( )
```

Definition at line 3 of file owners.py.

```
3 def exists():
4     pass
```

6.16 core.queries.gets Namespace Reference

Namespaces

- banking
- clients
- contacts
- credentials
- currency
- goods
- ledger
- owner

6.17 core.queries.gets.banking Namespace Reference

Functions

- def get_client_id (bid)
- def get_banking_id (cid)
- def get_balance_by_cid (cid)
- def get_balance_by_credid (cred_id)

6.17.1 Function Documentation

6.17.1.1 get_balance_by_cid()

```
def core.queries.gets.banking.get_balance_by_cid (
              cid )
```

```
called at the server side to retrieve the account balance d of the given client_id (cid)

@param cid: client id
@return bid: banking id
```

Definition at line 26 of file banking.py.

```
26 def get_balance_by_cid(cid):
27     """called at the server side to retrieve the account balance d of the given client_id (cid)
28
29     @param cid: client id
30     @return bid: banking id
31     """
32     query=sql.SQL("SELECT (balance) FROM banking WHERE client_id={cid} LIMIT
       1").format(cid=sql.Literal(cid))
33     db_log.debug(query)
34     return pd.read_sql(query, conn).ix[0]
35
```

References client.client.format.

6.17.1.2 get_balance_by_credid()

```
def core.queries.gets.banking.get_balance_by_credid (
              cred_id )

get balance of client with given credential id

@param cred_id: client credential id
```

Definition at line 36 of file banking.py.

```
36 def get_balance_by_credid(cred_id):
37     """ get balance of client with given credential id
38
39     @param cred_id: client credential id
40     """
41     query=sql.SQL("SELECT (b.balance) FROM banking as b JOIN WITH credentials AS c WHERE
       c.cred_id={credid} AND c.id==b.client_id;").format(credid=sql.Literal(cred_id))
42     db_log.debug(query)
43     cur.execute(query)
44     return cur.fetchone()[0]
```

References client.client.format.

6.17.1.3 get_banking_id()

```
def core.queries.gets.banking.get_banking_id (
              cid )

retrieve the corresponding banking_id of the given  client_id (cid) (called at the server side)

@param cid: client id
@return bid: banking id
```

Definition at line 16 of file banking.py.

```
16 def get_banking_id(cid):
17     """retrieve the corresponding banking_id of the given  client_id (cid) (called at the server side)
18
19     @param cid: client id
20     @return bid: banking id
21     """
22     query=sql.SQL("SELECT (id) FROM banking WHERE client_id={cid} LIMIT 1").format(cid=sql.Literal(cid))
23     db_log.debug(query)
24     return pd.read_sql(query, conn).ix[0]
25
```

References client.client.format.

6.17.1.4 get_client_id()

```
def core.queries.gets.banking.get_client_id (
              bid )

retrieve the corresponding client_id of the given banking_id (bid) (called at the server side)

@param bid: banking id
@return cid: contact id
```

Definition at line 6 of file banking.py.

```
6 def get_client_id(bid):
7     """ retrieve the corresponding client_id of the given banking_id (bid) (called at the server side)
8
9     @param bid: banking id
10     @return cid: contact id
11     """
12     query=sql.SQL("SELECT (client_id) FROM banking WHERE id={bid} LIMIT 1").format(bid=sql.Literal(bid))
13     db_log.debug(query)
14     return pd.read_sql(query, conn).ix[0]
15
```

References client.client.format.

6.18 core.queries.gets.clients Namespace Reference

Functions

- def get_all ()
- def get (cid)
- def get_name (cid)

6.18.1 Function Documentation

6.18.1.1 get()

```
def core.queries.gets.clients.get (
              cid )
```

```
retrieve client into with given client id (cid)

@param cid: client id
@return tuple (id, name, join date)
```

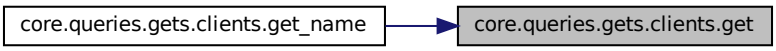
Definition at line 15 of file clients.py.

```
15 def get(cid):
16     """retrieve client into with given client id (cid)
17
18     @param cid: client id
19     @return tuple (id, name, join date)
20     """
21     query=sql.SQL("SELECT (id, contact_name, client_join_dt) FROM clients WHERE
       id={cid};").format(cid=sql.Literal(cid))
22     db_log.debug(query)
23     cur.execute(query)
24     return cur.fetchone()
25
```

References client.client.format.

Referenced by core.queries.gets.clients.get_name().

Here is the caller graph for this function:



6.18.1.2 get_all()

```
def core.queries.gets.clients.get_all ( )
```

retrieve all clients info

Definition at line 7 of file clients.py.

```
7 def get_all():
8     """retrieve all clients info
9 
10     """
11     query="SELECT * FROM clients;"
12     db_log.debug(query)
13     return pd.read_sql(query, conn)
14 
```

6.18.1.3 get_name()

```
def core.queries.gets.clients.get_name (
              cid )
```

retrieve client name corresponding to given client id (cid)

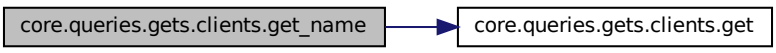
@param cid: client id
@return client name

Definition at line 26 of file clients.py.

```
26 def get_name(cid):
27     """retrieve client name corresponding to given client id (cid)
28 
29     @param cid: client id
30     @return client name
31     """
32     return get(cid)[1]
```

References core.queries.gets.clients.get().

Here is the call graph for this function:



6.19 core.queries.gets.contacts Namespace Reference

Functions

- def get_all ()
- def get_banking_id (cid)

6.19.1 Function Documentation

6.19.1.1 get_all()

```
def core.queries.gets.contacts.get_all ( )
```

Definition at line 6 of file contacts.py.

```
6 def get_all():
7     query = "SELECT * FROM contacts;"
8     db_log.debug(query)
9     return pd.read_sql(query, conn)
10
```

6.19.1.2 get_banking_id()

```
def core.queries.gets.contacts.get_banking_id (
              cid )
```

```
called at the client side, to retrieve the stored banking id in the contacts

@param cid: contact id
@return banking_id or the associated banking id for the given contact id
```

Definition at line 11 of file contacts.py.

```
11 def get_banking_id(cid):
12     """ called at the client side, to retrieve the stored banking id in the contacts
13
14     @param cid: contact id
15     @return banking_id or the associated banking id for the given contact id
16     """
17     query=sql.SQL("SELECT (bank_account_id) FROM contacts WHERE contact_id='{cid}' LIMIT 1;").\
18         format(cid=sql.Literal(cid))
19     db_log.debug(query)
20     return pd.read_sql(query, conn).ix[0]
```

References client.client.format.

6.20 core.queries.gets.credentials Namespace Reference

Functions

- def get_all ()
- def get_credential (cid)
- def get_id (cred_id)
- def get_password (cred_id)
- def get_credid_with_gid (gid)

6.20.1 Function Documentation

6.20.1.1 get_all()

```
def core.queries.gets.credentials.get_all ( )
```

Definition at line 6 of file credentials.py.

```
6 def get_all():
7     query = "SELECT * FROM credentials;"
8     db_log.debug(query)
9     ret = pd.read_sql(conn, query)
10     return ret
11
```

6.20.1.2 get_credential()

```
def core.queries.gets.credentials.get_credential (
              cid )
```

```
get the credential for the client with given cid(CALLED FROM SERVER SIDE),
or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)

@param cid: client id, or 1 (in case of call from client side for it's own credential)
```

Definition at line 12 of file credentials.py.

```
12 def get_credential(cid):
13     """ get the credential for the client with given cid(CALLED FROM SERVER SIDE),
14     or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)
15
16     @param cid: client id, or 1 (in case of call from client side for it's own credential)
17     """
18     query=sql.SQL("SELECT * FROM credentials WHERE id={cid} LIMIT 1;").\
19         format(cid=sql.Literal(cid))
20     db_log.debug(query)
21     ret = pd.read_sql(conn, query)
22
```

References client.client.format.

6.20.1.3 get_credid_with_gid()

```
def core.queries.gets.credentials.get_credid_with_gid (
              gid )
```

```
cross reference credential id, with good's id

@param gid: good's id
@return credential id credid
```

Definition at line 47 of file credentials.py.

```
47 def get_credid_with_gid(gid):
48     """cross reference credential id, with good's id
49
50     @param gid: good's id
51     @return credential id credid
52     """
53     query=sql.SQL("SELECT (C.cred_id) FROM credentials as c JOIN WITH goods AS g JOIN WITH owners as o
       WHERE g.id={gid} AND o.owner_id==c.id LIMIT 1;").\
54         format(gid=sql.Literal(gid))
55     db_log.debug(query)
56     cur.execute(query)
57     return cur.fetchone()[0]
```

References client.client.format.

6.20.1.4 get_id()

```
def core.queries.gets.credentials.get_id (
              cred_id )
```

```
get client id

@param cred_id: credential id
@return the id, or None if doesn't exist
```

Definition at line 23 of file credentials.py.

```
23 def get_id(cred_id):
24     """ get client id
25
26     @param cred_id: credential id
27     @return the id, or None if doesn't exist
28     """
29     query=sql.SQL("SELECT id FROM credentials WHERE cred_id={credid} LIMIT 1;").\
30         format(credid=sql.Literal(cred_id))
31     db_log.debug(query)
32     cur.execute(query)
33     return cur.fetchone()[0]
34
```

References client.client.format.

6.20.1.5 get_password()

```
def core.queries.gets.credentials.get_password (
              cred_id )
```

```
get user's passcode for authentication

@param cred_id: credential id
@return list of the id, or empty list of doesn't exist
```

Definition at line 35 of file credentials.py.

```
35 def get_password(cred_id):
36     """ get user's passcode for authentication
37
38     @param cred_id: credential id
39     @return list of the id, or empty list of doesn't exist
40     """
41     query=sql.SQL("SELECT (passcode) FROM credentials WHERE cred_id={credid} LIMIT 1;").\
42         format(credid=sql.Literal(cred_id))
43     db_log.debug(query)
44     cur.execute(query)
45     return cur.fetchone()[0]
46
```

References client.client.format.

6.21 core.queries.gets.currency Namespace Reference

Functions

- def to_dollar (cid)

6.21.1 Function Documentation

6.21.1.1 to_dollar()

```
def core.queries.gets.currency.to_dollar (
              cid )
```

```
convert currency of the corresponding id to dollar ratio

for example if currency A = 2 dollars, then the conversion would be 0.5,
for another currency B = 0.5 dollar, then the conversion to dollar would be 2
such that for given cost of xA, would be 0.5x$.
@param cid is the id of the corresponding currency
@return transformation ratio to dollar
```

Definition at line 6 of file currency.py.

```
6 def to_dollar(cid):
7     """ convert currency of the corresponding id to dollar ratio
8
9     for example if currency A = 2 dollars, then the conversion would be 0.5,
10     for another currency B = 0.5 dollar, then the conversion to dollar would be 2
11     such that for given cost of xA, would be 0.5x$.
12     @param cid is the id of the corresponding currency
13     @return transformation ratio to dollar
14     """
15     query = sql.SQL("SELECT * FROM currency WHERE id=cid;").\
16         format(cid=sql.Literal(cid))
17     db_log.debug(query)
18     ratio = 1.0/pd.read_sql(query, conn)['currency_value'].ix[0]
19     return ratio
```

References client.client.format.

6.22 core.queries.gets.goods Namespace Reference

Functions

- def get_all ()
- def get_good (gid)
- def get_commodity (gname, quality=0)
- def get_new_price (gid)

6.22.1 Function Documentation

6.22.1.1 get_all()

```
def core.queries.gets.goods.get_all ( )
```

Definition at line 6 of file goods.py.

```
6 def get_all():
7     query="SELECT * FROM goods;"
8     #return pd.read_sql(query, conn, index_col='id').to_json()
9     return pd.read_sql(query, conn).to_json()
10
```

6.22.1.2 get_commodity()

```
def core.queries.gets.goods.get_commodity (
              gname,
              quality = 0 )

retrive good for the given goods constraints

@param gname: goods name
@param quality: retrieve goods with quality > given threshold
@return pandas data frame of the corresponding constrains
```

Definition at line 22 of file goods.py.

```
22 def get_commodity(gname, quality=0):
23     """retrive good for the given goods constraints
24
25     @param gname: goods name
26     @param quality: retrieve goods with quality > given threshold
27     @return pandas data frame of the corresponding constrains
28     """
29     query = sql.SQL("SELECT * FROM goods WHERE good_name={gname} AND good_quality>={gquality}").\
30         format(gname=sql.Literal(gname), \
31                quality=sql.Literal(gquality))
32     db_log.debug(query)
33     return pd.read_sql(query, conn)
34
```

References client.client.format.

6.22.1.3 get_good()

```
def core.queries.gets.goods.get_good (
              gid )

retrive good for the given goods id (gid)

@param gid: goods id
@return pandas data series of the corresponding row
```

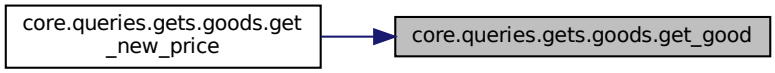
Definition at line 11 of file goods.py.

```
11 def get_good(gid):
12     """retrive good for the given goods id (gid)
13
14     @param gid: goods id
15     @return pandas data series of the corresponding row
16     """
17     query = sql.SQL("SELECT * FROM goods WHERE id={gid};").\
18         format(gid=sql.Literal(gid))
19     db_log.debug(query)
20     return pd.read_sql(query, conn)
21
```

References client.client.format.

Referenced by core.queries.gets.goods.get_new_price().

Here is the caller graph for this function:



6.22.1.4 get_new_price()

```
def core.queries.gets.goods.get_new_price (
              gid )
```

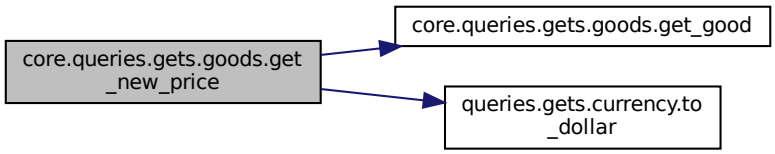
```
get good price with given good's id

@param gid: good's id
@return price in dollar
```

Definition at line 35 of file goods.py.

```
35 def get_new_price(gid):
36     """ get good price with given good's id
37
38     @param gid: good's id
39     @return price in dollar
40     """
41     df = get_good(gid)
42     cur_id = df['good_currency_id'].ix[0]
43     return df['good_cost'].ix[0]*to_dollar(cur_id)
```

References core.queries.gets.goods.get_good(), and queries.gets.currency.to_dollar().

Here is the call graph for this function:



6.23 core.queries.gets.ledger Namespace Reference

Functions

- def get_transactions (st_dt, end_dt=dt.datetime.now())
- def get_sells (dest, st_dt, end_dt=None)
- def get_last_timestamp ()

6.23.1 Function Documentation

6.23.1.1 get_last_timestamp()

```
def core.queries.gets.ledger.get_last_timestamp ( )
```

retrieve the timestamp of the last transaction (CALLED FROM THE CLIENT SIDE)

@return timestamp

Definition at line 29 of file ledger.py.

```
29 def get_last_timestamp():
30     """ retrieve the timestamp of the last transaction (CALLED FROM THE CLIENT SIDE)
31
32     @return timestamp
33     """
34     query="SELECT currval(pg_get_serial_sequence('ledger', 'trx_id'));"
35     db_log.debug(query)
36     cur.execute(query)
37     return cur.fetchone()[0]
```

6.23.1.2 get_sells()

```
def core.queries.gets.ledger.get_sells (
              dest,
              st_dt,
              end_dt = None )
```

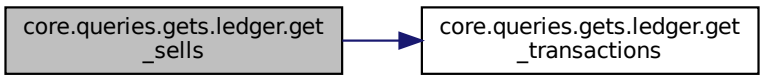
get sells transaction within the st_dt, end_dt period, while there destined to dest (CALLED AT SERVER SIDE)
@param dest: the destination credential id
@return sells transactions

Definition at line 20 of file ledger.py.

```
20 def get_sells(dest, st_dt, end_dt=None):
21     """ get sells transaction within the st_dt, end_dt period, while there destined to dest (CALLED AT
       SERVER SIDE)
22     @param dest: the destination credential id
23     @return sells transactions
24 """
25     trx=get_transactions(st_dt, end_dt).to_json()
26     trx.apply(lambda x:x['trx_dest']==dest, inplace=True)
27     return trx
28
```

References core.queries.gets.ledger.get_transactions().

Here is the call graph for this function:



6.23.1.3 get_transactions()

```
def core.queries.gets.ledger.get_transactions (
              st_dt,
              end_dt = dt.datetime.now() )


get the transactions within the given period exclusively

@param st_dt: the start datetime
@param end_dt: the end datetime
@return dataframe of the transactions
```

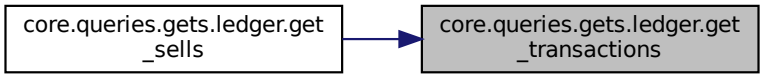
Definition at line 7 of file ledger.py.

```
7 def get_transactions(st_dt, end_dt=dt.datetime.now()):
8     """ get the transactions within the given period exclusively
9
10     @param st_dt: the start datetime
11     @param end_dt: the end datetime
12     @return dataframe of the transactions
13     """
14     stat = sql.SQL("SELECT * FROM ledger WHERE trx_dt>{st_dt} AND trx_dt<{end_dt};").\
15         format(st_dt=sql.Literal(st_dt), \
16                end_dt=sql.Literal(end_dt))
17     db_log.debug(stat)
18     return pd.read_sql(conn, stat)
19
```

References client.client.format.

Referenced by core.queries.gets.ledger.get_sells().

Here is the caller graph for this function:



6.24 core.queries.gets.owner Namespace Reference

Functions

- def get_all ()
- def get_good_owner (gid)
- def get_owner_goods (oid)

6.24.1 Function Documentation

6.24.1.1 get_all()

```
def core.queries.gets.owner.get_all ( )
```

Definition at line 6 of file owner.py.

```
6 def get_all():
7     query="SELECT * FROM owner;"
8     return pd.read_sql(query, conn, index_col='id')
9
```

6.24.1.2 get_good_owner()

```
def core.queries.gets.owner.get_good_owner (
              gid )
```

```
return owner id (oid) for the given gid

@param gid: good
@return the owner id
```

Definition at line 10 of file owner.py.

```
10 def get_good_owner(gid):
11     """return owner id (oid) for the given gid
12
13     @param gid: good
14     @return the owner id
15     """
16     query = sql.SQL("SELECT (owner_id) FROM owners WHERE good_id={gid}").\
17         format(gid=sql.Literal(gid))
18     db_log.debug(query)
19     return pd.read_sql(query, conn).ix[0]
20
```

References client.client.format.

6.24.1.3 get_owner_goods()

```
def core.queries.gets.owner.get_owner_goods (
              oid )
```

```
return the good assigned to the given owner id (oid)

@param oid: is the owner id
@return json dict of good's ids
```

Definition at line 21 of file owner.py.

```
21 def get_owner_goods(oid):
22     """return the good assigned to the given owner id (oid)
23
24     @param oid: is the owner id
25     @return json dict of good's ids
26     """
27     query = sql.SQL("SELECT (good_id) FROM owners WHERE owner_id={oid}").\
28         format(oid=sql.Literal(oid))
29     db_log.debug(query)
30     return pd.read_sql(query, conn).to_json()
```

References client.client.format.

6.25 core.queries.inserts Namespace Reference

Namespaces

- banking
- clients
- contacts
- credentials
- goods
- ledger
- owners

6.26 core.queries.inserts.banking Namespace Reference

Functions

- def insert_banking (cid, balance)

6.26.1 Function Documentation

6.26.1.1 insert_banking()

```
def core.queries.inserts.banking.insert_banking (
              cid,
              balance )


give the client with the given id (cid) banking account (CALLED AT SERVER SIDE)

@param cid: client id
@param balance: client account balance
```

Definition at line 6 of file banking.py.

```
6 def insert_banking(cid, balance):
7     """ give the client with the given id (cid) banking account (CALLED AT SERVER SIDE)
8
9     @param cid: client id
10     @param balance: client account balance
11     """
12     stat=sql.SQL("INSERT INTO banking (client_id, balance, balance_dt) VALUES ({cid}, {balance},
       {dt});"). \
13         format(cid=sql.Literal(cid), \
14                balance=sql.Literal(balance), \
15                dt=sql.Literal(dt.datetime.now().strftime(TIMESTAMP_FORMAT)))
16     db_log.debug(stat)
17     cur.execute(stat)
18     stat="SELECT currval(pg_get_serial_sequence('banking', 'id'));"
19     db_log.debug(stat)
20     cur.execute(stat);
21     return cur.fetchone()[0]
```

References client.client.format.

6.27 core.queries.inserts.clients Namespace Reference

Functions

- def insert_client (name)

6.27.1 Function Documentation

6.27.1.1 insert_client()

```
def core.queries.inserts.clients.insert_client (
              name )
```

```
add new client to the network (CALLED AT THE SERVER SIDE),

note that some clients might not have banking id yet
@param name: client name
```

Definition at line 4 of file clients.py.

```
4 def insert_client(name):
5     """ add new client to the network (CALLED AT THE SERVER SIDE),
6
7     note that some clients might not have banking id yet
8     @param name: client name
9     """
10     stat=sql.SQL("INSERT INTO clients (contact_name) VALUES ({name})").\
11         format(name=sql.Literal(name))
12     cur.execute(stat)
13     cur.execute("SELECT currval(pg_get_serial_sequence('clients', 'id'));")
14     return cur.fetchone()[0]
```

References client.client.format.

6.28 core.queries.inserts.contacts Namespace Reference

Functions

- def insert_contact (cid, cname, bid)

6.28.1 Function Documentation

6.28.1.1 insert_contact()

```
def core.queries.inserts.contacts.insert_contact (
              cid,
              cname,
              bid )

insert new contact (CALLED AT THE CLIENT SIDE)

@param cid: contact id (the same as client id in the server side)
@param cname: contact name
@param bid: bank account id
```

Definition at line 5 of file contacts.py.

```
5 def insert_contact(cid, cname, bid):
6     """ insert new contact (CALLED AT THE CLIENT SIDE)
7
8     @param cid: contact id (the same as client id in the server side)
9     @param cname: contact name
10     @param bid: bank account id
11     """
12     stat=sql.SQL("INSERT INTO contacts (contact_id, contact_name bank_account_id) VALUES ({cid}, {cname},
       {bid})").\
13         format(cid=sql.Literal(cid), \
14                cname=sql.Literal(cname), \
15                bid=sql.Literal(bid))
16     db_log.debug(stat)
17     cur.execute(stat)
```

References client.client.format.

6.29 core.queries.inserts.credentials Namespace Reference

Functions

- def new_cred (passcode, cred_id)
- def register (cid)

6.29.1 Function Documentation

6.29.1.1 new_cred()

```
def core.queries.inserts.credentials.new_cred (
              passcode,
              cred_id )

add client credentials returned from the server

@param cid: client id
```

Definition at line 9 of file credentials.py.

```
9 def new_cred(passcode, cred_id):
10     """add client credentials returned from the server
11
12     @param cid: client id
13     """
14     stat=sql.SQL("INSERT INTO credentials (passcode, cred_id) VALUES ({passcode}, {credid});").\
15         format(passcode=sql.Literal(passcode), \
16                credid=sql.Literal(cred_id))
17     db_log.debug(stat)
18     cur.execute(stat)
19
```

References client.client.format.

6.29.1.2 register()

```
def core.queries.inserts.credentials.register (
              cid )
```

```
register new client credentials with given cid (CALLED FROM SERVER SIDE)

@param cid: client id
@return a tuple (cred_id, passcode)
```

Definition at line 20 of file credentials.py.

```
20 def register(cid):
21     """register new client credentials with given cid (CALLED FROM SERVER SIDE)
22 
23     @param cid: client id
24     @return a tuple (cred_id, passcode)
25     """
26     cred_id=rand.random()*MAX_CRED_ID
27     passcode=''.join(rand.choice(string.ascii_uppercase+\
28                                  string.ascii_lowercase+string.digits)\
29                      for _ in range(9))
30     stat=sql.SQL("INSERT INTO credentials (id, passcode, cred_id) VALUES ({cid}, {passcode},
       {credid});").\
31         format(cid=sql.Literal(cid), \
32                passcode=sql.Literal(passcode), \
33                credid=sql.Literal(cred_id))
34     db_log.debug(stat)
35     cur.execute(stat)
36     return (cred_id, passcode)
37 
38 '''
39 def add_cred(passcode, cred_id):
40     """add client credentials returned from the server(CALLED FROM SERVER SIDE)
41 
42     @param cid: client id
43     """
44     stat=sql.SQL("INSERT INTO credentials (passcode, cred_id) VALUES ({passcode}, {credid});").\
45         format(passcode=sql.Literal(passcode), \
46                credid=sql.Literal(cred_id))
47     db_log.debug(stat)
48     cur.execute(stat)
49 '''
```

References client.client.format.

6.30 core.queries.inserts.goods Namespace Reference

Functions

- def add_good (gname, gquality, gcost, gcid=1)

6.30.1 Function Documentation

6.30.1.1 add_good()

```
def core.queries.inserts.goods.add_good (
              gname,
              gquality,
              gcost,
              gcid = 1 )
```

```
INSERT new good into the goods table

@param gname: good name
@param gquality: good quality
@param gcost: good cost
@param gcid: good currency id
```

Definition at line 5 of file goods.py.

```
5 def add_good(gname, gquality, gcost, gcid=1):
6     """ INSERT new good into the goods table
7
8     @param gname: good name
9     @param gquality: good quality
10     @param gcost: good cost
11     @param gcid: good currency id
12     """
13     stat=sql.SQL("INSERT INTO goods (good_name, good_quality, good_cost, good_currency_id) VALUES
       ({gname}, {gquality}, {gcost}, {gcid});").\
14         format(gname=sql.Literal(gname), \
15                gquality=sql.Literal(gquality), \
16                gcost=sql.Literal(gcost), \
17                gcid=sql.Literal(gcid))
18     db_log.debug(stat)
19     cur.execute(stat)
20     stat="SELECT currval(pg_get_serial_sequence('goods', 'id'));"
21     cur.execute(stat)
22     db_log.debug(stat)
23     return cur.fetchone()[0]
```

References client.client.format.

6.31 core.queries.inserts.ledger Namespace Reference

Functions

- def insert_trx (des, src, gid)

6.31.1 Function Documentation

6.31.1.1 insert_trx()

```
def core.queries.inserts.ledger.insert_trx (
              des,
              src,
              gid )

insert transaction from 'src' to 'des' for good with 'gid'

@param des: the transaction destination
@param src: the transaction source
@param gid: the good's id
```

Definition at line 5 of file ledger.py.

```
5 def insert_trx(des, src, gid):
6     """ insert transaction from 'src' to 'des' for good with 'gid'
7
8     @param des: the transaction destination
9     @param src: the transaction source
10     @param gid: the good's id
11     """
12     stat=sql.SQL("INSERT INTO ledger (trx_dest, trx_src, good_id) VALUES ({des}, {src}, {gid});").\
13         format(des=sql.Literal(des), \
14                src=sql.Literal(src), \
15                gid=sql.Literal(gid))
16     db_log.debug(stat)
17     cur.execute(stat)
```

References client.client.format.

6.32 core.queries.inserts.owners Namespace Reference

Functions

- def add_owner (oid, gid)

6.32.1 Function Documentation

6.32.1.1 add_owner()

```
def core.queries.inserts.owners.add_owner (
              oid,
              gid )

assign ownership of owner with id (oid) to the good with id (gid)

@param oid: owner id
@param gid: good id
```

Definition at line 5 of file owners.py.

```
5 def add_owner(oid, gid):
6     """assign ownership of owner with id (oid) to the good with id (gid)
7
8     @param oid: owner id
9     @param gid: good id
10     """
11     stat=sql.SQL("INSERT INTO owners (owner_id, good_id) VALUES ({oid}, {gid})").\
12         format(oid=sql.Literal(oid), \
13                gid=sql.Literal(gid))
14     db.log(stat)
15     cur.execute(stat)
```

References client.client.format.

6.33 core.queries.updates Namespace Reference

Namespaces

- banking
- owners

6.34 core.queries.updates.banking Namespace Reference

Functions

- def update_account (cid, balance)

6.34.1 Function Documentation

6.34.1.1 update_account()

```
def core.queries.updates.banking.update_account (
              cid,
              balance )
```

update the banking account with the calculated new balance (CALLED FROM SERVER SIDE)

@param cid: client id
@param balance: the account balance

Definition at line 6 of file banking.py.

```
6 def update_account(cid, balance):
7     """update the banking account with the calculated new balance (CALLED FROM SERVER SIDE)
8
9     @param cid: client id
10     @param balance: the account balance
11     """
12     stat = sql.SQL("UPDATE banking SET (balance, balance_dt) = ({balance}, {dt}) WHERE
       client_id={cid}").\
13         format(balance=sql.Literal(balance), \
14                dt=dt.datetime().strftime(TIMESTAMP_FORMAT), \
15                cid=sql.Literal(cid))
16     cur.execute(stat)
```

References client.client.format.

6.35 core.queries.updates.owners Namespace Reference

Functions

- def update_owner (oid, gid)

6.35.1 Function Documentation

6.35.1.1 update_owner()

```
def core.queries.updates.owners.update_owner (
              oid,
              gid )
```

```
reassign the good's ownership with corresponding gid

@param gid: good id
```

Definition at line 4 of file owners.py.

```
4 def update_owner(oid, gid):
5     """reassign the good's ownership with corresponding gid
6
7     @param gid: good id
8     """
9     stat = sql.SQL("UPDATE owners SET (owner_id) = {oid} WHERE good_id={gid}").\
10         format(oid=sql.Literal(oid), \
11                bid=sql.Literal(gid))
12     cur.execute(stat)
```

References client.client.format.

6.36 core.server Namespace Reference

Namespaces

- server

6.37 core.server.server Namespace Reference

Functions

- def get_credid ()
- def is_email (email)
- def authenticate (user, passcode)
- def unauthorized ()
- def register_client ()
- def add_bank_account ()
- def add_contact ()
- def get_balance ()
- def update_balance_preference ()
- def update_ledger ()
- def make_transaction ()

Variables

- seed = int.from_bytes(os.urandom(2), 'big')
- string db_configs = "dbname='demo' user='tahweela' password='tahweela'"
- db = database(db_configs)
- filename
- format
- filemode
- level
- logger = logging.getLogger()
- app = Flask('tahweela')
- auth = HTTPBasicAuth()
- client_passcode = None
- client_cred_id = None
- debug

6.37.1 Function Documentation

6.37.1.1 add_bank_account()

```
def core.server.server.add_bank_account ( )
```

```
register bank account for the authenticated client of the current session

@param: the post body is expect to be json with keys ["bank_name", branch_number", "account_number", "name_ref
@return return bid (banking id), since multiple bank accounts are supported, bid need to be sent for each tran
```

Definition at line 194 of file server.py.

```
194 def add_bank_account():
195   """ register bank account for the authenticated client of the current session
196
197   @param: the post body is expect to be json with keys ["bank_name", branch_number", "account_number",
       "name_reference"], a client can register more than one bank account for tahweela account.
198   @return return bid (banking id), since multiple bank accounts are supported, bid need to be sent for
       each transaction so that, the transactions are done with it.
199   """
200   req=request.get_json(force=True)
201   bank_name=req.get("bank_name", None)
202   branch_number=req.get("branch_number", None)
203   account_number=req.get("account_number", None)
204   name_reference=req.get("name_reference", "")
205   if not req or bank_name==None or branch_number==None or account_number==None:
206     logger.critical("incomplete request")
207     abort(401)
208   db.init()
209   ADD_BANK_ACCOUNT_LOCK=7
210   lock=ADD_BANK_ACCOUNT_LOCK
211   try:
212     db.lock_advisory(lock)
213     cid=db.gets.credid2cid(client_cred_id)
214     logger.debug("client added")
215     bank=PaymentGate(bank_name, branch_number, account_number, name_reference)
216     if not bank.authenticated():
217         raise Exception('payment gate authentication failure!')
218     balance_dt=bank.get_balance()
219     balance=balance_dt['balance']
220     base_currency=balance_dt['base']
221     if not db.exists.currency(base_currency):
222         currency=Currency(base_currency)
223         db.inserts.add_currency(base_currency, currency.rate)
224     base_currency_id=db.gets.get_currency_id(base_currency)
225     db.inserts.add_bank_account(cid, balance, bank_name, branch_number, account_number, name_reference,
      base_currency_id)
```

```
226     db.commit(lock)
227   except psycopg2.DatabaseError as error:
228     print('err1')
229     db.rollback(lock)
230     logger.critical("assigning bank account failed, error: "+str(error))
231     abort(300)
232   except Exception as error:
233     print('err2')
234     db.rollback(lock)
235     logger.critical("adding bank account failed, error: ", str(error))
236     abort(300)
237   finally:
238     db.close()
239   return jsonify({'balance':balance, 'base': base_currency}), 201
240
241 @app.route(CONTACTS, methods=['POST'])
242 @auth.login_required
```

6.37.1.2 add_contact()

```
def core.server.server.add_contact ( )
```

get the credential for the given contact

Definition at line 243 of file server.py.

```
243 def add_contact():
244     """get the credential for the given contact
245     """
246     req=request.get_json(force=True)
247     email=req.get('email', None)
248     logger.info("requesting contact")
249     if not req or email==None:
250         logger.debug("incomplete URL")
251         abort(401)
252     payload={}
253     db.init()
254     try:
255         db.repeatable_read()
256         if not db.exists.account_byemail(email):
257             logger.critical("contact doesn't exist")
258             raise Exception("contact doesn't exists!")
259         cid=db.gets.get_client_id_byemail(email)
260         credid=db.gets.cid2credid(cid)
261         # get name given cid
262         name=db.gets.get_name(cid)
263         payload = {"credid": credid}
264         db.commit()
265     except psycopg2.DatabaseError as error:
266         db.rollback()
267         logger.critical("request failed, error:"+str(error))
268         abort(300)
269     except Exception as error:
270         db.rollback()
271         logger.critical("requesting failed"+str(error))
272         print('FAILURE, err: ', str(error))
273         abort(400)
274     finally:
275         db.close()
276     return jsonify(payload), 201
277
278 @app.route(BALANCE, methods=['GET'])
279 @auth.login_required
```

6.37.1.3 authenticate()

```
def core.server.server.authenticate (
              user,
              passcode )
```

authenticate user, with username/password

user can be user's email, or registered name

Definition at line 49 of file server.py.

```
49 def authenticate(user, passcode):
50     """authenticate user, with username/password
51 
52     user can be user's email, or registered name
53     """
54     print("AUTHENTICATING.....")
55     '''
56     #TODO (fix) it seams that is_email isn't strong enough, it fails for some emails
57     if is_email(user):
58       print('authenticating user by email: ', user)
59       if db.exists.account_byemail(user):
60         cid=db.gets.get_client_id_byemail(user)
61       else:
62         cid=-1
63     else:
64       print('authenticating user by name', user)
65       if db.exists.account_byname(user, passcode):
66         cid=db.gets.get_client_id_byname(user)
67       else:
68         cid=-1
69     '''
70     print('authenticating [{}+{}]...'.format(user, passcode))
71     print('authenticating user by email: ', user)
72     if db.exists.account_byemail(user):
73       cid=db.gets.get_client_id_byemail(user)
74     else:
75       print('doesnt exist!')
76       cid=-1
77     if cid==-1:
78       print('authenticating user by name', user)
79       if db.exists.account_byname(user, passcode):
80         cid=db.gets.get_client_id_byname(user)
81       else:
82         print('doesnt exist!')
83         cid=-1
84     #TODO support user as credid itself
85     if cid==-1: #doesn't exists
86         logger.critical("authentication failed")
87         print('authentication failed ,doesnt exist')
88         return None
89     print('SET GLOBAL!')
90     #TODO HOW TO MAKE SURE THAT CREDID IS UNIQUE?
91     # brute force, keep trying new values that does exist,
92     # is the simplest, otherwise use conguential generator
93     global client_cred_id, client_passcode
94     credid=db.gets.cid2credid(cid)
95     #TODO implement cid2credid
96     logger.debug("authenticating client with cred {}:{}". format(user, passcode))
97     print('credid: ', credid)
98     print("username: ", user)
99     print("password: ", passcode)
100     db.init()
101     try:
102       db.repeatable_read()
103       passcode_eq= db.gets.get_password(credid)
104       print("pass_code: ", passcode_eq)
105       logger.info("username:"+str(user)+", passcode:"+str(passcode))
106       if not passcode==passcode_eq:
107         print('FAILURE PASSOWRD MISMATCH')
108         raise Exception("password mismatch!")
109       db.commit()
110       print('AUTHENTICATION SUCCESS....')
111     except psycopg2.DatabaseError as error:
112       print('AUTHENTICATION FAILURE')
113       db.rollback()
114       logger.critical("authentication failed with error: "+str(error))
115       abort(300)
116       return None #doesn't reach here
117     except:
```

```
118       print('AUTHENTICATION FAILURE')
119       db.rollback()
120       logger.critical("authentication failed ")
121       abort(300)
122       return None #doesn't reach here!
123     finally:
124       db.close()
125     client_cred_id=credid
126     client_passcode=passcode
127     return user
128
129 @auth.error_handler
```

References core.server.server.format.

6.37.1.4 get_balance()

```
def core.server.server.get_balance ( )
```

get balance of the current client

@return {'balance': balance, 'base': base}

Definition at line 280 of file server.py.

```
280 def get_balance():
281     """ get balance of the current client
282
283     @return {'balance': balance, 'base': base}
284     """
285     balance=None
286     logger.info("balance requested")
287     db.init()
288     try:
289         db.repeatable_read()
290         cid=db.gets.credid2cid(client_cred_id)
291         if not db.exists.bank_account_bycid(cid):
292             raise Exception("no bank account added yet!")
293         #this would return balance in bank base
294         balance=db.gets.get_balance_by_credid(client_cred_id)
295         # transform balance to user preference
296         pref_cur=db.gets.get_preference_currency_bycid(cid)
297         amount=balance['balance']
298         print('|--------------> amount {}'.format(amount))
299         base=balance['base']
300         currency=Currency(pref_cur, base)
301         pref_balance=currency.exchange(amount)
302         print('|--------------> pref_balance {}'.format(pref_balance))
303         payload={'balance': pref_balance, 'base':pref_cur}
304         db.commit()
305     except psycopg2.DatabaseError as error:
306         db.rollback()
307         logger.critical("failed request, error: "+str(error))
308         abort(300)
309     except:
310         db.rollback()
311         logger.critical("failed request")
312         abort(300)
313     finally:
314         db.close()
315     return jsonify(payload), 201
316
317 @app.route(CURRENCY, methods=['POST'])
318 @auth.login_required
```

References core.server.server.format.

6.37.1.5 get_credid()

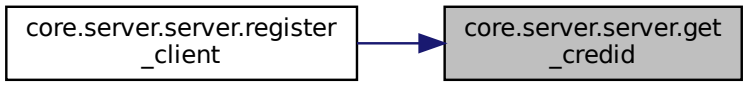
```
def core.server.server.get_credid ( )
```

Definition at line 41 of file server.py.

```
41 def get_credid():
42     return int(3333333333333*random.random())
43
```

Referenced by core.server.server.register_client().

Here is the caller graph for this function:



6.37.1.6 is_email()

```
def core.server.server.is_email (
               email )
```

Definition at line 44 of file server.py.

```
44 def is_email(email):
45   return bool(re.search(r"^[\w\.\+\-]+\@[\w]+\.[a-z]{2,3}$", email))
46
47
48 @auth.verify_password
```

6.37.1.7 make_transaction()

```
def core.server.server.make_transaction ( )
```

Definition at line 389 of file server.py.

```
389 def make_transaction():
390     req=request.get_json(force=True)
391     recipt_credid=req['credid']
392     # the amount of transaction in Euro
393     orig_amount=req['amount']
394     currency_base=req.get('currency', EUR)
395     #exchange amount to euro for processing
396     to_euro = Currency(EUR, currency_base)
397     amount=to_euro.exchange(orig_amount)
398     trx_name=req.get('trx_name', '')
399     #TRANSACTION LIMITS IN EUROS
400     max_daily=daily_limit()
401     max_weekly=weekly_limit()
402     #here the weekly/daily conditions are pivoted by the current moment only, note that the bank system
      can have specific pivot hour (the first momemnt of the day, it's specific for the bank system, and
      need to be known before hand)
403     week_past=datetime.datetime.now()-datetime.timedelta(days=7)
404     day_past=datetime.datetime.now()-datetime.timedelta(days=1)
```

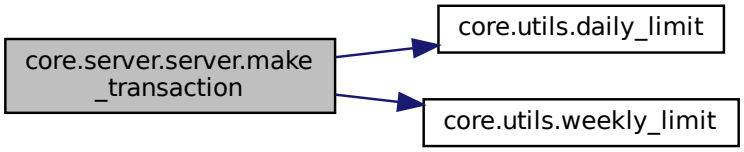
```
405     #TODO abide to the  the constrains
406     logger.info("making purchase")
407     if not req or amount==None or recipt_credid==None:
408         logger.critical("incomplete URL empty request")
409         abort(401)
410     #gid=req['id']
411     db.init()
412     MAKE_TRANSACTION_LOCK=9
413     lock=MAKE_TRANSACTION_LOCK
414     print('start transaction')
415     try:
416         db.lock_advisory(lock)
417         #if this client have a bank account yet
418         cid=db.gets.credid2cid(client_cred_id)
419         if not db.exists.bank_account_bycid(cid):
420             raise Exception("client doesn't have any associated bank account!")
421         #balance in bank base
422         src_balance=db.gets.get_balance_by_credid(client_cred_id)
423         src_balance_exchange=Currency(EUR, src_balance['base'])
424         src_balance_euro=src_balance_exchange.exchange(src_balance['balance'])
425         if src_balance_euro<amount+FEE:
426             logger.info("client doesn't have enough credit to make transaction")
427             raise Exception("no enough balance to make transaction")
428         #get transaction sum in euro
429         weekly_trx_sum=db.gets.get_transactions_sum(client_cred_id, week_past)
430         daily_trx_sum=db.gets.get_transactions_sum(client_cred_id, day_past)
431         print('got trx sum! weekly: {}, daily{}'.format(weekly_trx_sum, daily_trx_sum))
432         if weekly_trx_sum+amount>max_weekly or daily_trx_sum+amount>max_daily:
433             logger.info("client passed the daily/weekly limit")
434             raise Exception("client passed the daily/weekly limits")
435         cur_id=db.gets.get_currency_id(currency_base)
436         #add transaction
437         db.inserts.insert_trx(recipt_credid, client_cred_id, amount, cur_id, trx_name)
438         #TODO this can be minimized directly by credid
439         #dest balance in bank base
440         dest_balance=db.gets.get_balance_by_credid(recipt_credid)
441         dest_balance_exchange=Currency(EUR, dest_balance['base'])
442         dest_balance_euro=dest_balance_exchange.exchange(dest_balance['balance'])
443         src_balance_new=src_balance_euro-(amount+FEE)
444         dest_balance_new=dest_balance_euro+amount
445         #exchange back to bank bas
446         src_balance_new=src_balance_exchange.exchange_back(src_balance_new)
447         dest_balance_new=dest_balance_exchange.exchange_back(dest_balance_new)
448         src_cid=db.gets.credid2cid(client_cred_id)
449         des_cid=db.gets.credid2cid(recipt_credid)
450         if src_cid==des_cid:
451             logger.critical("you can't make transaction with oneself!")
452             abort(403)
453             raise Exception("you can't make transaction with oneself!")
454         db.updates.update_account(src_cid, src_balance_new)
455         db.updates.update_account(des_cid, dest_balance_new)
456         trx = {'trx_dest': recipt_credid,  \
457                'trx_src': client_cred_id, \
458                'trx_cost': orig_amount, \
459                'trx_name':trx_name}
460         payload={'balance': src_balance_new, \
461                  'transactions': trx}
462         db.commit()
463     except psycopg2.DatabaseError as error:
464         db.rollback()
465         logger.critical("transaction failed, error: "+str(error))
466         abort(300)
467     except:
468         db.rollback()
469         logger.critical("transaction failed")
470         abort(300)
471     finally:
472         db.unlock_advisory(lock)
473         db.close()
474     return jsonify(payload), 201
475 '''
476 @app.route(GOODS, methods=['POST'])
477 @auth.login_required
478 def add_goods():
479     req=request.get_json(force=True)
480     logger.info("adding new good to the market")
481     #TODO goods should be passed with authentication,
482     # how to authenticate both in requests, and in flask
483     if not req or 'goods' not in req:
484         logger.critical("URL is incomplete missing goods")
485         abort(401)
486     goods={}
487     db.init()
488     lock=1
489     try:
490         db.lock_advisory(lock)
491         cid=db.gets.credid2cid(client_cred_id)
```

```
492         if cid==None:
493             logger.critical("client not found associated with given id")
494             abort(403)
495             raise Exception("invalid client")
496         goods=req['goods']
497         for good in goods:
498             print("good", good[0], good[1], good[2])
499             gid=db.inserts.add_good(good[0], good[1], good[2])
500             db.inserts.add_owner(cid, gid)
501         db.commit(lock)
502     except psycopg2.DatabaseError as error:
503         db.rollback(lock)
504         logger.critical("error adding good, error: "+str(error))
505         abort(300)
506     except:
507         db.rollback(lock)
508         logger.critical("error adding good")
509         abort(300)
510     finally:
511         db.close()
512     return jsonify({'goods':goods}), 201
513
514 @app.route(GOODS, methods=['GET'])
515 @auth.login_required
516 def get_goods():
517     logger.info("requesting good")
518     goods={}
519     db.init()
520     try:
521         db.repeatable_read()
522         goods = db.gets.get_all_goods()
523         print('goods:', goods)
524         db.commit()
525     except psycopg2.DatabaseError as error:
526         print('db except!')
527         db.rollback()
528         logger.critical("process failed, error: "+str(error))
529         abort(300)
530     except:
531         print("except")
532         db.rollback()
533         logger.critical("process failed")
534         abort(300)
535     finally:
536         db.close()
537         #TODO change column names
538     return jsonify({"goods": goods}), 201
539 @app.route(PURCHASE, methods=['POST'])
540 @auth.login_required
541 def make_purchase():
542     req=request.get_json(force=True)
543     logger.info("making purchase")
544     if not req:
545         logger.critical("incomplete URL empty request")
546         abort(401)
547     gid=req['id']
548     db.init()
549     lock=4
550     try:
551         db.lock_advisory(lock)
552         # cross reference owner_id, with good_id, with credentials
553         # return credential id of the owner
554         credid=db.gets.get_credid_with_gid(gid)
555         # make, and add new transaction such that increase,
556         # and decrease of the src/des balance need to be performed in single transaction, then add the
    transactioin to the ledger, if failed rollback
557         cost=db.gets.get_new_price(gid)+FEE
558         src_balance=db.gets.get_balance_by_credid(client_cred_id)-(cost+FEE)
559         des_balance=db.gets.get_balance_by_credid(credid)+cost
560         src_cid=db.gets.credid2cid(client_cred_id)
561         des_cid=db.gets.credid2cid(credid)
562         if src_cid==des_cid:
563             logger.critical("you can't make purchase with oneself!")
564             abort(403)
565             raise Exception("you can't make purchase with oneself!")
566         db.updates.update_account(src_cid, src_balance)
567         db.updates.update_account(des_cid, des_balance)
568         #TODO the ownership in the client side need to be updated as well,
569         # in the database, and in the stateful list in memory
570         # also fetch corresponding oid!
571         #update_owner(oid, gid)
572         trx = {'trx_dest': credid,
573                'trx_src': client_cred_id,
574                'good_id': gid}
575         payload={'balance': src_balance,
576                  'transactions': trx}
577         db.commit()
```

```
578     except psycopg2.DatabaseError as error:
579         db.rollback()
580         logger.critical("purchase failed, error: "+str(error))
581         abort(300)
582     except:
583         db.rollback()
584         logger.critical("purchase failed")
585         abort(300)
586     finally:
587         db.unlock_advisory(lock)
588         db.close()
589     return jsonify(payload), 201
590 "'
591
```

References core.utils.daily_limit(), core.server.server.format, and core.utils.weekly_limit().

Here is the call graph for this function:



6.37.1.8 register_client()

```
def core.server.server.register_client ( )
```

```
register new client with email/name + password, expected json body would have keys ["name", "email", "passcode
```

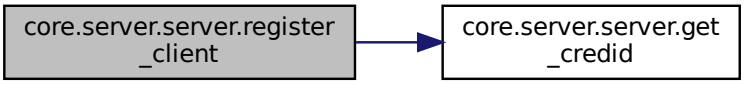
Definition at line 135 of file server.py.

```
135 def register_client():
136   """register new client with email/name + password, expected json body would have keys ["name",
       "email", "passcode"]
137
138   """
139   #TODO this can through exception, need to handle it
140   req=request.get_json(force=True)
141   print('req: ', req)
142   name=req.get('name', None)
143   passcode=req.get('passcode', None)
144   email=req.get('email', None)
145   cur_pref=req.get('cur_pref', EUR)
146   if not req or name==None or email==None or passcode==None:
147     logger.critical("url is incomplete missing name")
148     print("incomplete!!!")
149     abort(400)
150   #TODO add constraint, and verification for the name/email, and passcode (at least not None!)
151   cred_id=get_credid()
152   logger.info("registering trader for client: "+ name)
153   bid=0
154   db.init()
155   lock=2
156   #TODO generalize get_credid
157   #TODO add errors to response code
158   try:
159       db.lock_advisory(lock)
160       email_exists=db.exists.account_byemail(email)
```

```
161       if email_exists:
162           print('email exists')
163           abort(400)
164           logger.debug("account {}/{} + {} already exists".\
165                        format(name, email, passcode))
166           raise Exception("account already exists!")
167       if not db.exists.currency(cur_pref):
168           currency=Currency(cur_pref)
169           if not currency.valid():
170               raise Exception("currency isn't supported!")
171           db.inserts.add_currency(cur_pref, currency.rate)
172       cur_pref_id=db.gets.get_currency_id(cur_pref)
173       cid=db.inserts.add_client(req['name'], req['email'], cur_pref_id)
174       logger.debug("client added")
175       db.inserts.register(cid, passcode, cred_id)
176       db.commit(lock)
177     except psycopg2.DatabaseError as error:
178       print('REGISTRATION FAILURE')
179       db.rollback(lock)
180       logger.critical("registering failed, error: "+str(error))
181       abort(300)
182     except:
183       print('REGISTRATION FAILURE')
184       db.rollback(lock)
185       logger.critical("registering failed")
186       abort(400)
187     finally:
188       db.close()
189       res = {'cred_id': cred_id}
190     return jsonify(res), 201
191
192 @app.route(ADD_BANK_ACCOUNT, methods=['POST'])
193 @auth.login_required
```

References core.server.server.format, and core.server.server.get_credid().

Here is the call graph for this function:



6.37.1.9 unauthorized()

```
def core.server.server.unauthorized ( )
```

Definition at line 130 of file server.py.

```
130 def unauthorized():
131     return make_response(jsonify({'error': "forbidden access"}), 403)
132
133 #TODO add message to status code response
134 @app.route(REGISTER, methods=['POST'])
```

6.37.1.10 update_balance_preference()

```
def core.server.server.update_balance_preference ( )
```

update balance preference

Definition at line 319 of file server.py.

```
319 def update_balance_preference():
320     """update balance preference
321 
322     """
323     req=request.get_json(force=True)
324     base = req.get('base', None)
325     logger.info("updating balance preference")
326     if not req or base==None:
327         logger.cirtical('incomplete url')
328         abort(401)
329     CURRENCY_LOCK=11
330     lock=CURRENCY_LOCK
331     db.init()
332     try:
333         db.lock_advisory(lock)
334         if not db.exists.currency(base):
335             currency=Currency(base)
336             db.inserts.add_currency(basey, currency.rate)
337         base_currency_id=db.gets.get_currency_id(base_currency)
338         cid=db.gets.credid2cid(client_cred_id)
339         db.updates.currency_preference(cid, base)
340         db.commit()
341     except psycopg2.DatabaseError as error:
342         db.rollback()
343         logger.critical("request failure, error: "+ str(error))
344         abort(300)
345     except Exception as error:
346         db.rollback()
347         logger.critical("request failure, error: "+ str(error))
348         abort(300)
349     finally:
350         db.unlock_advisory(lock)
351         db.close()
352 
353 @app.route(LEDGER, methods=['POST'])
354 @auth.login_required
```

6.37.1.11 update_ledger()

```
def core.server.server.update_ledger ( )
```

Definition at line 355 of file server.py.

```
355 def update_ledger():
356     req=request.get_json(force=True)
357     logger.info("requesting ledger update")
358     if not req:
359         logger.critical("incomplete URL empty request!")
360         abort(401)
361     st_dt=req['trx_dt']
362     payload={}
363     db.init()
364     lock=3
365     try:
366         db.lock_advisory(lock)
367         print("getting sells")
368         sells_trax=db.gets.get_sells(client_cred_id, st_dt).to_json()
369         print("sells: ", sells_trax)
370         balance=db.gets.get_balance_by_credid(client_cred_id)
371         payload={'transactions': sells_trax, \
372                  'balance': balance}
373         db.commit()
374     except psycopg2.DatabaseError as error:
375         db.rollback()
376         logger.critical("request failure, error: "+ str(error))
377         abort(300)
378     except:
```

```
379         db.rollback()
380         logger.critical("request failure")
381         abort(300)
382     finally:
383         db.unlock_advisory(lock)
384         db.close()
385     return jsonify(payload), 201
386
387 @app.route(TRANSACTION, methods=['POST'])
388 @auth.login_required
```

6.37.2 Variable Documentation

6.37.2.1 app

```
core.server.server.app = Flask('tahweela')
```

Definition at line 32 of file server.py.

6.37.2.2 auth

```
core.server.server.auth = HTTPBasicAuth()
```

Definition at line 33 of file server.py.

6.37.2.3 client_cred_id

```
core.server.server.client_cred_id = None
```

Definition at line 37 of file server.py.

6.37.2.4 client_passcode

```
core.server.server.client_passcode = None
```

Definition at line 36 of file server.py.

6.37.2.5 db

```
core.server.server.db = database(db_configs)
```

Definition at line 25 of file server.py.

6.37.2.6 db_configs

```
string core.server.server.db_configs = "dbname='demo' user='tahweela' password='tahweela'"
```

Definition at line 22 of file server.py.

6.37.2.7 debug

```
core.server.server.debug
```

Definition at line 593 of file server.py.

6.37.2.8 filemode

```
core.server.server.filemode
```

Definition at line 28 of file server.py.

6.37.2.9 filename

```
core.server.server.filename
```

Definition at line 26 of file server.py.

6.37.2.10 format

```
core.server.server.format
```

Definition at line 27 of file server.py.

Referenced by core.server.server.authenticate(), core.server.server.get_balance(), core.server.server.make_↩ transaction(), and core.server.server.register_client().

6.37.2.11 level

```
core.server.server.level
```

Definition at line 29 of file server.py.

6.37.2.12 logger

```
core.server.server.logger = logging.getLogger()
```

Definition at line 30 of file server.py.

6.37.2.13 seed

```
core.server.server.seed = int.from_bytes(os.urandom(2), 'big')
```

Definition at line 11 of file server.py.

6.38 core.utils Namespace Reference

Data Structures

- class Currency
- class PaymentGate

Functions

- def exchangerate_rate (base, pref)
- def fixer_rate (base, pref)
- def exchange (base, pref)
- def daily_limit (pref=EUR)
- def weekly_limit (pref=EUR)
- def process_cur (cur)
- def unwrap_cur (cur)

Variables

- seed = int.from_bytes(os.urandom(2), 'big')
- string TIMESTAMP_FORMAT = "%Y-%m-%d %H:%M:%S.%f"
- float FEE = 0.01
- float QUALITY_REDUCTION = 0.1
- int MAX_COST = 1000000
- int MAX_GOODS = 100
- string REGISTER = "/api/v0.1/register"
- string LEDGER = "/api/v0.1/ledger"
- string CONTACTS = "/api/v0.1/contacts"
- string PURCHASE = "/api/v0.1/purchase"
- string GOODS = "/api/v0.1/goods"
- string GOODS_URL = "http://localhost:5000/api/v0.1/goods"
- string BALANCE_URL = "http://localhost:5000/api/v0.1/balance"
- string BALANCE = "/api/v0.1/balance"
- string CURRENCY_URL = "http://localhost:5000/api/v0.1/currency"
- string CURRENCY = "/api/v0.1/currency"

- string CONTACTS_URL = "http://localhost:5000/api/v0.1/contacts"
- string REGISTER_URL = "http://localhost:5000/api/v0.1/register"
- string LEDGER_URL = "http://localhost:5000/api/v0.1/ledger"
- string PURCHASE_URL = "http://localhost:5000/api/v0.1/purchase"
- string ADD_BANK_ACCOUNT_URL = "http://localhost:5000/api/v0.1/addbank"
- string ADD_BANK_ACCOUNT = "/api/v0.1/addbank"
- string TRANSACTION_URL = "http://localhost:5000/api/v0.1/transaction"
- string TRANSACTION = "/api/v0.1/transaction"
- int MAX_CRED_ID = 9223372036854775807
- int MAX_BALANCE = MAX_COST∗10
- float STOCHASTIC_TRADE_THRESHOLD = 0.9
- int DAILY_LIMIT_EGP = 10000
- int WEEKLY_LIMIT_EGP = 50000
- string EUR = 'EUR'
- string EGP = 'EGP'
- string USD = 'USD'
- string db_configs = "dbname='demo' user='tahweela' password='tahweela'"
- filename
- format
- filemode
- log = logging.getLogger()

6.38.1 Function Documentation

6.38.1.1 daily_limit()

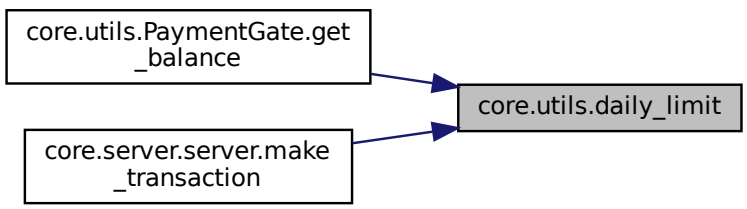
```
def core.utils.daily_limit (
               pref = EUR )

daily transaction limits
```

Definition at line 134 of file utils.py.

```
134 def daily_limit(pref=EUR):
135     """ daily transaction limits
136
137     """
138     currency = Currency(pref, EGP)
139     return currency.exchange(DAILY_LIMIT_EGP)
140
```

Referenced by core.utils.PaymentGate.get_balance(), and core.server.server.make_transaction().

Here is the caller graph for this function:



6.38.1.2 exchange()

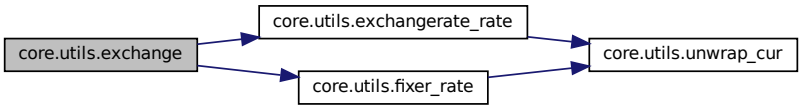
```
def core.utils.exchange (
              base,
              pref )
```

Definition at line 84 of file utils.py.

```
84 def exchange(base, pref):
85     rate=0
86     while rate==0:
87         rate=exchangerate_rate(base, pref)
88         if rate==0:
89             rate=fixer_rate(base, pref)
90         sleep(1)
91         print("make sure there is internet connection!")
92     return rate
93
```

References core.utils.exchangerate_rate(), and core.utils.fixer_rate().

Here is the call graph for this function:



6.38.1.3 exchangerate_rate()

```
def core.utils.exchangerate_rate (
              base,
              pref )
```

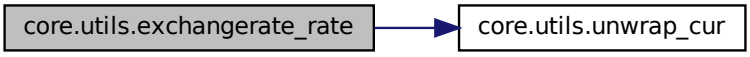
Definition at line 58 of file utils.py.

```
58 def exchangerate_rate(base, pref):
59     site='https://api.exchangeratesapi.io/latest?base={}'.format(unwrap_cur(base))
60     res=requests.get(site)
61     try:
62         response=res.json()
63         assert res.status_code<300 , 'exchange currency failed'
64         rate=response['rates'][unwrap_cur(pref)]
65     except:
66         log.critical('exchange rate fetch failed base: {}, pref: {}'. format(base, pref))
67         rate=0
68     return rate
69
```

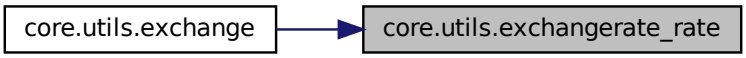
References core.utils.format, and core.utils.unwrap_cur().

Referenced by core.utils.exchange().

Here is the call graph for this function:



Here is the caller graph for this function:



6.38.1.4 fixer_rate()

```
def core.utils.fixer_rate (
              base,
              pref )
```

Definition at line 70 of file utils.py.

```
70 def fixer_rate(base, pref):
71     site='http://data.fixer.io/api/latest?access_key=9c87591ccfb9716f0850e500ceceef7a'
72     res=requests.get(site)
73     try:
74         response=res.json()
75         assert res.status_code<300 , 'exchange currency failed'
76         base_rate=response['rates'][unwrap_cur(base)]
77         pref_rate=response['rates'][unwrap_cur(pref)]
78         rate=base_rate/pref_rate
79     except:
80         log.critical('fixer.io rate fetch failed base: {}, pref: {}'. format(base, pref))
81         rate=0
82     return rate
83
```

References core.utils.format, and core.utils.unwrap_cur().

Referenced by core.utils.exchange().

Here is the call graph for this function:

Here is the caller graph for this function:

6.38.1.5 process_cur()

```
def core.utils.process_cur (
              cur )
```

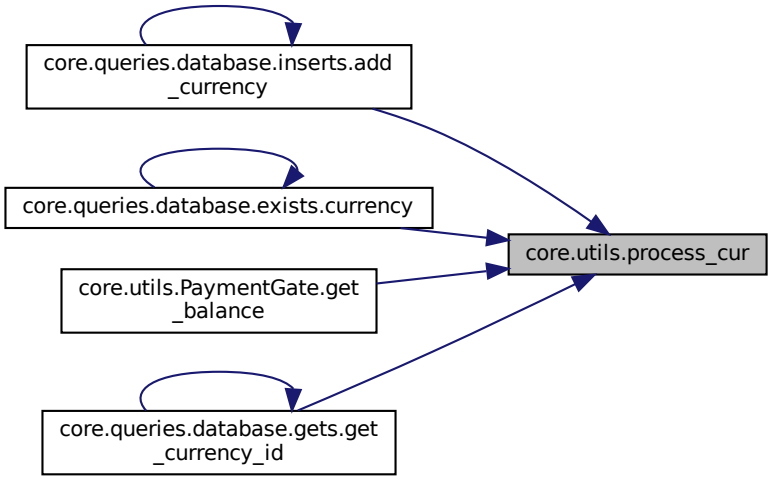
Definition at line 148 of file utils.py.

```
148 def process_cur(cur):
149     cur.strip("'")
150     cur.replace("'", '')
151     return '\'{}\''.format(cur)
```

References core.utils.format.

Referenced by core.queries.database.inserts.add_currency(), core.queries.database.exists.currency(), core.utils.↩
PaymentGate.get_balance(), and core.queries.database.gets.get_currency_id().

Here is the caller graph for this function:



6.38.1.6 unwrap_cur()

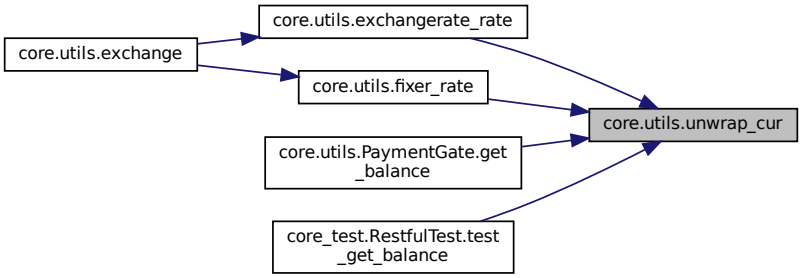
```
def core.utils.unwrap_cur (
              cur )
```

Definition at line 152 of file utils.py.

```
152 def unwrap_cur(cur):
153     return cur.replace("'", '')
```

Referenced by core.utils.exchangerate_rate(), core.utils.fixer_rate(), core.utils.PaymentGate.get_balance(), and core_test.RestfulTest.test_get_balance().

Here is the caller graph for this function:



6.38.1.7 weekly_limit()

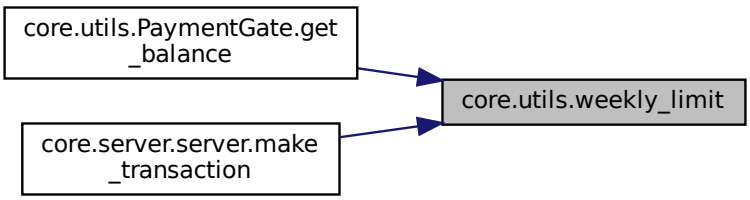
```
def core.utils.weekly_limit (
              pref = EUR )

daily transaction limits
```

Definition at line 141 of file utils.py.

```
141 def weekly_limit(pref=EUR):
142     """ daily transaction limits
143 
144     """
145     currency = Currency(pref, EGP)
146     return currency.exchange(WEEKLY_LIMIT_EGP)
147 
```

Referenced by core.utils.PaymentGate.get_balance(), and core.server.server.make_transaction().

Here is the caller graph for this function:



6.38.2 Variable Documentation

6.38.2.1 ADD_BANK_ACCOUNT

```
string core.utils.ADD_BANK_ACCOUNT = "/api/v0.1/addbank"
```

Definition at line 36 of file utils.py.

6.38.2.2 ADD_BANK_ACCOUNT_URL

```
string core.utils.ADD_BANK_ACCOUNT_URL = "http://localhost:5000/api/v0.1/addbank"
```

Definition at line 35 of file utils.py.

6.38.2.3 BALANCE

```
string core.utils.BALANCE = "/api/v0.1/balance"
```

Definition at line 25 of file utils.py.

6.38.2.4 BALANCE_URL

```
string core.utils.BALANCE_URL = "http://localhost:5000/api/v0.1/balance"
```

Definition at line 24 of file utils.py.

6.38.2.5 CONTACTS

```
string core.utils.CONTACTS = "/api/v0.1/contacts"
```

Definition at line 18 of file utils.py.

6.38.2.6 CONTACTS_URL

```
string core.utils.CONTACTS_URL = "http://localhost:5000/api/v0.1/contacts"
```

Definition at line 30 of file utils.py.

6.38.2.7 CURRENCY

```
string core.utils.CURRENCY = "/api/v0.1/currency"
```

Definition at line 28 of file utils.py.

6.38.2.8 CURRENCY_URL

```
string core.utils.CURRENCY_URL = "http://localhost:5000/api/v0.1/currency"
```

Definition at line 27 of file utils.py.

6.38.2.9 DAILY_LIMIT_EGP

```
int core.utils.DAILY_LIMIT_EGP = 10000
```

Definition at line 43 of file utils.py.

6.38.2.10 db_configs

```
string core.utils.db_configs = "dbname='demo' user='tahweela' password='tahweela'"
```

Definition at line 50 of file utils.py.

6.38.2.11 EGP

```
string core.utils.EGP = 'EGP'
```

Definition at line 47 of file utils.py.

6.38.2.12 EUR

```
string core.utils.EUR = 'EUR'
```

Definition at line 46 of file utils.py.

6.38.2.13 FEE

```
float core.utils.FEE = 0.01
```

Definition at line 11 of file utils.py.

6.38.2.14 filemode

```
core.utils.filemode
```

Definition at line 54 of file utils.py.

6.38.2.15 filename

```
core.utils.filename
```

Definition at line 52 of file utils.py.

6.38.2.16 format

```
core.utils.format
```

Definition at line 53 of file utils.py.

Referenced by core.utils.Currency.__init__(), core.utils.Currency.exchange(), core.utils.Currency.exchange_back(), core.utils.exchangerate_rate(), core.utils.fixer_rate(), core.utils.PaymentGate.get_balance(), and core.utils.process_cur().

6.38.2.17 GOODS

```
string core.utils.GOODS = "/api/v0.1/goods"
```

Definition at line 21 of file utils.py.

6.38.2.18 GOODS_URL

```
string core.utils.GOODS_URL = "http://localhost:5000/api/v0.1/goods"
```

Definition at line 22 of file utils.py.

6.38.2.19 LEDGER

```
string core.utils.LEDGER = "/api/v0.1/ledger"
```

Definition at line 17 of file utils.py.

6.38.2.20 LEDGER_URL

```
string core.utils.LEDGER_URL = "http://localhost:5000/api/v0.1/ledger"
```

Definition at line 32 of file utils.py.

6.38.2.21 log

```
core.utils.log = logging.getLogger()
```

Definition at line 55 of file utils.py.

6.38.2.22 MAX_BALANCE

```
int core.utils.MAX_BALANCE = MAX_COST*10
```

Definition at line 41 of file utils.py.

6.38.2.23 MAX_COST

```
int core.utils.MAX_COST = 1000000
```

Definition at line 13 of file utils.py.

6.38.2.24 MAX_CRED_ID

```
int core.utils.MAX_CRED_ID = 9223372036854775807
```

Definition at line 40 of file utils.py.

6.38.2.25 MAX_GOODS

```
int core.utils.MAX_GOODS = 100
```

Definition at line 14 of file utils.py.

6.38.2.26 PURCHASE

```
string core.utils.PURCHASE = "/api/v0.1/purchase"
```

Definition at line 19 of file utils.py.

6.38.2.27 PURCHASE_URL

```
string core.utils.PURCHASE_URL = "http://localhost:5000/api/v0.1/purchase"
```

Definition at line 33 of file utils.py.

6.38.2.28 QUALITY_REDUCTION

```
float core.utils.QUALITY_REDUCTION = 0.1
```

Definition at line 12 of file utils.py.

6.38.2.29 REGISTER

```
string core.utils.REGISTER = "/api/v0.1/register"
```

Definition at line 16 of file utils.py.

6.38.2.30 REGISTER_URL

```
string core.utils.REGISTER_URL = "http://localhost:5000/api/v0.1/register"
```

Definition at line 31 of file utils.py.

6.38.2.31 seed

```
core.utils.seed = int.from_bytes(os.urandom(2), 'big')
```

Definition at line 7 of file utils.py.

6.38.2.32 STOCHASTIC_TRADE_THRESHOLD

```
float core.utils.STOCHASTIC_TRADE_THRESHOLD = 0.9
```

Definition at line 42 of file utils.py.

6.38.2.33 TIMESTAMP_FORMAT

```
string core.utils.TIMESTAMP_FORMAT = "%Y-%m-%d %H:%M:%S.%f"
```

Definition at line 10 of file utils.py.

6.38.2.34 TRANSACTION

```
string core.utils.TRANSACTION = "/api/v0.1/transaction"
```

Definition at line 38 of file utils.py.

6.38.2.35 TRANSACTION_URL

```
string core.utils.TRANSACTION_URL = "http://localhost:5000/api/v0.1/transaction"
```

Definition at line 37 of file utils.py.

6.38.2.36 USD

```
string core.utils.USD = 'USD'
```

Definition at line 48 of file utils.py.

6.38.2.37 WEEKLY_LIMIT_EGP

```
int core.utils.WEEKLY_LIMIT_EGP = 50000
```

Definition at line 44 of file utils.py.

6.39 core_test Namespace Reference

Data Structures

- class Client
- class RestfulTest

Functions

- def get_bank_name ()
- def get_branch_number ()
- def get_account_number ()
- def get_name_reference ()
- def get_name ()
- def get_email ()
- def get_balance ()
- def get_credid ()
- def get_rand_pass (L=9)
- def rand_alphanum (L=9)
- def get_amount ()

Variables

- seed = int.from_bytes(os.urandom(3), 'big')
- faker = Faker(seed)
- string db_configs = "dbname='demo' user='tahweela' password='tahweela'"
- filename
- format
- filemode
- logger = logging.getLogger()

6.39.1 Function Documentation

6.39.1.1 get_account_number()

```
def core_test.get_account_number ( )
```

Definition at line 33 of file core_test.py.

```
33 def get_account_number():
34     return int(3333333*random.random())
```

6.39.1.2 get_amount()

```
def core_test.get_amount ( )
```

Definition at line 57 of file core_test.py.

```
57 def get_amount():
58     return 3333*random.random()
59
```

6.39.1.3 get_balance()

```
def core_test.get_balance ( )
```

Definition at line 41 of file core_test.py.

```
41 def get_balance():
42     return 333*random.random()
```

6.39.1.4 get_bank_name()

```
def core_test.get_bank_name ( )
```

Definition at line 29 of file core_test.py.

```
29 def get_bank_name():
30     return faker.name().split()[0]
```

6.39.1.5 get_branch_number()

```
def core_test.get_branch_number ( )
```

Definition at line 31 of file core_test.py.

```
31 def get_branch_number():
32     return int(33*random.random())
```

6.39.1.6 get_credid()

```
def core_test.get_credid ( )
```

Definition at line 43 of file core_test.py.

```
43 def get_credid():
44     return int(3333333333333*random.random())
```

6.39.1.7 get_email()

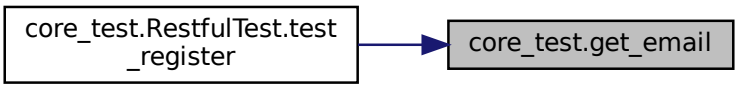
```
def core_test.get_email ( )
```

Definition at line 39 of file core_test.py.

```
39 def get_email():
40     return faker.email()
```

Referenced by core_test.RestfulTest.test_register().

Here is the caller graph for this function:



6.39.1.8 get_name()

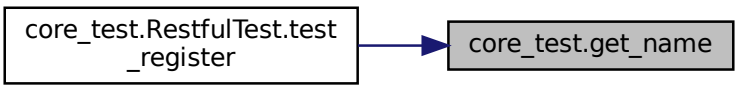
```
def core_test.get_name ( )
```

Definition at line 37 of file core_test.py.

```
37 def get_name():
38     return faker.name()
```

Referenced by core_test.RestfulTest.test_register().

Here is the caller graph for this function:



6.39.1.9 get_name_reference()

```
def core_test.get_name_reference ( )
```

Definition at line 35 of file core_test.py.

```
35 def get_name_reference():
36     return faker.name().split()[1]
```

6.39.1.10 get_rand_pass()

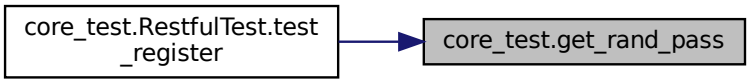
```
def core_test.get_rand_pass (
              L = 9 )
```

Definition at line 45 of file core_test.py.

```
45 def get_rand_pass(L=9):
46     passcode=''.join(random.choice(string.ascii_uppercase+\
47                                    string.ascii_lowercase+\
48                                    string.digits)\
49                      for _ in range(L))
50     return passcode
```

Referenced by core_test.RestfulTest.test_register().

Here is the caller graph for this function:



6.39.1.11 rand_alphanum()

```
def core_test.rand_alphanum (
              L = 9 )
```

Definition at line 51 of file core_test.py.

```
51 def rand_alphanum(L=9):
52     passcode=''.join(random.choice(string.ascii_uppercase+\
53                                    string.ascii_lowercase+\
54                                    string.digits)\
55                      for _ in range(L))
56     return passcode
```

6.39.2 Variable Documentation

6.39.2.1 db_configs

```
string core_test.db_configs = "dbname='demo' user='tahweela' password='tahweela'"
```

Definition at line 22 of file core_test.py.

6.39.2.2 faker

```
core_test.faker = Faker(seed)
```

Definition at line 21 of file core_test.py.

6.39.2.3 filemode

```
core_test.filemode
```

Definition at line 25 of file core_test.py.

6.39.2.4 filename

```
core_test.filename
```

Definition at line 23 of file core_test.py.

6.39.2.5 format

```
core_test.format
```

Definition at line 24 of file core_test.py.

Referenced by core_test.RestfulTest.__auth(), core_test.RestfulTest.autotract(), core_test.Client.basic_auth(), core_test.Client.get_balance(), core_test.Client.make_transaction(), core_test.Client.register(), and core_test.↩ RestfulTest.test_register().

6.39.2.6 logger

```
core_test.logger = logging.getLogger()
```

Definition at line 26 of file core_test.py.

6.39.2.7 seed

```
core_test.seed = int.from_bytes(os.urandom(3), 'big')
```

Definition at line 19 of file core_test.py.

6.40 database Namespace Reference

Data Structures

- class ServerDatabaseTest

Functions

- def get_bank_name ()
- def get_branch_number ()
- def get_account_number ()
- def get_name_reference ()
- def get_name ()
- def get_email ()
- def get_balance ()
- def get_credid ()
- def get_rand_pass (L=9)

Variables

- seed = int.from_bytes(os.urandom(2), "big")
- faker = Faker(seed)
- string db_configs = "dbname='demo' user='tahweela' password='tahweela'"
- db = database(db_configs)

6.40.1 Function Documentation

6.40.1.1 get_account_number()

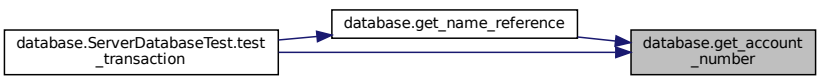
```
def database.get_account_number ( )
```

Definition at line 20 of file database.py.

```
20 def get_account_number():
21     return int(3333333*random.random())
```

Referenced by get_name_reference(), and database.ServerDatabaseTest.test_transaction().

Here is the caller graph for this function:



6.40.1.2 get_balance()

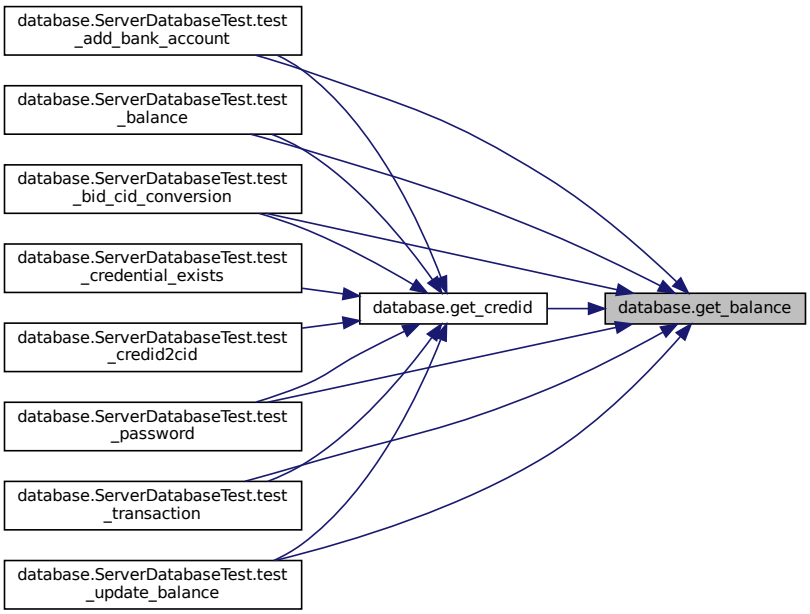
```
def database.get_balance ( )
```

Definition at line 34 of file database.py.

```
34 def get_balance():
35     return 333*random.random()
```

Referenced by get_credid(), database.ServerDatabaseTest.test_add_bank_account(), database.ServerDatabaseTest.test_balance(), database.ServerDatabaseTest.test_bid_cid_conversion(), database.ServerDatabaseTest.test_password(), database.ServerDatabaseTest.test_transaction(), and database.ServerDatabaseTest.test_update_balance().

Here is the caller graph for this function:



6.40.1.3 get_bank_name()

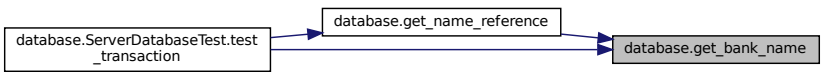
```
def database.get_bank_name ( )
```

Definition at line 16 of file database.py.

```
16 def get_bank_name():
17     return faker.name().split()[0]
```

Referenced by get_name_reference(), and database.ServerDatabaseTest.test_transaction().

Here is the caller graph for this function:



6.40.1.4 get_branch_number()

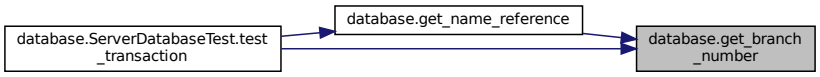
```
def database.get_branch_number ( )
```

Definition at line 18 of file database.py.

```
18 def get_branch_number():
19     return int(33*random.random())
```

Referenced by get_name_reference(), and database.ServerDatabaseTest.test_transaction().

Here is the caller graph for this function:



6.40.1.5 get_credid()

```
def database.get_credid ( )
```

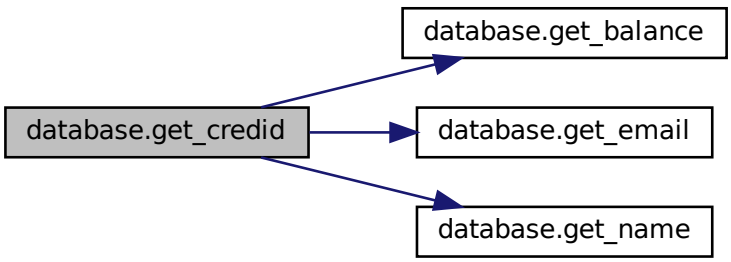
Definition at line 36 of file database.py.

```
36 def get_credid():
37     return int(3333333333333*random.random())
38
39 email=get_email()
40 name=get_name()
41 credid=get_credid()
42 balance=get_balance()
43 ADD_CURRENCY_LOCK=1
44 lock=ADD_CURRENCY_LOCK
45
```

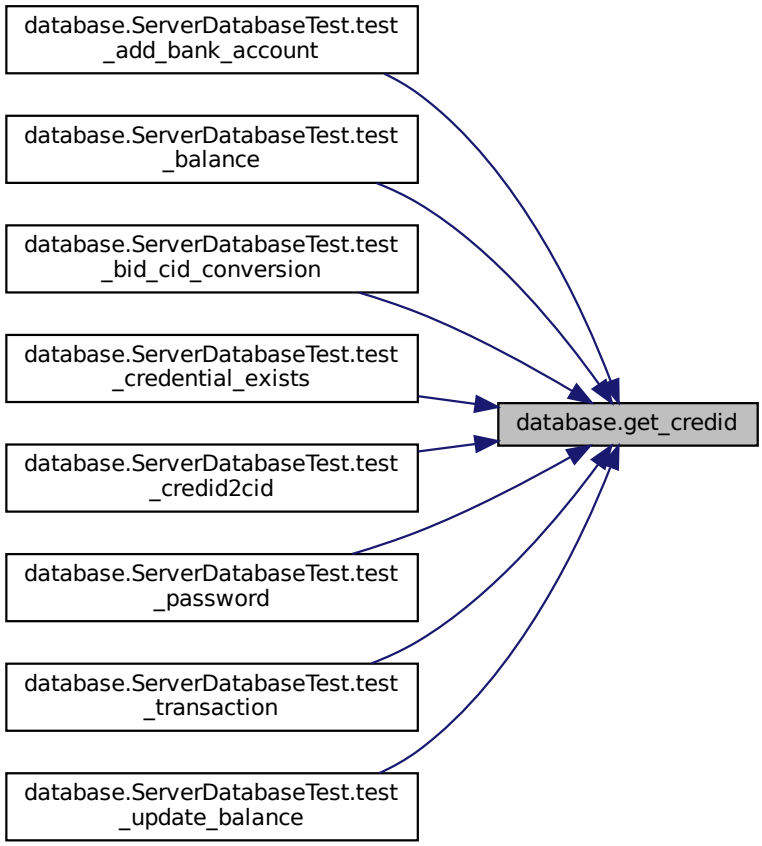
References get_balance(), get_email(), and get_name().

Referenced by database.ServerDatabaseTest.test_add_bank_account(), database.ServerDatabaseTest.test↩
_balance(), database.ServerDatabaseTest.test_bid_cid_conversion(), database.ServerDatabaseTest.test↩
_credential_exists(), database.ServerDatabaseTest.test_credid2cid(), database.ServerDatabaseTest.test_↩
password(), database.ServerDatabaseTest.test_transaction(), and database.ServerDatabaseTest.test_update↩
_balance().

Here is the call graph for this function:



Here is the caller graph for this function:



6.40.1.6 get_email()

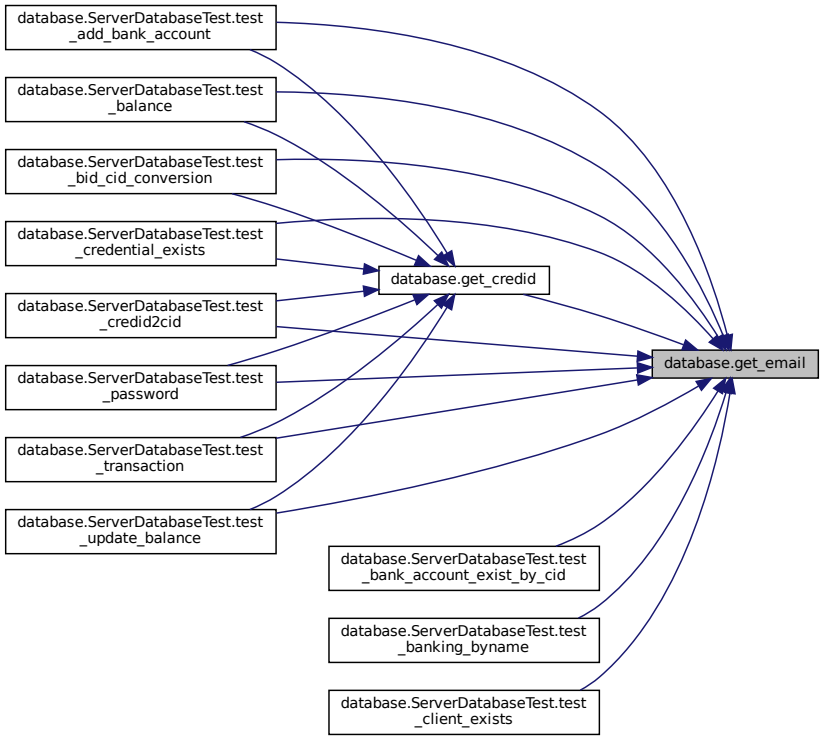
```
def database.get_email ( )
```

Definition at line 32 of file database.py.

```
32 def get_email():
33     return faker.email()
```

Referenced by get_credid(), database.ServerDatabaseTest.test_add_bank_account(), database.ServerDatabase↩ Test.test_balance(), database.ServerDatabaseTest.test_bank_account_exist_by_cid(), database.Server↩ DatabaseTest.test_banking_byname(), database.ServerDatabaseTest.test_bid_cid_conversion(), database.↩ ServerDatabaseTest.test_client_exists(), database.ServerDatabaseTest.test_credential_exists(), database.↩ ServerDatabaseTest.test_credid2cid(), database.ServerDatabaseTest.test_password(), database.Server↩ DatabaseTest.test_transaction(), and database.ServerDatabaseTest.test_update_balance().

Here is the caller graph for this function:



6.40.1.7 get_name()

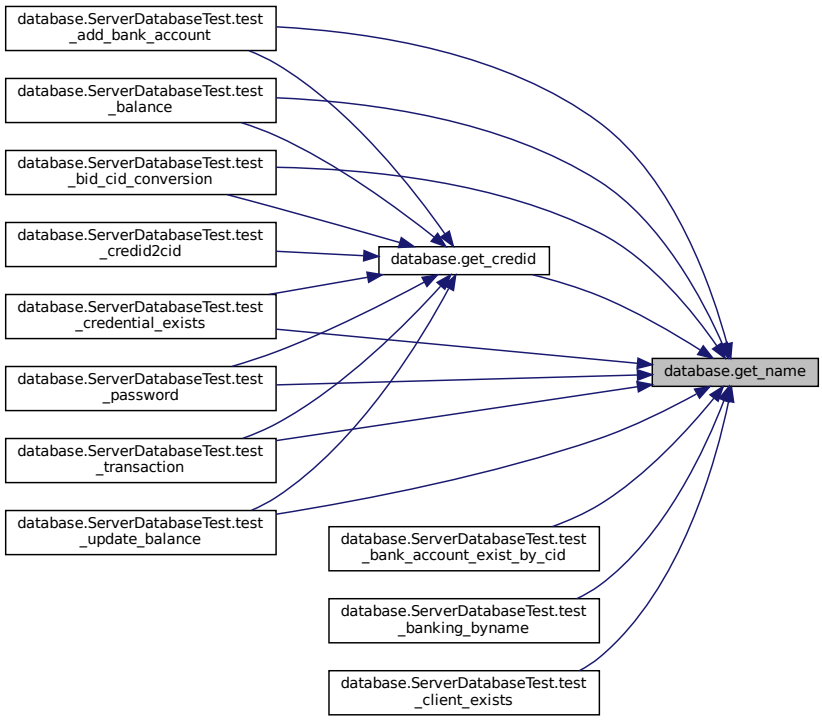
```
def database.get_name ( )
```

Definition at line 30 of file database.py.

```
30 def get_name():
31     return faker.name()
```

Referenced by get_credid(), database.ServerDatabaseTest.test_add_bank_account(), database.ServerDatabase↩ Test.test_balance(), database.ServerDatabaseTest.test_bank_account_exist_by_cid(), database.Server↩ DatabaseTest.test_banking_byname(), database.ServerDatabaseTest.test_bid_cid_conversion(), database.↩ ServerDatabaseTest.test_client_exists(), database.ServerDatabaseTest.test_credential_exists(), database.↩ ServerDatabaseTest.test_password(), database.ServerDatabaseTest.test_transaction(), and database.Server↩ DatabaseTest.test_update_balance().

Here is the caller graph for this function:



6.40.1.8 get_name_reference()

```
def database.get_name_reference ( )
```

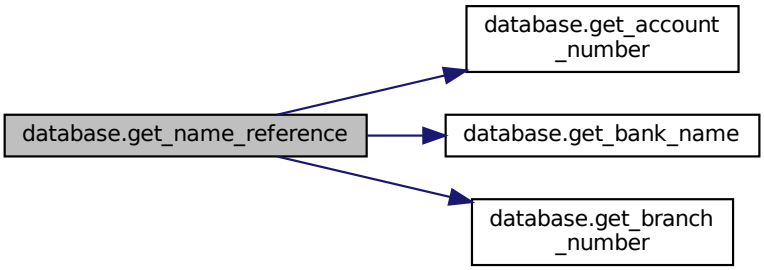
Definition at line 22 of file database.py.

```
22 def get_name_reference():
23     return faker.name().split()[1]
24
25 bank_name=get_bank_name()
26 branch_number=get_branch_number()
27 account_number=get_account_number()
28 name_reference=get_name_reference()
29
```

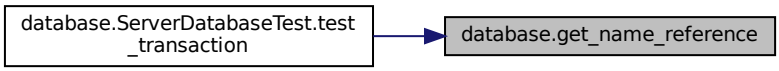
References get_account_number(), get_bank_name(), and get_branch_number().

Referenced by database.ServerDatabaseTest.test_transaction().

Here is the call graph for this function:



Here is the caller graph for this function:



6.40.1.9 get_rand_pass()

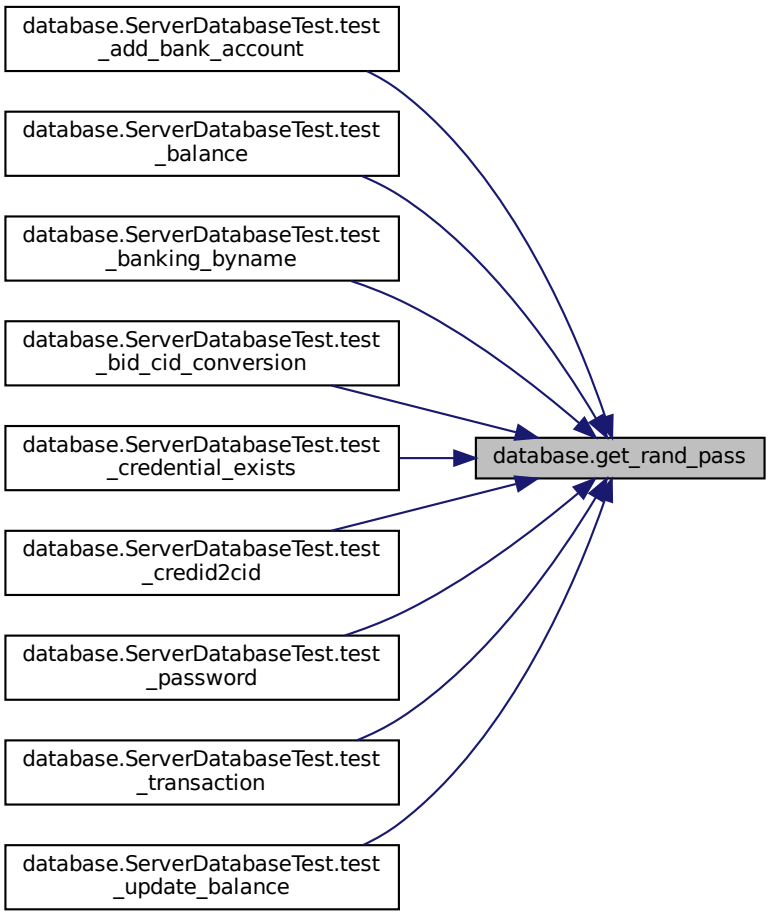
```
def database.get_rand_pass (
               L = 9 )
```

Definition at line 46 of file database.py.

```
46 def get_rand_pass(L=9):
47     passcode=''.join(random.choice(string.ascii_uppercase+\
48                                    string.ascii_lowercase+\
49                                    string.digits)\
50                      for _ in range(L))
51     return passcode
52
```

Referenced by database.ServerDatabaseTest.test_add_bank_account(), database.ServerDatabaseTest.test↩
_balance(), database.ServerDatabaseTest.test_banking_byname(), database.ServerDatabaseTest.test_bid_↩
cid_conversion(), database.ServerDatabaseTest.test_credential_exists(), database.ServerDatabaseTest.test_↩
credid2cid(), database.ServerDatabaseTest.test_password(), database.ServerDatabaseTest.test_transaction(), and database.ServerDatabaseTest.test_update_balance().

Here is the caller graph for this function:



6.40.2 Variable Documentation

6.40.2.1 db

```
database.db = database(db_configs)
```

Definition at line 14 of file database.py.

6.40.2.2 db_configs

```
string database.db_configs = "dbname='demo' user='tahweela' password='tahweela'"
```

Definition at line 13 of file database.py.

6.40.2.3 faker

```
database.faker = Faker(seed)
```

Definition at line 12 of file database.py.

6.40.2.4 seed

```
database.seed = int.from_bytes(os.urandom(2), "big")
```

Definition at line 10 of file database.py.

6.41 queries Namespace Reference

Namespaces

- database
- exists
- gets
- inserts
- updates

6.42 queries.database Namespace Reference

Data Structures

- class database
- class exists
- class gets
- class inserts
- class updates

6.43 queries.exists Namespace Reference

Namespaces

- banking
- clients
- contacts
- credentials
- goods
- owners

6.44 queries.exists.banking Namespace Reference

Functions

- def exists (cid)

6.44.1 Function Documentation

6.44.1.1 exists()

```
def queries.exists.banking.exists (
              cid )
```

verify that a banking account with the given client id is available (CALLED AT THE SERVER SIDE)

@param cid: client id
@return boolean wither the banking account for give client exists or note

Definition at line 4 of file banking.py.

```
4 def exists(cid):
5     """verify that a banking account with the given client id is available (CALLED AT THE SERVER SIDE)
6
7     @param cid: client id
8     @return boolean wither the banking account for give client exists or note
9     """
10     stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM banking WHERE client_id={cid});").\
11         format(cid=sql.Literal(cid))
12     cur.execute(stat)
13     return cur.fetchone()[0]
```

References client.client.format.

6.45 queries.exists.clients Namespace Reference

Functions

- def exists (cid)

6.45.1 Function Documentation

6.45.1.1 exists()

```
def queries.exists.clients.exists (
              cid )
```

verify that a client with given id is available (CALLED AT THE SERVER SIDE)

@param cid: client id
@return boolean wither the client exists or note

Definition at line 4 of file clients.py.

```
4 def exists(cid):
5     """verify that  a client with given id is available (CALLED AT THE SERVER SIDE)
6
7     @param cid: client id
8     @return boolean wither the client exists or note
9     """
10     stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM clients WHERE id={cid});").\
11         format(cid=sql.Literal(cid))
12     cur.execute(stat)
13     return cur.fetchone()[0]
```

References client.client.format.

6.46 queries.exists.contacts Namespace Reference

Functions

- def exists (cid)

6.46.1 Function Documentation

6.46.1.1 exists()

```
def queries.exists.contacts.exists (
              cid )
```

verify that a contact with given id is available (CALLED AT THE CLIENT SIDE)

@param cid: contact id
@return boolean wither the contact exists or note

Definition at line 4 of file contacts.py.

```
4 def exists(cid):
5     """verify that a contact with given id is available (CALLED AT THE CLIENT SIDE)
6
7     @param cid: contact id
8     @return boolean wither the contact exists or note
9     """
10     stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM contacts WHERE contact_id={cid});").\
11         format(cid=sql.Literal(cid))
12     cur.execute(stat)
13     return cur.fetchone()[0]
```

References client.client.format.

6.47 queries.exists.credentials Namespace Reference

Functions

- def exists (cid)

6.47.1 Function Documentation

6.47.1.1 exists()

```
def queries.exists.credentials.exists (
              cid )
```

```
verify the credential for the client with given cid(CALLED FROM SERVER SIDE),
or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)

@param cid: client id, or 1 (in case of call from client side for it's own credential)
@return boolean for wither the client (with given cid) is registered or not
```

Definition at line 4 of file credentials.py.

```
4 def exists(cid):
5     """ verify the credential for the client with given cid(CALLED FROM SERVER SIDE),
6     or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)
7
8     @param cid: client id, or 1 (in case of call from client side for it's own credential)
9     @return boolean for wither the client (with given cid) is registered or not
10     """
11     stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM credentials WHERE id={cid})").\
12         format(cid=sql.Literal(cid))
13     cur.execute(stat)
14     return cur.fetchone()[0]
15
16 "'
17 def exists(cred_id, passcode):
18     """ verify the credential for the client with given cid(CALLED FROM SERVER SIDE),
19     or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)
20
21     @param cid: client id, or 1 (in case of call from client side for it's own credential)
22     @return boolean for wither the client (with given cid) is registered or not
23     """
24     stat="SELECT EXISTS (SELECT 1 FROM credentials WHERE cred_id={} AND passcode={})".\
25         format(cred, passcode)
26     cur.execute(cid)
27     return cur.fetchone()[0]
28 "'
```

References client.client.format.

6.48 queries.exists.goods Namespace Reference

Functions

- def exists (gid)

6.48.1 Function Documentation

6.48.1.1 exists()

```
def queries.exists.goods.exists (
              gid )
```

verify that a good with given id is available

@param gid: good id
@return boolean wither the good exists or note

Definition at line 4 of file goods.py.

```
4 def exists(gid):
5     """verify that a good with given id is available
6
7     @param gid: good id
8     @return boolean wither the good exists or note
9     """
10     stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM goods WHERE id={gid});").format(gid=sql.Literal(gid))
11     cur.execute(stat)
12     return cur.fetchone()[0]
```

References client.client.format.

6.49 queries.exists.owners Namespace Reference

Functions

- def exists ()

6.49.1 Function Documentation

6.49.1.1 exists()

```
def queries.exists.owners.exists ( )
```

Definition at line 3 of file owners.py.

```
3 def exists():
4     pass
```

6.50 queries.gets Namespace Reference

Namespaces

- banking
- clients
- contacts
- credentials
- currency
- goods
- ledger
- owner

6.51 queries.gets.banking Namespace Reference

Functions

- def get_client_id (bid)
- def get_banking_id (cid)
- def get_balance_by_cid (cid)
- def get_balance_by_credid (cred_id)

6.51.1 Function Documentation

6.51.1.1 get_balance_by_cid()

```
def queries.gets.banking.get_balance_by_cid (
              cid )
```

```
called at the server side to retrieve the account balance d of the given client_id (cid)

@param cid: client id
@return bid: banking id
```

Definition at line 26 of file banking.py.

```
26 def get_balance_by_cid(cid):
27     """called at the server side to retrieve the account balance d of the given client_id (cid)
28 
29     @param cid: client id
30     @return bid: banking id
31     """
32     query=sql.SQL("SELECT (balance) FROM banking WHERE client_id={cid} LIMIT
       1").format(cid=sql.Literal(cid))
33     db_log.debug(query)
34     return pd.read_sql(query, conn).ix[0]
35 
```

References client.client.format.

6.51.1.2 get_balance_by_credid()

```
def queries.gets.banking.get_balance_by_credid (
              cred_id )

get balance of client with given credential id

@param cred_id: client credential id
```

Definition at line 36 of file banking.py.

```
36 def get_balance_by_credid(cred_id):
37     """ get balance of client with given credential id
38
39     @param cred_id: client credential id
40     """
41     query=sql.SQL("SELECT (b.balance) FROM banking as b JOIN WITH credentials AS c WHERE
       c.cred_id={credid} AND c.id==b.client_id;").format(credid=sql.Literal(cred_id))
42     db_log.debug(query)
43     cur.execute(query)
44     return cur.fetchone()[0]
```

References client.client.format.

6.51.1.3 get_banking_id()

```
def queries.gets.banking.get_banking_id (
              cid )

retrieve the corresponding banking_id of the given  client_id (cid) (called at the server side)

@param cid: client id
@return bid: banking id
```

Definition at line 16 of file banking.py.

```
16 def get_banking_id(cid):
17     """retrieve the corresponding banking_id of the given  client_id (cid) (called at the server side)
18
19     @param cid: client id
20     @return bid: banking id
21     """
22     query=sql.SQL("SELECT (id) FROM banking WHERE client_id={cid} LIMIT 1").format(cid=sql.Literal(cid))
23     db_log.debug(query)
24     return pd.read_sql(query, conn).ix[0]
25
```

References client.client.format.

6.51.1.4 get_client_id()

```
def queries.gets.banking.get_client_id (
              bid )

retrieve the corresponding client_id of the given banking_id (bid) (called at the server side)

@param bid: banking id
@return cid: contact id
```

Definition at line 6 of file banking.py.

```
6 def get_client_id(bid):
7     """ retrieve the corresponding client_id of the given banking_id (bid) (called at the server side)
8
9     @param bid: banking id
10     @return cid: contact id
11     """
12     query=sql.SQL("SELECT (client_id) FROM banking WHERE id={bid} LIMIT 1").format(bid=sql.Literal(bid))
13     db_log.debug(query)
14     return pd.read_sql(query, conn).ix[0]
15
```

References client.client.format.

6.52 queries.gets.clients Namespace Reference

Functions

- def get_all ()
- def get (cid)
- def get_name (cid)

6.52.1 Function Documentation

6.52.1.1 get()

```
def queries.gets.clients.get (
              cid )
```

```
retrieve client into with given client id (cid)

@param cid: client id
@return tuple (id, name, join date)
```

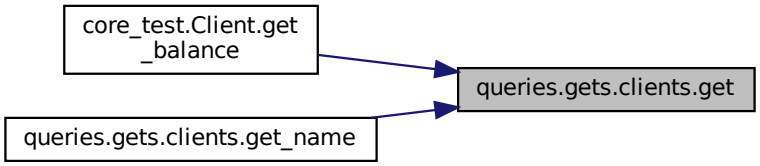
Definition at line 15 of file clients.py.

```
15 def get(cid):
16     """retrieve client into with given client id (cid)
17 
18     @param cid: client id
19     @return tuple (id, name, join date)
20     """
21     query=sql.SQL("SELECT (id, contact_name, client_join_dt) FROM clients WHERE
       id={cid};").format(cid=sql.Literal(cid))
22     db_log.debug(query)
23     cur.execute(query)
24     return cur.fetchone()
25 
```

References client.client.format.

Referenced by core_test.Client.get_balance(), and queries.gets.clients.get_name().

Here is the caller graph for this function:



6.52.1.2 get_all()

```
def queries.gets.clients.get_all ( )
```

retrieve all clients info

Definition at line 7 of file clients.py.

```
7 def get_all():
8     """retrieve all clients info
9
10     """
11     query="SELECT * FROM clients;"
12     db_log.debug(query)
13     return pd.read_sql(query, conn)
14
```

6.52.1.3 get_name()

```
def queries.gets.clients.get_name (
               cid )
```

retrieve client name corresponding to given client id (cid)

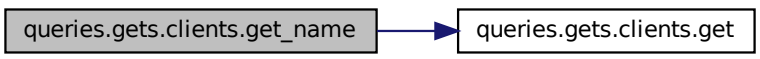
@param cid: client id
@return client name

Definition at line 26 of file clients.py.

```
26 def get_name(cid):
27     """retrieve client name corresponding to given client id (cid)
28
29     @param cid: client id
30     @return client name
31     """
32     return get(cid)[1]
```

References queries.gets.clients.get().

Here is the call graph for this function:



6.53 queries.gets.contacts Namespace Reference

Functions

- def get_all ()
- def get_banking_id (cid)

6.53.1 Function Documentation

6.53.1.1 get_all()

```
def queries.gets.contacts.get_all ( )
```

Definition at line 6 of file contacts.py.

```
6 def get_all():
7     query = "SELECT * FROM contacts;"
8     db_log.debug(query)
9     return pd.read_sql(query, conn)
10
```

6.53.1.2 get_banking_id()

```
def queries.gets.contacts.get_banking_id (
              cid )
```

```
called at the client side, to retrieve the stored banking id in the contacts

@param cid: contact id
@return banking_id or the associated banking id for the given contact id
```

Definition at line 11 of file contacts.py.

```
11 def get_banking_id(cid):
12     """ called at the client side, to retrieve the stored banking id in the contacts
13
14     @param cid: contact id
15     @return banking_id or the associated banking id for the given contact id
16     """
17     query=sql.SQL("SELECT (bank_account_id) FROM contacts WHERE contact_id='{cid}' LIMIT 1;").\
18         format(cid=sql.Literal(cid))
19     db_log.debug(query)
20     return pd.read_sql(query, conn).ix[0]
```

References client.client.format.

6.54 queries.gets.credentials Namespace Reference

Functions

- def get_all ()
- def get_credential (cid)
- def get_id (cred_id)
- def get_password (cred_id)
- def get_credid_with_gid (gid)

6.54.1 Function Documentation

6.54.1.1 get_all()

```
def queries.gets.credentials.get_all ( )
```

Definition at line 6 of file credentials.py.

```
6 def get_all():
7     query = "SELECT * FROM credentials;"
8     db_log.debug(query)
9     ret = pd.read_sql(conn, query)
10     return ret
11
```

6.54.1.2 get_credential()

```
def queries.gets.credentials.get_credential (
              cid )
```

```
get the credential for the client with given cid(CALLED FROM SERVER SIDE),
or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)

@param cid: client id, or 1 (in case of call from client side for it's own credential)
```

Definition at line 12 of file credentials.py.

```
12 def get_credential(cid):
13     """ get the credential for the client with given cid(CALLED FROM SERVER SIDE),
14     or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)
15
16     @param cid: client id, or 1 (in case of call from client side for it's own credential)
17     """
18     query=sql.SQL("SELECT * FROM credentials WHERE id={cid} LIMIT 1;").\
19         format(cid=sql.Literal(cid))
20     db_log.debug(query)
21     ret = pd.read_sql(conn, query)
22
```

References client.client.format.

6.54.1.3 get_credid_with_gid()

```
def queries.gets.credentials.get_credid_with_gid (
              gid )
```

```
cross reference credential id, with good's id

@param gid: good's id
@return credential id credid
```

Definition at line 47 of file credentials.py.

```
47 def get_credid_with_gid(gid):
48     """cross reference credential id, with good's id
49
50     @param gid: good's id
51     @return credential id credid
52     """
53     query=sql.SQL("SELECT (C.cred_id) FROM credentials as c JOIN WITH goods AS g JOIN WITH owners as o
       WHERE g.id={gid} AND o.owner_id==c.id LIMIT 1;").\
54         format(gid=sql.Literal(gid))
55     db_log.debug(query)
56     cur.execute(query)
57     return cur.fetchone()[0]
```

References client.client.format.

6.54.1.4 get_id()

```
def queries.gets.credentials.get_id (
              cred_id )
```

get client id

@param cred_id: credential id
@return the id, or None if doesn't exist

Definition at line 23 of file credentials.py.

```
23 def get_id(cred_id):
24     """ get client id
25 
26     @param cred_id: credential id
27     @return the id, or None if doesn't exist
28     """
29     query=sql.SQL("SELECT id FROM credentials WHERE cred_id={credid} LIMIT 1;").\
30         format(credid=sql.Literal(cred_id))
31     db_log.debug(query)
32     cur.execute(query)
33     return cur.fetchone()[0]
34 
```

References client.client.format.

6.54.1.5 get_password()

```
def queries.gets.credentials.get_password (
              cred_id )
```

get user's passcode for authentication

@param cred_id: credential id
@return list of the id, or empty list of doesn't exist

Definition at line 35 of file credentials.py.

```
35 def get_password(cred_id):
36     """ get user's passcode for authentication
37 
38     @param cred_id: credential id
39     @return list of the id, or empty list of doesn't exist
40     """
41     query=sql.SQL("SELECT (passcode) FROM credentials WHERE cred_id={credid} LIMIT 1;").\
42         format(credid=sql.Literal(cred_id))
43     db_log.debug(query)
44     cur.execute(query)
45     return cur.fetchone()[0]
46 
```

References client.client.format.

6.55 queries.gets.currency Namespace Reference

Functions

- def to_dollar (cid)

6.55.1 Function Documentation

6.55.1.1 to_dollar()

```
def queries.gets.currency.to_dollar (
              cid )

convert currency of the corresponding id to dollar ratio

for example if currency A = 2 dollars, then the conversion would be 0.5,
for another currency B = 0.5 dollar, then the conversion to dollar would be 2
such that for given cost of xA, would be 0.5x$.
@param cid is the id of the corresponding currency
@return transformation ratio to dollar
```

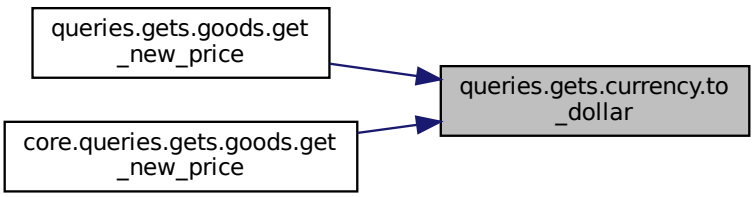
Definition at line 6 of file currency.py.

```
6 def to_dollar(cid):
7     """ convert currency of the corresponding id to dollar ratio
8
9     for example if currency A = 2 dollars, then the conversion would be 0.5,
10     for another currency B = 0.5 dollar, then the conversion to dollar would be 2
11     such that for given cost of xA, would be 0.5x$.
12     @param cid is the id of the corresponding currency
13     @return transformation ratio to dollar
14     """
15     query = sql.SQL("SELECT * FROM currency WHERE id=cid;").\
16         format(cid=sql.Literal(cid))
17     db_log.debug(query)
18     ratio = 1.0/pd.read_sql(query, conn)['currency_value'].ix[0]
19     return ratio
```

References client.client.format.

Referenced by queries.gets.goods.get_new_price(), and core.queries.gets.goods.get_new_price().

Here is the caller graph for this function:



6.56 queries.gets.goods Namespace Reference

Functions

- def get_all ()
- def get_good (gid)
- def get_commodity (gname, quality=0)
- def get_new_price (gid)

6.56.1 Function Documentation

6.56.1.1 get_all()

```
def queries.gets.goods.get_all ( )
```

Definition at line 6 of file goods.py.

```
6 def get_all():
7     query="SELECT * FROM goods;"
8     #return pd.read_sql(query, conn, index_col='id').to_json()
9     return pd.read_sql(query, conn).to_json()
10
```

6.56.1.2 get_commodity()

```
def queries.gets.goods.get_commodity (
              gname,
              quality = 0 )
```

```
retrive good for the given goods constraints

@param gname: goods name
@param quality: retrieve goods with quality > given threshold
@return pandas data frame of the corresponding constrains
```

Definition at line 22 of file goods.py.

```
22 def get_commodity(gname, quality=0):
23     """retrive good for the given goods constraints
24
25     @param gname: goods name
26     @param quality: retrieve goods with quality > given threshold
27     @return pandas data frame of the corresponding constrains
28     """
29     query = sql.SQL("SELECT * FROM goods WHERE good_name={gname} AND good_quality>={gquality}").\
30         format(gname=sql.Literal(gname), \
31                quality=sql.Literal(gquality))
32     db_log.debug(query)
33     return pd.read_sql(query, conn)
34
```

References client.client.format.

6.56.1.3 get_good()

```
def queries.gets.goods.get_good (
               gid )
```

```
retrive good for the given goods id (gid)

@param gid: goods id
@return pandas data series of the corresponding row
```

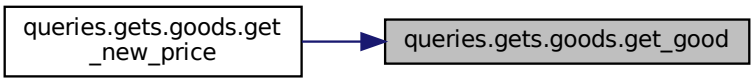
Definition at line 11 of file goods.py.

```
11 def get_good(gid):
12     """retrive good for the given goods id (gid)
13 
14     @param gid: goods id
15     @return pandas data series of the corresponding row
16     """
17     query = sql.SQL("SELECT * FROM goods WHERE id={gid};").\
18         format(gid=sql.Literal(gid))
19     db_log.debug(query)
20     return pd.read_sql(query, conn)
21 
```

References client.client.format.

Referenced by queries.gets.goods.get_new_price().

Here is the caller graph for this function:



6.56.1.4 get_new_price()

```
def queries.gets.goods.get_new_price (
               gid )
```

```
get good price with given good's id

@param gid: good's id
@return price in dollar
```

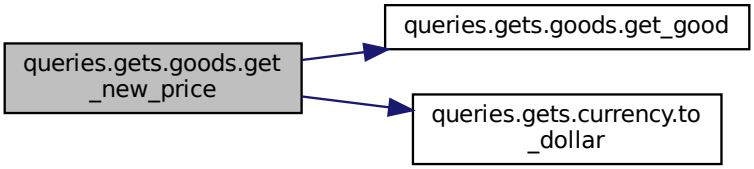
Definition at line 35 of file goods.py.

```
35 def get_new_price(gid):
36     """ get good price with given good's id
37 
38     @param gid: good's id
39     @return price in dollar
40     """
41     df = get_good(gid)
42     cur_id = df['good_currency_id'].ix[0]
```

```
43     return df['good_cost'].ix[0]*to_dollar(cur_id)
```

References queries.gets.goods.get_good(), and queries.gets.currency.to_dollar().

Here is the call graph for this function:



6.57 queries.gets.ledger Namespace Reference

Functions

- def get_transactions (st_dt, end_dt=dt.datetime.now())
- def get_sells (dest, st_dt, end_dt=None)
- def get_last_timestamp ()

6.57.1 Function Documentation

6.57.1.1 get_last_timestamp()

```
def queries.gets.ledger.get_last_timestamp ( )
```

```
retrieve the timestamp of the last transaction (CALLED FROM THE CLIENT SIDE)

@return timestamp
```

Definition at line 29 of file ledger.py.

```
29 def get_last_timestamp():
30     """ retrieve the timestamp of the last transaction (CALLED FROM THE CLIENT SIDE)
31
32     @return timestamp
33     """
34     query="SELECT currval(pg_get_serial_sequence('ledger', 'trx_id'));"
35     db_log.debug(query)
36     cur.execute(query)
37     return cur.fetchone()[0]
```

6.57.1.2 get_sells()

```
def queries.gets.ledger.get_sells (
              dest,
              st_dt,
              end_dt = None )
```

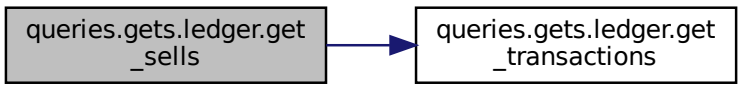
```
get sells transaction within the st_dt, end_dt period, while there destined to dest (CALLED AT SERVER SIDE)
@param dest: the destination credential id
@return sells transactions
```

Definition at line 20 of file ledger.py.

```
20 def get_sells(dest, st_dt, end_dt=None):
21     """ get sells transaction within the st_dt, end_dt period, while there destined to dest (CALLED AT
       SERVER SIDE)
22     @param dest: the destination credential id
23     @return sells transactions
24 """
25     trx=get_transactions(st_dt, end_dt).to_json()
26     trx.apply(lambda x:x['trx_dest']==dest, inplace=True)
27     return trx
28
```

References queries.gets.ledger.get_transactions().

Here is the call graph for this function:



6.57.1.3 get_transactions()

```
def queries.gets.ledger.get_transactions (
              st_dt,
              end_dt = dt.datetime.now() )
```

```
get the transactions within the given period exclusively

@param st_dt: the start datetime
@param end_dt: the end datetime
@return dataframe of the transactions
```

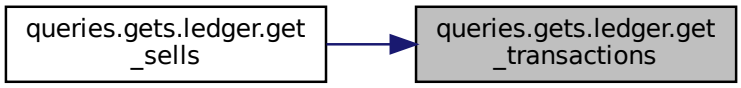
Definition at line 7 of file ledger.py.

```
7 def get_transactions(st_dt, end_dt=dt.datetime.now()):
8     """ get the transactions within the given period exclusively
9
10     @param st_dt: the start datetime
11     @param end_dt: the end datetime
12     @return dataframe of the transactions
13     """
14     stat = sql.SQL("SELECT * FROM ledger WHERE trx_dt>{st_dt} AND trx_dt<{end_dt};").\
15         format(st_dt=sql.Literal(st_dt), \
16                end_dt=sql.Literal(end_dt))
17     db_log.debug(stat)
18     return pd.read_sql(conn, stat)
19
```

References client.client.format.

Referenced by queries.gets.ledger.get_sells().

Here is the caller graph for this function:



6.58 queries.gets.owner Namespace Reference

Functions

- def get_all ()
- def get_good_owner (gid)
- def get_owner_goods (oid)

6.58.1 Function Documentation

6.58.1.1 get_all()

```
def queries.gets.owner.get_all ( )
```

Definition at line 6 of file owner.py.

```
6 def get_all():
7     query="SELECT * FROM owner;"
8     return pd.read_sql(query, conn, index_col='id')
9
```

6.58.1.2 get_good_owner()

```
def queries.gets.owner.get_good_owner (
              gid )

return owner id (oid) for the given gid

@param gid: good
@return the owner id
```

Definition at line 10 of file owner.py.

```
10 def get_good_owner(gid):
11     """return owner id (oid) for the given gid
12 
13     @param gid: good
14     @return the owner id
15     """
16     query = sql.SQL("SELECT (owner_id) FROM owners WHERE good_id={gid}").\
17         format(gid=sql.Literal(gid))
18     db_log.debug(query)
19     return pd.read_sql(query, conn).ix[0]
20 
```

References client.client.format.

6.58.1.3 get_owner_goods()

```
def queries.gets.owner.get_owner_goods (
              oid )

return the good assigned to the given owner id (oid)

@param oid: is the owner id
@return json dict of good's ids
```

Definition at line 21 of file owner.py.

```
21 def get_owner_goods(oid):
22     """return the good assigned to the given owner id (oid)
23 
24     @param oid: is the owner id
25     @return json dict of good's ids
26     """
27     query = sql.SQL("SELECT (good_id) FROM owners WHERE owner_id={oid}").\
28         format(oid=sql.Literal(oid))
29     db_log.debug(query)
30     return pd.read_sql(query, conn).to_json()
```

References client.client.format.

6.59 queries.inserts Namespace Reference

Namespaces

- banking
- clients
- contacts
- credentials
- goods
- ledger
- owners

6.60 queries.inserts.banking Namespace Reference

Functions

- def insert_banking (cid, balance)

6.60.1 Function Documentation

6.60.1.1 insert_banking()

```
def queries.inserts.banking.insert_banking (
              cid,
              balance )
```

```
give the client with the given id (cid) banking account (CALLED AT SERVER SIDE)

@param cid: client id
@param balance: client account balance
```

Definition at line 6 of file banking.py.

```
6 def insert_banking(cid, balance):
7     """ give the client with the given id (cid) banking account (CALLED AT SERVER SIDE)
8
9     @param cid: client id
10     @param balance: client account balance
11     """
12     stat=sql.SQL("INSERT INTO banking (client_id, balance, balance_dt) VALUES ({cid}, {balance},
       {dt});"). \
13         format(cid=sql.Literal(cid), \
14                balance=sql.Literal(balance), \
15                dt=sql.Literal(dt.datetime.now().strftime(TIMESTAMP_FORMAT)))
16     db_log.debug(stat)
17     cur.execute(stat)
18     stat="SELECT currval(pg_get_serial_sequence('banking', 'id'));"
19     db_log.debug(stat)
20     cur.execute(stat);
21     return cur.fetchone()[0]
```

References client.client.format.

6.61 queries.inserts.clients Namespace Reference

Functions

- def insert_client (name)

6.61.1 Function Documentation

6.61.1.1 insert_client()

```
def queries.inserts.clients.insert_client (
              name )

add new client to the network (CALLED AT THE SERVER SIDE),

note that some clients might not have banking id yet
@param name: client name
```

Definition at line 4 of file clients.py.

```
4 def insert_client(name):
5     """ add new client to the network (CALLED AT THE SERVER SIDE),
6
7     note that some clients might not have banking id yet
8     @param name: client name
9     """
10     stat=sql.SQL("INSERT INTO clients (contact_name) VALUES ({name})").\
11         format(name=sql.Literal(name))
12     cur.execute(stat)
13     cur.execute("SELECT currval(pg_get_serial_sequence('clients', 'id'));")
14     return cur.fetchone()[0]
```

References client.client.format.

6.62 queries.inserts.contacts Namespace Reference

Functions

- def insert_contact (cid, cname, bid)

6.62.1 Function Documentation

6.62.1.1 insert_contact()

```
def queries.inserts.contacts.insert_contact (
              cid,
              cname,
              bid )

insert new contact (CALLED AT THE CLIENT SIDE)

@param cid: contact id (the same as client id in the server side)
@param cname: contact name
@param bid: bank account id
```

Definition at line 5 of file contacts.py.

```
5 def insert_contact(cid, cname, bid):
6     """ insert new contact (CALLED AT THE CLIENT SIDE)
7
8     @param cid: contact id (the same as client id in the server side)
9     @param cname: contact name
10     @param bid: bank account id
11     """
12     stat=sql.SQL("INSERT INTO contacts (contact_id, contact_name bank_account_id) VALUES ({cid}, {cname},
       {bid})").\
13         format(cid=sql.Literal(cid), \
14                cname=sql.Literal(cname), \
15                bid=sql.Literal(bid))
16     db_log.debug(stat)
17     cur.execute(stat)
```

References client.client.format.

6.63 queries.inserts.credentials Namespace Reference

Functions

- def new_cred (passcode, cred_id)
- def register (cid)

6.63.1 Function Documentation

6.63.1.1 new_cred()

```
def queries.inserts.credentials.new_cred (
              passcode,
              cred_id )
```

```
add client credentials returned from the server

@param cid: client id
```

Definition at line 9 of file credentials.py.

```
9 def new_cred(passcode, cred_id):
10     """add client credentials returned from the server
11
12     @param cid: client id
13     """
14     stat=sql.SQL("INSERT INTO credentials (passcode, cred_id) VALUES ({passcode}, {credid});").\
15         format(passcode=sql.Literal(passcode), \
16                credid=sql.Literal(cred_id))
17     db_log.debug(stat)
18     cur.execute(stat)
19
```

References client.client.format.

6.63.1.2 register()

```
def queries.inserts.credentials.register (
              cid )
```

```
register new client credentials with given cid (CALLED FROM SERVER SIDE)

@param cid: client id
@return a tuple (cred_id, passcode)
```

Definition at line 20 of file credentials.py.

```
20 def register(cid):
21     """register new client credentials with given cid (CALLED FROM SERVER SIDE)
22 
23     @param cid: client id
24     @return a tuple (cred_id, passcode)
25     """
26     cred_id=rand.random()*MAX_CRED_ID
27     passcode=''.join(rand.choice(string.ascii_uppercase+\
28                                  string.ascii_lowercase+string.digits)\
29                      for _ in range(9))
30     stat=sql.SQL("INSERT INTO credentials (id, passcode, cred_id) VALUES ({cid}, {passcode},
       {credid});").\
31         format(cid=sql.Literal(cid), \
32                passcode=sql.Literal(passcode), \
33                credid=sql.Literal(cred_id))
34     db_log.debug(stat)
35     cur.execute(stat)
36     return (cred_id, passcode)
37 
38 '''
39 def add_cred(passcode, cred_id):
40     """add client credentials returned from the server(CALLED FROM SERVER SIDE)
41 
42     @param cid: client id
43     """
44     stat=sql.SQL("INSERT INTO credentials (passcode, cred_id) VALUES ({passcode}, {credid});").\
45         format(passcode=sql.Literal(passcode), \
46                credid=sql.Literal(cred_id))
47     db_log.debug(stat)
48     cur.execute(stat)
49 '''
```

References client.client.format.

6.64 queries.inserts.goods Namespace Reference

Functions

- def add_good (gname, gquality, gcost, gcid=1)

6.64.1 Function Documentation

6.64.1.1 add_good()

```
def queries.inserts.goods.add_good (
              gname,
              gquality,
              gcost,
              gcid = 1 )
```

INSERT new good into the goods table

@param gname: good name
@param gquality: good quality
@param gcost: good cost
@param gcid: good currency id

Definition at line 5 of file goods.py.

```
5 def add_good(gname, gquality, gcost, gcid=1):
6     """ INSERT new good into the goods table
7
8     @param gname: good name
9     @param gquality: good quality
10     @param gcost: good cost
11     @param gcid: good currency id
12     """
13     stat=sql.SQL("INSERT INTO goods (good_name, good_quality, good_cost, good_currency_id) VALUES
       ({gname}, {gquality}, {gcost}, {gcid});").\
14         format(gname=sql.Literal(gname), \
15                gquality=sql.Literal(gquality), \
16                gcost=sql.Literal(gcost), \
17                gcid=sql.Literal(gcid))
18     db_log.debug(stat)
19     cur.execute(stat)
20     stat="SELECT currval(pg_get_serial_sequence('goods', 'id'));"
21     cur.execute(stat)
22     db_log.debug(stat)
23     return cur.fetchone()[0]
```

References client.client.format.

6.65 queries.inserts.ledger Namespace Reference

Functions

- def insert_trx (des, src, gid)

6.65.1 Function Documentation

6.65.1.1 insert_trx()

```
def queries.inserts.ledger.insert_trx (
              des,
              src,
              gid )
```

```
insert transaction from 'src' to 'des' for good with 'gid'

@param des: the transaction destination
@param src: the transaction source
@param gid: the good's id
```

Definition at line 5 of file ledger.py.

```
5 def insert_trx(des, src, gid):
6     """ insert transaction from 'src' to 'des' for good with 'gid'
7
8     @param des: the transaction destination
9     @param src: the transaction source
10     @param gid: the good's id
11     """
12     stat=sql.SQL("INSERT INTO ledger (trx_dest, trx_src, good_id) VALUES ({des}, {src}, {gid});").\
13         format(des=sql.Literal(des), \
14                src=sql.Literal(src), \
15                gid=sql.Literal(gid))
16     db_log.debug(stat)
17     cur.execute(stat)
```

References client.client.format.

6.66 queries.inserts.owners Namespace Reference

Functions

- def add_owner (oid, gid)

6.66.1 Function Documentation

6.66.1.1 add_owner()

```
def queries.inserts.owners.add_owner (
              oid,
              gid )
```

```
assign ownership of owner with id (oid) to the good with id (gid)

@param oid: owner id
@param gid: good id
```

Definition at line 5 of file owners.py.

```
5 def add_owner(oid, gid):
6     """assign ownership of owner with id (oid) to the good with id (gid)
7
8     @param oid: owner id
9     @param gid: good id
10     """
11     stat=sql.SQL("INSERT INTO owners (owner_id, good_id) VALUES ({oid}, {gid})").\
12         format(oid=sql.Literal(oid), \
13                gid=sql.Literal(gid))
14     db.log(stat)
15     cur.execute(stat)
```

References client.client.format.

6.67 queries.updates Namespace Reference

Namespaces

- banking
- owners

6.68 queries.updates.banking Namespace Reference

Functions

- def update_account (cid, balance)

6.68.1 Function Documentation

6.68.1.1 update_account()

```
def queries.updates.banking.update_account (
              cid,
              balance )

update the banking account with the calculated new balance (CALLED FROM SERVER SIDE)

@param cid: client id
@param balance: the account balance
```

Definition at line 6 of file banking.py.

```
6 def update_account(cid, balance):
7     """update the banking account with the calculated new balance (CALLED FROM SERVER SIDE)
8
9     @param cid: client id
10     @param balance: the account balance
11     """
12     stat = sql.SQL("UPDATE banking SET (balance, balance_dt) = ({balance}, {dt}) WHERE
       client_id={cid}").\
13         format(balance=sql.Literal(balance), \
14                dt=dt.datetime().strftime(TIMESTAMP_FORMAT), \
15                cid=sql.Literal(cid))
16     cur.execute(stat)
```

References client.client.format.

6.69 queries.updates.owners Namespace Reference

Functions

- def update_owner (oid, gid)

6.69.1 Function Documentation

6.69.1.1 update_owner()

```
def queries.updates.owners.update_owner (
              oid,
              gid )

reassign the good's ownership with corresponding gid

@param gid: good id
```

Definition at line 4 of file owners.py.

```
4 def update_owner(oid, gid):
5     """reassign the good's ownership with corresponding gid
6
7     @param gid: good id
8     """
9     stat = sql.SQL("UPDATE owners SET (owner_id) = {oid} WHERE good_id={gid}").\
10         format(oid=sql.Literal(oid), \
11                bid=sql.Literal(gid))
12     cur.execute(stat)
```

References client.client.format.

6.70 server Namespace Reference

Namespaces

- server

6.71 server.server Namespace Reference

Functions

- def get_credid ()
- def is_email (email)
- def authenticate (user, passcode)
- def unauthorized ()
- def register_client ()
- def add_bank_account ()
- def add_contact ()
- def get_balance ()
- def update_ledger ()
- def make_transaction ()

Variables

- seed = int.from_bytes(os.urandom(2), 'big')
- string db_configs = "dbname='demo' user='tahweela' password='tahweela'"
- db = database(db_configs)
- filename
- format
- filemode
- level
- logger = logging.getLogger()
- app = Flask('tahweela')
- auth = HTTPBasicAuth()
- client_passcode = None
- client_cred_id = None
- debug

6.71.1 Function Documentation

6.71.1.1 add_bank_account()

```
def server.server.add_bank_account ( )
```

register bank account for the authenticated client of the current session

@param: the post body is expect to be json with keys ["bank_name", branch_number", "account_number", "name_ref
@return return bid (banking id), since multiple bank accounts are supported, bid need to be sent for each tran

Definition at line 184 of file server.py.

```
184 def add_bank_account():
185   """ register bank account for the authenticated client of the current session
186
187   @param: the post body is expect to be json with keys ["bank_name", branch_number", "account_number",
       "name_reference"], a client can register more than one bank account for tahweela account.
188   @return return bid (banking id), since multiple bank accounts are supported, bid need to be sent for
       each transaction so that, the transactions are done with it.
189   """
190   req=request.get_json(force=True)
191   bank_name=req.get("bank_name", None)
192   branch_number=req.get("branch_number", None)
193   account_number=req.get("account_number", None)
194   name_reference=req.get("name_reference", "")
195   if not req or bank_name==None or branch_number==None or account_number==None:
196     logger.critical("incomplete request")
197     abort(401)
198   db.init()
199   ADD_BANK_ACCOUNT_LOCK=7
200   lock=ADD_BANK_ACCOUNT_LOCK
201   balance={}
202   try:
203     db.lock_advisory(lock)
204     cid=db.gets.credid2cid(client_cred_id)
205     logger.debug("client added")
206     #TODO mock the bank to get the balance!
207     balance=rand.random()*MAX_BALANCE
208     db.inserts.add_bank_account(cid, balance, bank_name, branch_number, account_number, name_reference)
209     balance=db.gets.get_balance_by_credid(client_cred_id)
210     db.commit(lock)
211   except psycopg2.DatabaseError as error:
212     print('err1')
213     db.rollback(lock)
214     logger.critical("assigning bank account failed, error: "+str(error))
215     abort(300)
216   except:
217     print('err2')
218     db.rollback(lock)
219     logger.critical("adding bank account failed")
220     abort(300)
221   finally:
222     db.close()
223   return jsonify({'balance':balance}), 201
224
225 @app.route(CONTACTS, methods=['POST'])
226 @auth.login_required
```

6.71.1.2 add_contact()

```
def server.server.add_contact ( )
```

get the credential for the given contact

Definition at line 227 of file server.py.

```
227 def add_contact():
228     """get the credential for the given contact
229     """
230     req=request.get_json(force=True)
231     email=req.get('email', None)
232     logger.info("requesting contact")
```

```
233     if not req or email==None:
234         logger.debug("incomplete URL")
235         abort(401)
236     payload={}
237     db.init()
238     try:
239         db.repeatable_read()
240         if not db.exists.account_byemail(email):
241             logger.critical("contact doesn't exist")
242             raise Exception("contact doesn't exists!")
243         cid=db.gets.get_client_id_byemail(email)
244         credid=db.gets.cid2credid(cid)
245         # get name given cid
246         name=db.gets.get_name(cid)
247         payload = {"credid": credid}
248         db.commit()
249     except psycopg2.DatabaseError as error:
250         db.rollback()
251         logger.critical("request failed, error:"+str(error))
252         abort(300)
253     except Exception as error:
254         db.rollback()
255         logger.critical("requesting failed"+str(error))
256         print('FAILURE, err: ', str(error))
257         abort(400)
258     finally:
259         db.close()
260     return jsonify(payload), 201
261
262 @app.route(BALANCE, methods=['GET'])
263 @auth.login_required
```

6.71.1.3 authenticate()

```
def server.server.authenticate (
                user,
                passcode )
```

authenticate user, with username/password

user can be user's email, or registered name

Definition at line 46 of file server.py.

```
46 def authenticate(user, passcode):
47     """authenticate user, with username/password
48
49     user can be user's email, or registered name
50     """
51     print("AUTHENTICATING.....")
52     '''
53     #TODO (fix) it seams that is_email isn't strong enough, it fails for some emails
54     if is_email(user):
55       print('authenticating user by email: ', user)
56       if db.exists.account_byemail(user):
57         cid=db.gets.get_client_id_byemail(user)
58       else:
59         cid=-1
60     else:
61       print('authenticating user by name', user)
62       if db.exists.account_byname(user, passcode):
63         cid=db.gets.get_client_id_byname(user)
64       else:
65         cid=-1
66     '''
67     print('authenticating [{}+{}]...'.format(user, passcode))
68     print('authenticating user by email: ', user)
69     if db.exists.account_byemail(user):
70       cid=db.gets.get_client_id_byemail(user)
71     else:
72       print('doesnt exist!')
73       cid=-1
74     if cid==-1:
75       print('authenticating user by name', user)
76       if db.exists.account_byname(user, passcode):
```

```
77          cid=db.gets.get_client_id_byname(user)
78        else:
79          print('doesnt exist!')
80          cid=-1
81      #TODO support user as credid itself
82      if cid==-1: #doesn't exists
83          logger.critical("authentication failed")
84          print('authentication failed ,doesnt exist')
85          return None
86      print('SET GLOBAL!')
87      #TODO HOW TO MAKE SURE THAT CREDID IS UNIQUE?
88      # brute force, keep trying new values that does exist,
89      # is the simplest, otherwise use conguential generator
90      global client_cred_id, client_passcode
91      credid=db.gets.cid2credid(cid)
92      #TODO implement cid2credid
93      logger.debug("authenticating client with cred {}:{}". format(user, passcode))
94      print('credid: ', credid)
95      print("username: ", user)
96      print("password: ", passcode)
97      db.init()
98      try:
99        db.repeatable_read()
100       passcode_eq= db.gets.get_password(credid)
101       print("pass_code: ", passcode_eq)
102       logger.info("username:"+str(user)+", passcode:"+str(passcode))
103       if not passcode==passcode_eq:
104         print('FAILURE PASSOWRD MISMATCH')
105         raise Exception("password mismatch!")
106       db.commit()
107       print('AUTHENTICATION SUCCESS....')
108     except psycopg2.DatabaseError as error:
109       print('AUTHENTICATION FAILURE')
110       db.rollback()
111       logger.critical("authentication failed with error: "+str(error))
112       abort(300)
113       return None #doesn't reach here
114     except:
115       print('AUTHENTICATION FAILURE')
116       db.rollback()
117       logger.critical("authentication failed ")
118       abort(300)
119       return None #doesn't reach here!
120     finally:
121       db.close()
122     client_cred_id=credid
123     client_passcode=passcode
124     return user
125
126 @auth.error_handler
```

References server.server.format.

6.71.1.4 get_balance()

```
def server.server.get_balance ( )
```

get balance of the current client

Definition at line 264 of file server.py.

```
264 def get_balance():
265     """ get balance of the current client
266
267     """
268     balance=None
269     logger.info("balance requested")
270     db.init()
271     try:
272         db.repeatable_read()
273         balance=db.gets.get_balance_by_credid(client_cred_id)
274         db.commit()
275     except psycopg2.DatabaseError as error:
276         db.rollback()
277         logger.critical("failed request, error: "+str(error))
278         abort(300)
```

```
279     except:
280         db.rollback()
281         logger.critical("failed request")
282         abort(300)
283     finally:
284         db.close()
285     return jsonify({"balance": balance}), 201
286
287 @app.route(LEDGER, methods=['POST'])
288 @auth.login_required
```

6.71.1.5 get_credid()

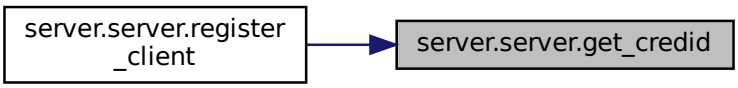
```
def server.server.get_credid ( )
```

Definition at line 39 of file server.py.

```
39 def get_credid():
40     return int(3333333333333*random.random())
41
```

Referenced by server.server.register_client().

Here is the caller graph for this function:



6.71.1.6 is_email()

```
def server.server.is_email (
              email )
```

Definition at line 42 of file server.py.

```
42 def is_email(email):
43   return bool(re.search(r"^[\w\.\+\-]+\@[\w]+\.[a-z]{2,3}$", email))
44
45 @auth.verify_password
```

6.71.1.7 make_transaction()

```
def server.server.make_transaction ( )
```

Definition at line 322 of file server.py.

```
322 def make_transaction():
323     req=request.get_json(force=True)
324     recipt_credid=req['credid']
325     amount=req['amount']
326     currency=req.get('currency', 'USD')
327     cur_balance=db.gets.get_balance_by_credid(client_cred_id)
328     if cur_balance<amount+FEE:
329         logger.info("client doesn't have enough credit to make transaction")
330         abort(400)
331     #TODO transform this to the required currency
332     # accept currency in the api
333     MAX_DAILY=10000
334     MAX_WEEKLY=50000
335     #here the weekly/daily conditions are pivoted by the current moment only, note that the bank system
        can have specific pivot hour (the first momemnt of the day, it's specific for the bank system, and
        need to be known before hand)
336     week_past=datetime.datetime.now()-datetime.timedelta(days=7)
337     day_past=datetime.datetime.now()-datetime.timedelta(days=1)
338     #TODO abide to the  the constrains
339     logger.info("making purchase")
340     if not req or amount==None or recipt_credid==None:
341         logger.critical("incomplete URL empty request")
342         abort(401)
343     #gid=req['id']
344     db.init()
345     MAKE_TRANSACTION_LOCK=9
346     lock=MAKE_TRANSACTION_LOCK
347     print('start transaction')
348     try:
349         db.lock_advisory(lock)
350         weekly_trx_sum=db.gets.get_transactions_sum(client_cred_id, week_past)
351         daily_trx_sum=db.gets.get_transactions_sum(client_cred_id, day_past)
352         print('got trx sum! weekly: {}, daily{}'.format(weekly_trx_sum, daily_trx_sum))
353         if weekly_trx_sum+amount>MAX_WEEKLY or daily_trx_sum+amount>MAX_DAILY:
354             logger.info("client passed the daily/weekly limit")
355             raise Exception("client passed the daily/weekly limits")
356         #add transaction
357         db.inserts.insert_trx(recipt_credid, client_cred_id, amount)
358         #TODO this can be minimized directly by credid
359         src_balance=db.gets.get_balance_by_credid(client_cred_id)-(amount+FEE)
360         des_balance=db.gets.get_balance_by_credid(recipt_credid)+amount
361         src_cid=db.gets.credid2cid(client_cred_id)
362         des_cid=db.gets.credid2cid(recipt_credid)
363         if src_cid==des_cid:
364             logger.critical("you can't make transaction with oneself!")
365             abort(403)
366             raise Exception("you can't make transaction with oneself!")
367         db.updates.update_account(src_cid, src_balance)
368         db.updates.update_account(des_cid, des_balance)
369         trx = {'trx_dest': recipt_credid,
370                'trx_src': client_cred_id,
371                'trx_cost': amount}
372         payload={'balance': src_balance,
373                  'transactions': trx}
374         db.commit()
375     except psycopg2.DatabaseError as error:
376         db.rollback()
377         logger.critical("transaction failed, error: "+str(error))
378         abort(300)
379     except:
380         db.rollback()
381         logger.critical("transaction failed")
382         abort(300)
383     finally:
384         db.unlock_advisory(lock)
385         db.close()
386     return jsonify(payload), 201
387 '''
388 @app.route(GOODS, methods=['POST'])
389 @auth.login_required
390 def add_goods():
391     req=request.get_json(force=True)
392     logger.info("adding new good to the market")
393     #TODO goods should be passed with authentication,
394     # how to authenticate both in requests, and in flask
395     if not req or 'goods' not in req:
396         logger.critical("URL is incomplete missing goods")
397         abort(401)
398     goods={}
399     db.init()
```

```
400     lock=1
401     try:
402         db.lock_advisory(lock)
403         cid=db.gets.credid2cid(client_cred_id)
404         if cid==None:
405             logger.critical("client not found associated with given id")
406             abort(403)
407             raise Exception("invalid client")
408         goods=req['goods']
409         for good in goods:
410             print("good", good[0], good[1], good[2])
411             gid=db.inserts.add_good(good[0], good[1], good[2])
412             db.inserts.add_owner(cid, gid)
413         db.commit(lock)
414     except psycopg2.DatabaseError as error:
415         db.rollback(lock)
416         logger.critical("error adding good, error: "+str(error))
417         abort(300)
418     except:
419         db.rollback(lock)
420         logger.critical("error adding good")
421         abort(300)
422     finally:
423         db.close()
424     return jsonify({'goods':goods}), 201
425
426 @app.route(GOODS, methods=['GET'])
427 @auth.login_required
428 def get_goods():
429     logger.info("requesting good")
430     goods={}
431     db.init()
432     try:
433         db.repeatable_read()
434         goods = db.gets.get_all_goods()
435         print('goods:', goods)
436         db.commit()
437     except psycopg2.DatabaseError as error:
438         print('db except!')
439         db.rollback()
440         logger.critical("process failed, error: "+str(error))
441         abort(300)
442     except:
443         print("except")
444         db.rollback()
445         logger.critical("process failed")
446         abort(300)
447     finally:
448         db.close()
449         #TODO change column names
450     return jsonify({"goods": goods}), 201
451 @app.route(PURCHASE, methods=['POST'])
452 @auth.login_required
453 def make_purchase():
454     req=request.get_json(force=True)
455     logger.info("making purchase")
456     if not req:
457         logger.critical("incomplete URL empty request")
458         abort(401)
459     gid=req['id']
460     db.init()
461     lock=4
462     try:
463         db.lock_advisory(lock)
464         # cross reference owner_id, with good_id, with credentials
465         # return credential id of the owner
466         credid=db.gets.get_credid_with_gid(gid)
467         # make, and add new transaction such that increase,
468         # and decrease of the src/des balance need to be performed in single transaction, then add the
    transactioin to the ledger, if failed rollback
469         cost=db.gets.get_new_price(gid)+FEE
470         src_balance=db.gets.get_balance_by_credid(client_cred_id)-(cost+FEE)
471         des_balance=db.gets.get_balance_by_credid(credid)+cost
472         src_cid=db.gets.credid2cid(client_cred_id)
473         des_cid=db.gets.credid2cid(credid)
474         if src_cid==des_cid:
475             logger.critical("you can't make purchase with oneself!")
476             abort(403)
477             raise Exception("you can't make purchase with oneself!")
478         db.updates.update_account(src_cid, src_balance)
479         db.updates.update_account(des_cid, des_balance)
480         #TODO the ownership in the client side need to be updated as well,
481         # in the database, and in the stateful list in memory
482         # also fetch corresponding oid!
483         #update_owner(oid, gid)
484         trx = {'trx_dest': credid,
485                'trx_src': client_cred_id,
```

```
486                 'good_id': gid}
487         payload={'balance': src_balance,
488                  'transactions': trx}
489         db.commit()
490     except psycopg2.DatabaseError as error:
491         db.rollback()
492         logger.critical("purchase failed, error: "+str(error))
493         abort(300)
494     except:
495         db.rollback()
496         logger.critical("purchase failed")
497         abort(300)
498     finally:
499         db.unlock_advisory(lock)
500         db.close()
501     return jsonify(payload), 201
502 "'
503
```

References server.server.format.

6.71.1.8 register_client()

```
def server.server.register_client ( )
```

```
register new client with email/name + password, expected json body would have keys ["name", "email", "passcode
```

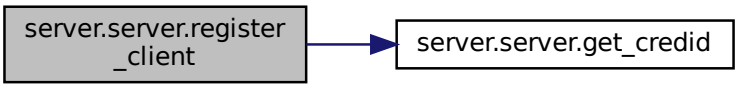
Definition at line 132 of file server.py.

```
132 def register_client():
133   """register new client with email/name + password, expected json body would have keys ["name",
       "email", "passcode"]
134
135   """
136   #TODO this can through exception, need to handle it
137   req=request.get_json(force=True)
138   print('req: ', req)
139   name=req.get('name', None)
140   passcode=req.get('passcode', None)
141   email=req.get('email', None)
142   if not req or name==None or email==None or passcode==None:
143     logger.critical("url is incomplete missing name")
144     print("incomplete!!!")
145     abort(400)
146   #TODO add constraint, and verification for the name/email, and passcode (at least not None!)
147
148   cred_id=get_credid()
149   logger.info("registering trader for client: "+ name)
150   bid=0
151   db.init()
152   lock=2
153   #TODO generalize get_credid
154   try:
155     db.lock_advisory(lock)
156     email_exists=db.exists.account_byemail(email)
157     if email_exists:
158       print('email exists')
159       abort(400)
160       logger.debug("account {}/{} + {} already exists".\
161                    format(name, email, passcode))
162       raise Exception("account already exists!")
163     cid=db.inserts.add_client(req['name'], req['email'])
164     logger.debug("client added")
165     db.inserts.register(cid, passcode, cred_id)
166     db.commit(lock)
167   except psycopg2.DatabaseError as error:
168     print('REGISTRATION FAILURE')
169     db.rollback(lock)
170     logger.critical("registering failed, error: "+str(error))
171     abort(300)
172   except:
173     print('REGISTRATION FAILURE')
174     db.rollback(lock)
175     logger.critical("registering failed")
176     abort(400)
177   finally:
```

```
178     db.close()
179     res = {'cred_id': cred_id}
180   return jsonify(res), 201
181
182 @app.route(ADD_BANK_ACCOUNT, methods=['POST'])
183 @auth.login_required
```

References server.server.format, and server.server.get_credid().

Here is the call graph for this function:



6.71.1.9 unauthorized()

```
def server.server.unauthorized ( )
```

Definition at line 127 of file server.py.

```
127 def unauthorized():
128     return make_response(jsonify({'error': "forbidden access"}), 403)
129
130 #TODO add message to status code response
131 @app.route(REGISTER, methods=['POST'])
```

6.71.1.10 update_ledger()

```
def server.server.update_ledger ( )
```

Definition at line 289 of file server.py.

```
289 def update_ledger():
290     req=request.get_json(force=True)
291     logger.info("requesting ledger update")
292     if not req:
293         logger.critical("incomplete URL empty request!")
294         abort(401)
295     st_dt=req['trx_dt']
296     payload={}
297     db.init()
298     lock=3
299     try:
300         db.lock_advisory(lock)
301         print("getting sells")
302         sells_trax=db.gets.get_sells(client_cred_id, st_dt).to_json()
303         print("sells: ", sells_trax)
304         balance=db.gets.get_balance_by_credid(client_cred_id)
305         payload={'transactions': sells_trax, \
306                  'balance': balance}
307         db.commit()
308     except psycopg2.DatabaseError as error:
309         db.rollback()
310         logger.critical("request failure, error: "+ str(error))
311         abort(300)
312     except:
313         db.rollback()
314         logger.critical("request failure")
315         abort(300)
316     finally:
317         db.unlock_advisory(lock)
318         db.close()
319     return jsonify(payload), 201
320 @app.route(TRANSACTION, methods=['POST'])
321 @auth.login_required
```

6.71.2 Variable Documentation

6.71.2.1 app

```
server.server.app = Flask('tahweela')
```

Definition at line 30 of file server.py.

6.71.2.2 auth

```
server.server.auth = HTTPBasicAuth()
```

Definition at line 31 of file server.py.

6.71.2.3 client_cred_id

```
server.server.client_cred_id = None
```

Definition at line 35 of file server.py.

6.71.2.4 client_passcode

```
server.server.client_passcode = None
```

Definition at line 34 of file server.py.

6.71.2.5 db

```
server.server.db = database(db_configs)
```

Definition at line 23 of file server.py.

6.71.2.6 db_configs

```
string server.server.db_configs = "dbname='demo' user='tahweela' password='tahweela'"
```

Definition at line 20 of file server.py.

6.71.2.7 debug

```
server.server.debug
```

Definition at line 505 of file server.py.

Referenced by queries.database.inserts.add_bank_account(), core.queries.database.inserts.add_bank_account(), queries.database.gets.cid2credid(), core.queries.database.gets.cid2credid(), queries.database.gets.credid2cid(), core.queries.database.gets.credid2cid(), queries.database.gets.get(), core.queries.database.gets.get(), queries.database.gets.get_all_clients(), core.queries.database.gets.get_all_clients(), queries.database.gets.get_all_contacts(), core.queries.database.gets.get_all_contacts(), queries.database.gets.get_all_credentials(), core.queries.database.gets.get_all_credentials(), queries.database.gets.get_balance_by_cid(), core.queries.database.gets.get_balance_by_cid(), queries.database.gets.get_balance_by_credid(), core.queries.database.gets.get_balance_by_credid(), queries.database.gets.get_banking_id(), core.queries.database.gets.get_banking_id(), queries.database.gets.get_client_id(), core.queries.database.gets.get_client_id(), queries.database.gets.get_client_id_byemail(), core.queries.database.gets.get_client_id_byemail(), queries.database.gets.get_client_id_byname(), core.queries.database.gets.get_client_id_byname(), queries.database.gets.get_credential(), core.queries.database.gets.get_credential(), core.queries.database.gets.get_currency_id(), core.queries.database.gets.get_currency_name(), queries.database.gets.get_last_timestamp(), core.queries.database.gets.get_last_timestamp(), queries.database.gets.get_password(), core.queries.database.gets.get_password(), core.queries.database.gets.get_preference_currency_bycid(), queries.database.gets.get_sells(), core.queries.database.gets.get_sells(), queries.database.gets.get_transactions(), core.queries.database.gets.get_transactions(), queries.database.gets.get_transactions_sum(), core.queries.database.gets.get_transactions_sum(), queries.database.inserts.insert_contact(), core.queries.database.inserts.insert_contact(), queries.database.inserts.insert_trx(), core.queries.database.inserts.insert_trx(), queries.database.inserts.register(), core.queries.database.inserts.register(), queries.database.gets.to_dollar(), and core.queries.database.gets.to_euro().

6.71.2.8 filemode

```
server.server.filemode
```

Definition at line 26 of file server.py.

6.71.2.9 filename

```
server.server.filename
```

Definition at line 24 of file server.py.

6.71.2.10 format

```
server.server.format
```

Definition at line 25 of file server.py.

Referenced by server.server.authenticate(), server.server.make_transaction(), and server.server.register_client().

6.71.2.11 level

```
server.server.level
```

Definition at line 27 of file server.py.

6.71.2.12 logger

```
server.server.logger = logging.getLogger()
```

Definition at line 28 of file server.py.

6.71.2.13 seed

```
server.server.seed = int.from_bytes(os.urandom(2), 'big')
```

Definition at line 11 of file server.py.

6.72 setup Namespace Reference

Data Structures

- class CleanCommand

Functions

- def read (fname)

Variables

- string description
- name
- version
- author
- author_email
- license
- packages
- long_description
- cmdclass

6.72.1 Function Documentation

6.72.1.1 read()

```
def setup.read (
               fname )
```

Definition at line 5 of file setup.py.

```
5 def read(fname):
6     return open(os.path.join(os.path.dirname(__file__), fname)).read()
7
8
```

6.72.2 Variable Documentation

6.72.2.1 author

```
setup.author
```

Definition at line 30 of file setup.py.

6.72.2.2 author_email

```
setup.author_email
```

Definition at line 31 of file setup.py.

6.72.2.3 cmdclass

`setup.cmdclass`

Definition at line 36 of file setup.py.

6.72.2.4 description

`setup.description`

Initial value:

```
1 =  """
2 demo for tahweela with stochastic trading
3 """
```

Definition at line 23 of file setup.py.

6.72.2.5 license

`setup.license`

Definition at line 33 of file setup.py.

6.72.2.6 long_description

`setup.long_description`

Definition at line 35 of file setup.py.

6.72.2.7 name

`setup.name`

Definition at line 28 of file setup.py.

Referenced by core_test.RestfulTest.__register_bank_account().

6.72.2.8 packages

`setup.packages`

Definition at line 34 of file setup.py.

6.72.2.9 version

`setup.version`

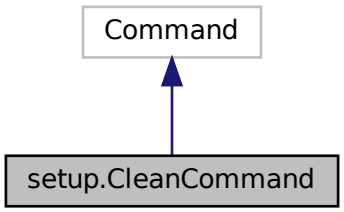
Definition at line 29 of file setup.py.

Chapter 7

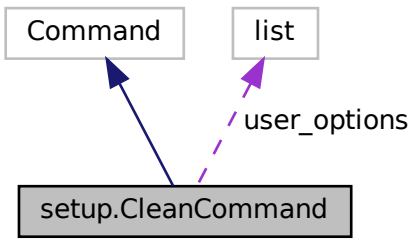
Data Structure Documentation

7.1 setup.CleanCommand Class Reference

Inheritance diagram for setup.CleanCommand:



Collaboration diagram for setup.CleanCommand:



Public Member Functions

- def initialize_options (self)
- def finalize_options (self)
- def run (self)

Static Public Attributes

- list user_options = [ ]

7.1.1 Detailed Description

```
Custom clean command to tidy up the project root.
```

Definition at line 9 of file setup.py.

7.1.2 Member Function Documentation

7.1.2.1 finalize_options()

```
def setup.CleanCommand.finalize_options (
              self )
```

Definition at line 16 of file setup.py.

```
16     def finalize_options(self):
17         pass
18
```

7.1.2.2 initialize_options()

```
def setup.CleanCommand.initialize_options (
              self )
```

Definition at line 13 of file setup.py.

```
13     def initialize_options(self):
14         pass
15
```

7.1.2.3 run()

```
def setup.CleanCommand.run (
              self )
```

Definition at line 19 of file setup.py.

```
19     def run(self):
20         os.system("rm -vrf ./build ./dist ./*.pyc ./*.egg-info")
21
22
```

7.1.3 Field Documentation

7.1.3.1 user_options

```
list setup.CleanCommand.user_options = []  [static]
```

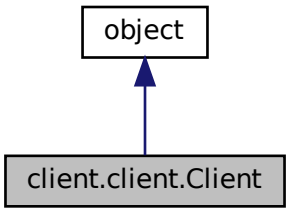
Definition at line 11 of file setup.py.

The documentation for this class was generated from the following file:

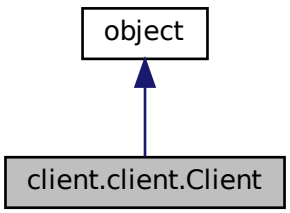
- setup.py

7.2 client.client.Client Class Reference

Inheritance diagram for client.client.Client:



Collaboration diagram for client.client.Client:



Public Member Functions

- def __init__ (self)
- def auth (self)
- def register (self)
- def add_contact (self, email, name)
- def autotract (self)

Data Fields

- faker
- name
- passcode
- email
- bank_name
- branch_number
- account_number
- name_reference
- db
- my_contacts
- uname

  TODO credid.
- pas
- balance

Private Member Functions

- def __register_bank_account (self, bank_name=None, branch_number=None, account_number=None, name_reference=None)
- def __transact (self, credid, amount, currency='USD')
- def __add_trax (self, trans)
- def __update_balance (self, balance)
- def __ledger_timestamp (self)
- def __update_ledger (self)

7.2.1 Detailed Description

Definition at line 71 of file client.py.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 __init__()

```
def client.client.Client.__init__ (
              self )
```

Definition at line 72 of file client.py.

```
72     def __init__(self):
73         seed=int.from_bytes(os.urandom(2), 'big')
74         self.faker = Faker()
75         random.seed(seed)
76         Faker.seed(seed)
77         #
78         self.name=get_name()
79         self.passcode=get_rand_pass()
80         self.email=get_email()
81         self.bank_name=get_bank_name()
82         self.branch_number=get_branch_number()
83         self.account_number=get_account_number()
84         self.name_reference=get_name_reference()
85         #
86         self.db = database(db_configs)
87         self.my_contacts = []
88         #Register user int he network
89         self.register()
90         #register user with bank account
91         self.__register_bank_account(self.bank_name, self.branch_number, self.account_number,
       self.name_reference)
92         logger.info("client initialized")
93         #self.all_goods = []
94         #self.__init_goods()
95         # credentials username:password
96
```

7.2.3 Member Function Documentation

7.2.3.1 __add_trax()

```
def client.client.Client.__add_trax (
              self,
              trans )  [private]
```

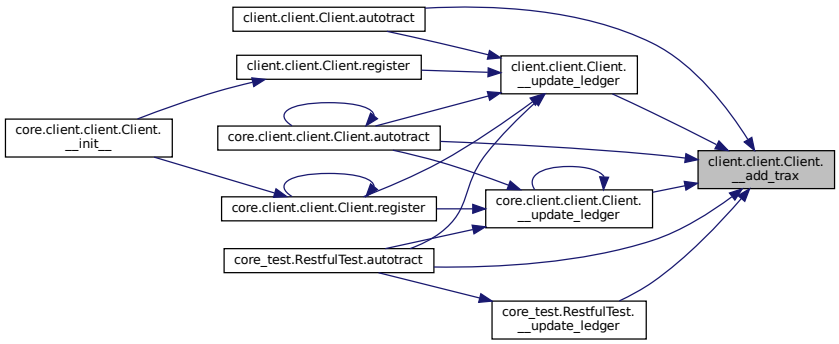
Definition at line 210 of file client.py.

```
210     def __add_trax(self, trans):
211         trans=trans['transactions'] #TODO (res)
212         self.db.init()
213         try:
214             print("trans: ", trans)
215             #self.db.inserts.insert_trx(trans["trx_src"], trans['trx_dest'], trans['trx_gid'])
216             self.db.inserts.insert_trx(trans["trx_src"], trans['trx_dest'], trans['trx_cost'])
217             self.db.commit()
218         except psycopg2.DatabaseError as error:
219             self.db.rollback()
220         finally:
221             self.db.close()
```

References client.client.Client.db.

Referenced by client.client.Client.__update_ledger(), core.client.client.Client.__update_ledger(), core_test.↩ RestfulTest.__update_ledger(), core.client.client.Client.autotract(), client.client.Client.autotract(), and core_test.↩ RestfulTest.autotract().

Here is the caller graph for this function:



7.2.3.2 __ledger_timestamp()

```
def client.client.Client.__ledger_timestamp (
            self )  [private]
```

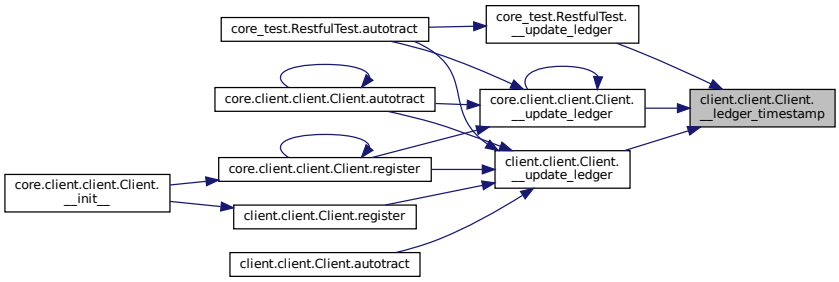
Definition at line 228 of file client.py.

```
228     def __ledger_timestamp(self):
229         dt=None
230         self.db.init()
231         try:
232             dt=self.db.gets.get_last_timestamp()
233             self.db.commmit()
234         except psycopg2.DatabaseError as error:
235             logger.critical("failed to retrieve ledger time stamp")
236             self.db.rollback()
237         finally:
238             self.db.close()
239         print("timestamp: ", dt)
240 
241 
242         return dt if not dt==None else datetime.datetime.now().strftime(TIMESTAMP_FORMAT)
```

References client.client.Client.db.

Referenced by client.client.Client.__update_ledger(), core.client.client.Client.__update_ledger(), and core_test.↩ RestfulTest.__update_ledger().

Here is the caller graph for this function:



7.2.3.3 __register_bank_account()

```
def client.client.Client.__register_bank_account (
              self,
              bank_name = None,
              branch_number = None,
              account_number = None,
              name_reference = None )  [private]
```

check if this client has credentials, if not register

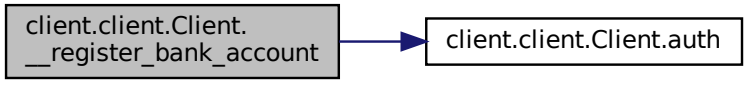
Definition at line 143 of file client.py.

```
143     def __register_bank_account(self, bank_name=None, branch_number=None, account_number=None,
        name_reference=None):
144         """ check if this client has credentials, if not register
145
146         """
147         #check if user have credentials
148         BRANCH_NUM_MAX=100
149         ACCOUNT_NUMBER_MAX=10000000000
150         self.db.init()
151         try:
152             #register user
153             bank_name = self.faker.name() if bank_name==None else bank_name
154             branch_number = random.random()*BRANCH_NUM_MAX if branch_number==None else branch_number
155             account_number= random.random()*ACCOUNT_NUMBER_MAX if account_number==None else
        account_number
156             name_reference= self.faker.name() if name_reference==None else name_reference
157             payload={'bank_name':bank_name,
158                      'branch_number':branch_number,
159                      'account_number':account_number,
160                      'name_reference':name_reference}
161             res=requests.post(ADD_BANK_ACCOUNT_URL, \
162                               data=json.dumps(payload), \
163                               auth=self.auth())
164             response = json.loads(res.text)
165             scode=res.status_code
166             #TODO support multiple accounts
167             #create local client table for banking,
168             #to insert balance, or each account.
169             print('response for balance:  ', response)
170             self.balance=response['balance']
171             self.db.commit()
172
173         except psycopg2.DatabaseError as error:
174             logger.critical("bank account registration failed!, error: "+ str(error))
175             print("bank account register failed!, error: "+str(error))
176             self.db.rollback()
177         finally:
178             self.db.close()
```

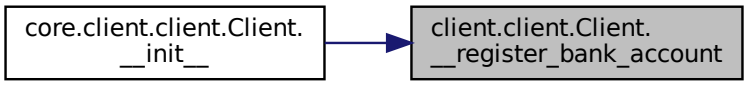
References client.client.Client.auth(), client.client.Client.db, client.client.Client.faker, and client.client.Client.name.

Referenced by core.client.client.Client.__init__().

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.4 __transact()

```
def client.client.Client.__transact (
                self,
                credid,
                amount,
                currency = 'USD' )  [private]
```

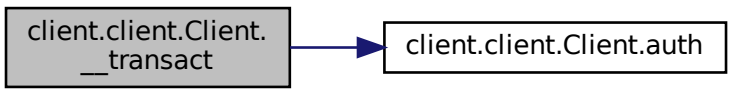
Definition at line 202 of file client.py.

```
202     def __transact(self, credid, amount, currency='USD'):
203         pur_item = {'credid': credid,
204                     'amount':amount,
205                     'currency': 'USD'}
206         ret=requests.post(PURCHASE_URL, data=json.dumps(pur_item), auth=self.auth())
207         response=ret.text#ret.text.decode('utf8')#.replace('"', "'")
208         trx=json.loads(response)
209         return trx
```

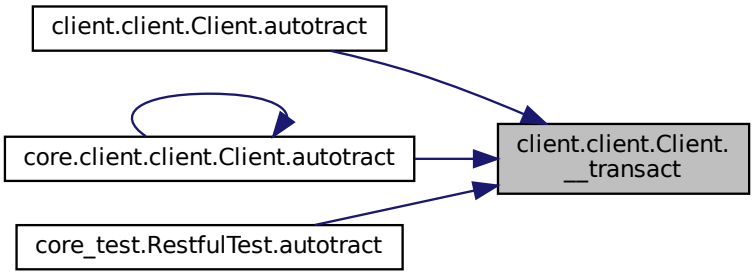
References client.client.Client.auth().

Referenced by client.client.Client.autotract(), core.client.client.Client.autotract(), and core_test.RestfulTest.↩ autotract().

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.5 __update_balance()

```
def client.client.Client.__update_balance (
              self,
              balance )  [private]
```

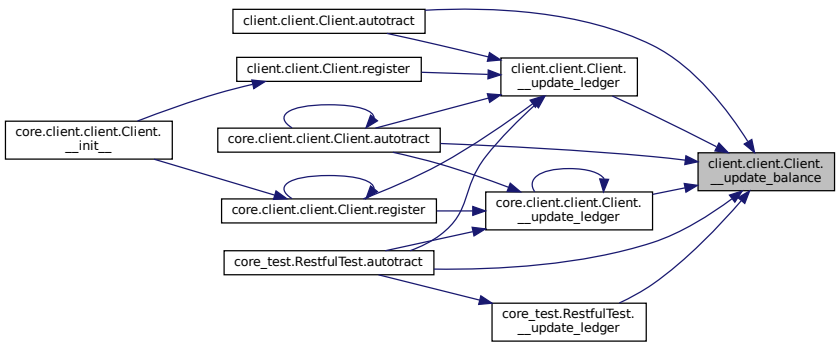
Definition at line 222 of file client.py.

```
222     def __update_balance(self, balance):
223         #TODO fix
224         if type(balance)==dict:
225             self.balance=balance['balance']
226             return
227         self.balance=balance
```

References client.client.Client.balance, and client.client.type.

Referenced by client.client.Client.__update_ledger(), core.client.client.Client.__update_ledger(), core_test.↩ RestfulTest.__update_ledger(), core.client.client.Client.autotract(), client.client.Client.autotract(), and core_test.↩ RestfulTest.autotract().

Here is the caller graph for this function:



7.2.3.6 __update_ledger()

```
def client.client.Client.__update_ledger (
              self )  [private]
```

update ledger with sold goods, and update balance

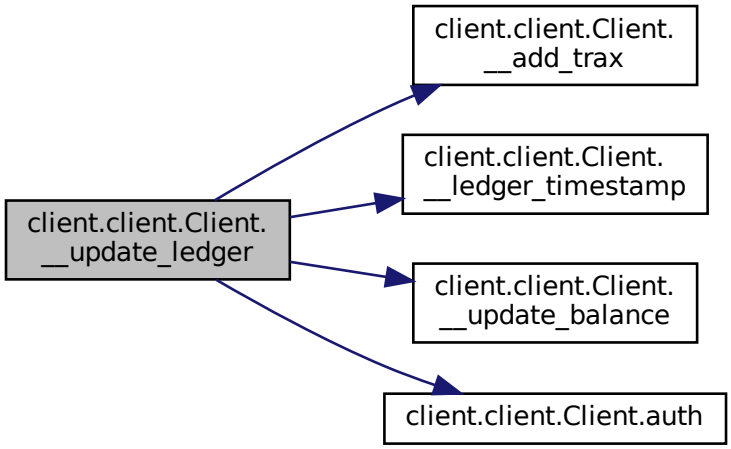
Definition at line 243 of file client.py.

```
243     def __update_ledger(self):
244         """update ledger with sold goods, and update balance
245 
246         """
247         ret=requests.post(LEDGER_URL, data=json.dumps({'trx_dt': self.__ledger_timestamp()}),
      auth=self.auth())
248         #TODO always process the status code!
249         print('status code: ', ret.status_code)
250         res = json.loads(ret.text) #self.__get_dict(ret.text)
251         print('res: ', res)
252         print("new balance: ", res['balance'])
253         self.__update_balance(res['balance'])
254         self.db.init()
255         try:
256             new_trxs=res['transactions']
257             print("trxs: ", new_trxs)
258             transactions=pd.read_json(new_trxs)
259             print("trxs pandas: ", transactions)
260             if len(transactions)==0:
261                 logger.info("ledger is up to date!")
262                 return
263             for i in range(len(transactions)-1):
264                 self.__add_trax(transactions.iloc[i])
265             db.commit()
266         except psycopg2.DatabaseError as error:
267             print('FAILURE LEDGER UPDATE: ', str(error))
268             self.db.rollback()
269         except Exception as error:
270             print('FAILURE LEDGER UPDATE: ', str(error))
271             self.db.rollback()
272         finally:
273             self.db.close()
```

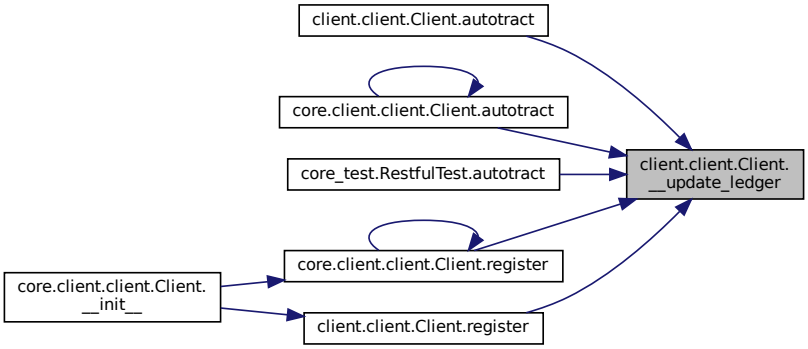
References client.client.Client.__add_trax(), client.client.Client.__ledger_timestamp(), client.client.Client.__update_balance(), client.client.Client.auth(), and client.client.Client.db.

Referenced by client.client.Client.autotract(), core.client.client.Client.autotract(), core_test.RestfulTest.autotract(), core.client.client.Client.register(), and client.client.Client.register().

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.7 add_contact()

```
def client.client.Client.add_contact (
              self,
              email,
              name )


Fetch the client with given email/name

@param email: contact email
@param name: contact name
```

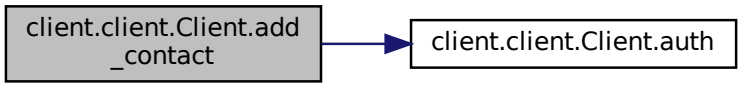
Definition at line 179 of file client.py.

```
179     def add_contact(self, email, name):
180         """Fetch the client with given email/name
181 
182         @param email: contact email
183         @param name: contact name
184         """
185         #get client credential id
186         cred={'email': email}
187         ret=requests.post(CONTACTS_URL, \
188                           data=json.dumps(cred), \
189                           auth=self.auth())
190         response=ret.text#.decode('utf8')#.replace('"', "'")
191         print('add contact response: ', response)
192         res = json.loads(response)
193         credid = res['credid']
194         self.db.init()
195         try:
196             self.db.inserts.insert_contact(email, name, credid)
197             self.db.commit()
198         except psycopg2.DatabaseError as error:
199             self.db.rollback()
200         finally:
201             self.db.close()
```

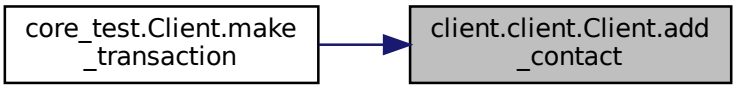
References client.client.Client.auth(), and client.client.Client.db.

Referenced by core_test.Client.make_transaction().

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.8 auth()

```
def client.client.Client.auth (
              self )
```

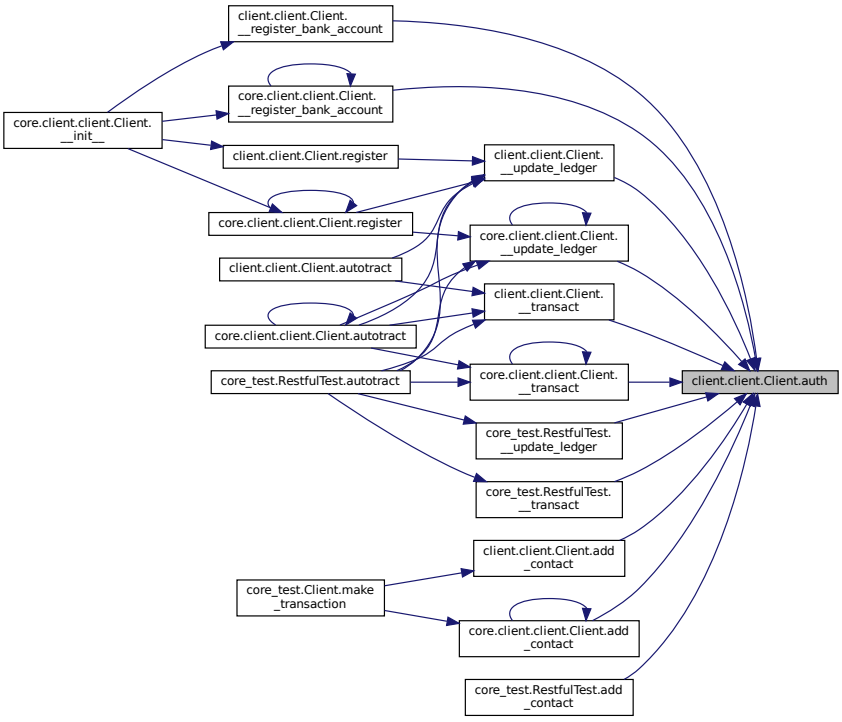
Definition at line 97 of file client.py.

```
97     def auth(self):
98         logger.info("auth ("+str(self.email)+":"+str(self.passcode)+")")
99         return HTTPBasicAuth(self.email, self.passcode)
100
```

References client.client.Client.email, and client.client.Client.passcode.

Referenced by client.client.Client.__register_bank_account(), core.client.client.Client.__register_bank_account(), client.client.Client.__transact(), core.client.client.Client.__transact(), core_test.RestfulTest.__transact(), core.client.client.Client.__update_ledger(), client.client.Client.__update_ledger(), core_test.RestfulTest.__update_ledger(), client.client.Client.add_contact(), core.client.client.Client.add_contact(), and core_test.RestfulTest.add_contact().

Here is the caller graph for this function:



7.2.3.9 autotract()

```
def client.client.Client.autotract (
              self )
```

```
Stochastic fake auto-tracting

trade with randomly with maintained contacts with 0.5 probability for each, and 0.1 probability for all contac
```

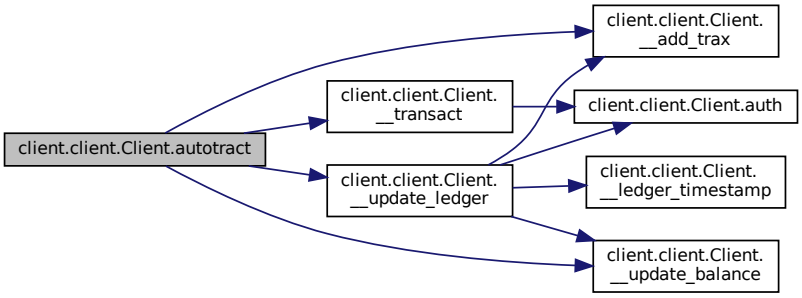
Definition at line 274 of file client.py.

```
274     def autotract(self):
275         """Stochastic fake auto-tracting
276 
277         trade with randomly with maintained contacts with 0.5 probability for each, and 0.1 probability
       for all contacts goods, update balance, and goods for each contact

278         """
279         #update balance, and add new transactions
280         MAX_TRACT_BALANCE=100000
281         self.__update_ledger()
282         for i in range(len(contacts_df-1)):
283             contact_credid=contacts_df.iloc[i]['contact_id']
284             contact_name=contacts_df.iloc[i]['contact_name']
285             contact_email=contacts_df.iloc[i]['contact_email']
286             amount=random.random()*MAX_TRACT_BALANCE
287             logger.info("making transaction with {}[{}] by amount: {}".\
288                         format(contact_name, contact_email, amount))
289             #if random.random()> STOCHASTIC_TRADE_THRESHOLD and good.cost <= self.balance:
290             #TODO (fix) doesn't work!
291             #trx=self.__purchase(good.gid)
```

```
292             trx=self.__transact(contact_credid, amount)
293             self.__add_trax(trx)
294             self.__update_balance(trx)
```

References client.client.Client.__add_trax(), client.client.Client.__transact(), client.client.Client.__update_balance(), client.client.Client.__update_ledger(), and client.client.format.

Here is the call graph for this function:



7.2.3.10 register()

```
def client.client.Client.register (
                self )
```

check if this client has credentials, if not register

Definition at line 101 of file client.py.

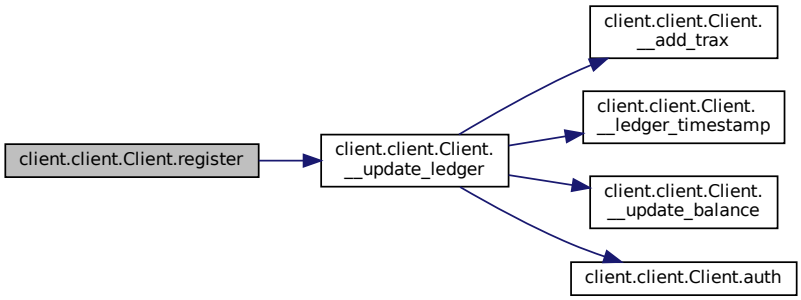
```
101     def register(self):
102         """ check if this client has credentials, if not register
103 
104         """
105         #check if user have credentials
106         self.db.init()
107         try:
108             if not self.db.exists.credential_exists(1):
109                 #register user
110                 #my_goods = [good(self.faker) for i in range(math.ceil(random.random()*MAX_GOODS))]
111                 #TODO check repose status_code makes sure it's request.codes.ok
112                 payload={'name':self.name, \
113                          'email': self.email, \
114                          'passcode': self.passcode}
115                 print('payload: ', payload)
116                 res=requests.post(REGISTER_URL, data=json.dumps(payload))
117                 response = json.loads(res.text)
118                 assert res.status_code==201, "status code is error {} ".format(res.status_code)
119                 print(response)
120                 credid=response['cred_id']
121                 #cid set to 0, since it never matters in the client side
122 
123                 self.db.inserts.register(0, self.passcode, credid)
124                 logger.debug("user registered with credentials username: {}, email: {}, passcode: {}"\
125                              .format(name, email, passcode))
126 
132             else:
133                 self.__update_ledger()
134             self.db.commit()
135             self.uname=email
136             self.pas=passcode
```

```
137         except psycopg2.DatabaseError as error:
138             logger.critical("registration failed!, error: "+ str(error))
139             print("register failed!, error: "+str(error))
140             self.db.rollback()
141         finally:
142             self.db.close()
```

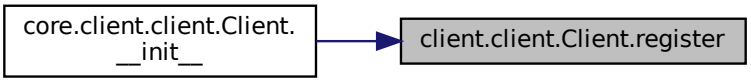
References client.client.Client.__update_ledger(), client.client.Client.db, client.client.Client.email, client.client.↩ format, client.client.Client.name, and client.client.Client.passcode.

Referenced by core.client.client.Client.__init__().

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.4 Field Documentation

7.2.4.1 account_number

`client.client.Client.account_number`

Definition at line 83 of file client.py.

Referenced by core.client.client.Client.__init__(), core.utils.PaymentGate.__init__(), and core_test.Client.register↩ _bank_account().

7.2.4.2 balance

client.client.Client.balance

Definition at line 170 of file client.py.

Referenced by core.client.client.Client.__register_bank_account(), core.client.client.Client.__update_balance(), client.client.Client.__update_balance(), core_test.RestfulTest.__update_balance(), core.utils.PaymentGate.↩ authenticated(), core.utils.PaymentGate.get_balance(), and core_test.Client.get_balance().

7.2.4.3 bank_name

client.client.Client.bank_name

Definition at line 81 of file client.py.

Referenced by core.client.client.Client.__init__(), core.utils.PaymentGate.__init__(), and core_test.Client.register↩ _bank_account().

7.2.4.4 branch_number

client.client.Client.branch_number

Definition at line 82 of file client.py.

Referenced by core.client.client.Client.__init__(), core.utils.PaymentGate.__init__(), and core_test.Client.register↩ _bank_account().

7.2.4.5 db

client.client.Client.db

Definition at line 86 of file client.py.

Referenced by client.client.Client.__add_trax(), core.client.client.Client.__add_trax(), core_test.RestfulTest.__↩ add_trax(), core.client.client.Client.__init__(), client.client.Client.__ledger_timestamp(), core.client.client.Client.↩ __ledger_timestamp(), core_test.RestfulTest.__ledger_timestamp(), core.client.client.Client.__register_bank_↩ account(), client.client.Client.__register_bank_account(), core_test.RestfulTest.__register_bank_account(), core.↩ client.client.Client.__update_ledger(), client.client.Client.__update_ledger(), core_test.RestfulTest.__update_↩ ledger(), core.client.client.Client.add_contact(), client.client.Client.add_contact(), core_test.RestfulTest.add_↩ contact(), core.client.client.Client.register(), and client.client.Client.register().

7.2.4.6 email

client.client.Client.email

Definition at line 80 of file client.py.

Referenced by core.client.client.Client.__init__(), client.client.Client.auth(), core.client.client.Client.auth(), core_test.Client.basic_auth(), core_test.Client.get_balance(), core_test.Client.register(), core.client.client.Client.register(), and client.client.Client.register().

7.2.4.7 faker

client.client.Client.faker

Definition at line 74 of file client.py.

Referenced by core.client.client.Client.__init__(), core.client.client.Client.__register_bank_account(), client.client.Client.__register_bank_account(), and core_test.RestfulTest.__register_bank_account().

7.2.4.8 my_contacts

client.client.Client.my_contacts

Definition at line 87 of file client.py.

Referenced by core.client.client.Client.__init__().

7.2.4.9 name

client.client.Client.name

Definition at line 78 of file client.py.

Referenced by core.client.client.Client.__init__(), client.client.Client.__register_bank_account(), core_test.Client.get_balance(), core_test.Client.register(), core.client.client.Client.register(), and client.client.Client.register().

7.2.4.10 name_reference

client.client.Client.name_reference

Definition at line 84 of file client.py.

Referenced by core.client.client.Client.__init__(), core.utils.PaymentGate.__init__(), and core_test.Client.register_bank_account().

7.2.4.11 pas

client.client.Client.pas

Definition at line 136 of file client.py.

Referenced by core_test.RestfulTest.__auth(), core.client.client.Client.register(), and core_test.RestfulTest.test_↩ register().

7.2.4.12 passcode

client.client.Client.passcode

Definition at line 79 of file client.py.

Referenced by core.client.client.Client.__init__(), client.client.Client.auth(), core.client.client.Client.auth(), core↩ _test.Client.basic_auth(), core_test.Client.get_balance(), core_test.Client.register(), core.client.client.Client.↩ register(), and client.client.Client.register().

7.2.4.13 uname

client.client.Client.uname

TODO credid.

post goods #

payload=json.dumps({'goods': [(g.name, g.cost, g.quality) for g in my_goods]}) requests.post(GOODS_URL, data=payload, auth=self.auth()) TODO assert that requests.text is equivalent to payload

Definition at line 135 of file client.py.

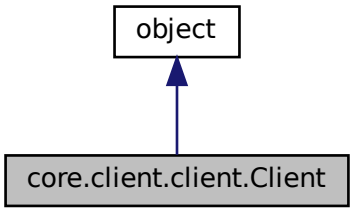
Referenced by core_test.RestfulTest.__auth(), core.client.client.Client.register(), and core_test.RestfulTest.test_↩ register().

The documentation for this class was generated from the following file:

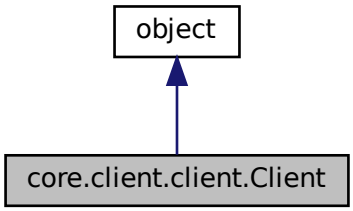
- core/build/lib/client/client.py

7.3 core.client.client.Client Class Reference

Inheritance diagram for core.client.client.Client:



Collaboration diagram for core.client.client.Client:



Public Member Functions

- def __init__ (self)
- def auth (self)
- def register (self)
- def add_contact (self, email, name)
- def autotract (self)
- def __init__ (self)
- def auth (self)
- def register (self)
- def add_contact (self, email, name)
- def autotract (self)

Data Fields

- faker
- name
- passcode
- email
- bank_name
- branch_number
- account_number
- name_reference
- db
- my_contacts
- uname

  TODO credid.
- pas
- balance

Private Member Functions

- def __register_bank_account (self, bank_name=None, branch_number=None, account_number=None, name_reference=None)
- def __transact (self, credid, amount, currency='USD')
- def __add_trax (self, trans)
- def __update_balance (self, balance)
- def __ledger_timestamp (self)
- def __update_ledger (self)
- def __register_bank_account (self, bank_name=None, branch_number=None, account_number=None, name_reference=None)
- def __transact (self, credid, amount, currency='USD')
- def __add_trax (self, trans)
- def __update_balance (self, balance)
- def __ledger_timestamp (self)
- def __update_ledger (self)

7.3.1 Detailed Description

Definition at line 71 of file client.py.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 __init__() [1/2]

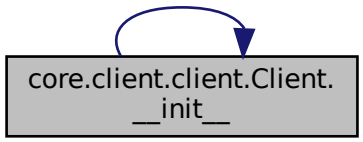
```
def core.client.client.Client.__init__ (
                self )
```

Definition at line 72 of file client.py.

```
72     def __init__(self):
73         seed=int.from_bytes(os.urandom(2), 'big')
74         self.faker = Faker()
75         random.seed(seed)
76         Faker.seed(seed)
77         #
78         self.name=get_name()
79         self.passcode=get_rand_pass()
80         self.email=get_email()
81         self.bank_name=get_bank_name()
82         self.branch_number=get_branch_number()
83         self.account_number=get_account_number()
84         self.name_reference=get_name_reference()
85         #
86         self.db = database(db_configs)
87         self.my_contacts = []
88         #Register user int he network
89         self.register()
90         #register user with bank account
91         self.__register_bank_account(self.bank_name, self.branch_number, self.account_number,
       self.name_reference)
92         logger.info("client initialized")
93         #self.all_goods = []
94         #self.__init_goods()
95         # credentials username:password
96
```

Referenced by core.client.client.Client.__init__().

Here is the caller graph for this function:



7.3.2.2 __init__() [2/2]

```
def core.client.client.Client.__init__ (
                self )
```

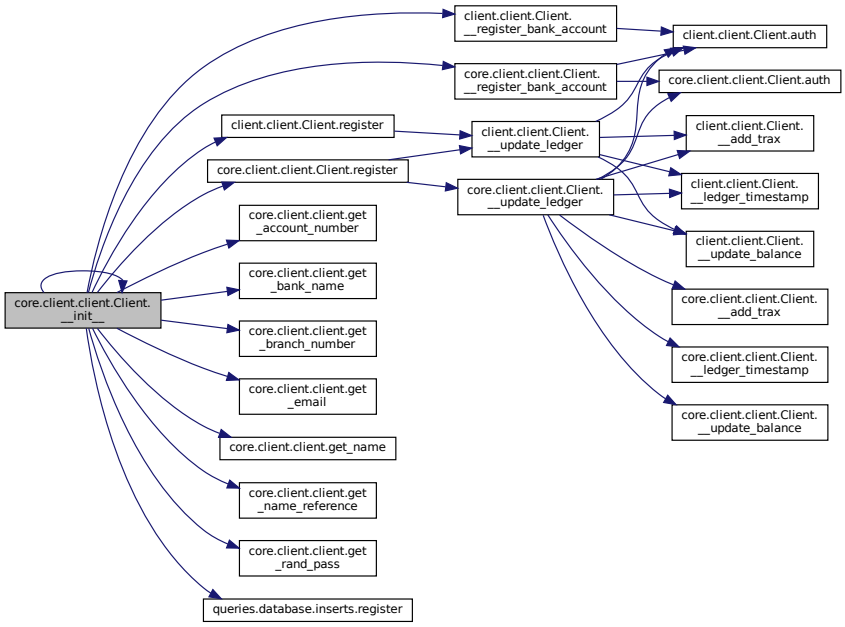
Definition at line 72 of file client.py.

```
72     def __init__(self):
73         seed=int.from_bytes(os.urandom(2), 'big')
74         self.faker = Faker()
75         random.seed(seed)
76         Faker.seed(seed)
77         #
78         self.name=get_name()
79         self.passcode=get_rand_pass()
80         self.email=get_email()
```

```
81          self.bank_name=get_bank_name()
82          self.branch_number=get_branch_number()
83          self.account_number=get_account_number()
84          self.name_reference=get_name_reference()
85          #
86          self.db = database(db_configs)
87          self.my_contacts = []
88          #Register user int he network
89          self.register()
90          #register user with bank account
91          self.__register_bank_account(self.bank_name, self.branch_number, self.account_number,
        self.name_reference)
92          logger.info("client initialized")
93          #self.all_goods = []
94          #self.__init_goods()
95          # credentials username:password
96
```

References core.client.client.Client.__init__(), client.client.Client.__register_bank_account(), core.client.client.↩ Client.__register_bank_account(), client.client.Client.account_number, core.client.client.Client.account_number, client.client.Client.bank_name, core.client.client.Client.bank_name, client.client.Client.branch_number, core.↩ client.client.Client.branch_number, client.client.Client.db, core.client.client.Client.db, client.client.Client.email, core.client.client.Client.email, client.client.Client.faker, core.client.client.Client.faker, core.client.client.get_↩ account_number(), core.client.client.get_bank_name(), core.client.client.get_branch_number(), core.client.client.↩ get_email(), core.client.client.get_name(), core.client.client.get_name_reference(), core.client.client.get_rand↩ _pass(), client.client.Client.my_contacts, core.client.client.Client.my_contacts, client.client.Client.name, core.↩ client.client.Client.name, client.client.Client.name_reference, core.client.client.Client.name_reference, core.↩ client.client.Client.passcode, client.client.Client.passcode, client.client.Client.register(), core.client.client.Client.↩ register(), and queries.database.inserts.register().

Here is the call graph for this function:



7.3.3 Member Function Documentation

7.3.3.1 __add_trax() [1/2]

```
def core.client.client.Client.__add_trax (
                self,
                trans )  [private]
```

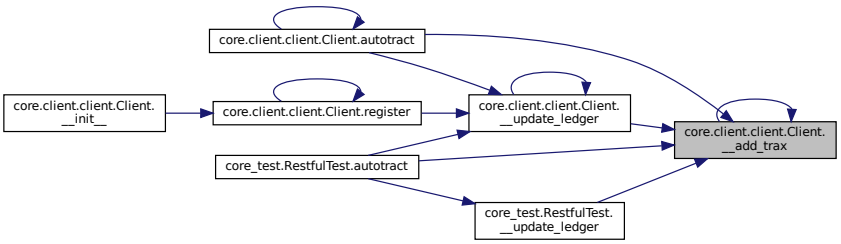
Definition at line 210 of file client.py.

```
210     def __add_trax(self, trans):
211         trans=trans['transactions'] #TODO (res)
212         self.db.init()
213         try:
214             print("trans: ", trans)
215             #self.db.inserts.insert_trx(trans["trx_src"], trans['trx_dest'], trans['trx_gid'])
216             self.db.inserts.insert_trx(trans["trx_src"], trans['trx_dest'], trans['trx_cost'])
217             self.db.commit()
218         except psycopg2.DatabaseError as error:
219             self.db.rollback()
220         finally:
221             self.db.close()
```

References client.client.Client.db, and core.client.client.Client.db.

Referenced by core.client.client.Client.__add_trax(), core.client.client.Client.__update_ledger(), core_test.Restful↩ Test.__update_ledger(), core.client.client.Client.autotract(), and core_test.RestfulTest.autotract().

Here is the caller graph for this function:



7.3.3.2 __add_trax() [2/2]

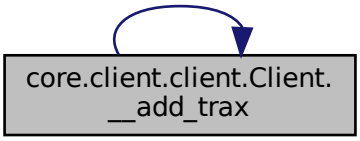
```
def core.client.client.Client.__add_trax (
                self,
                trans )  [private]
```

Definition at line 210 of file client.py.

```
210     def __add_trax(self, trans):
211         trans=trans['transactions'] #TODO (res)
212         self.db.init()
213         try:
214             print("trans: ", trans)
215             #self.db.inserts.insert_trx(trans["trx_src"], trans['trx_dest'], trans['trx_gid'])
216             self.db.inserts.insert_trx(trans["trx_src"], trans['trx_dest'], trans['trx_cost'])
217             self.db.commit()
218         except psycopg2.DatabaseError as error:
219             self.db.rollback()
220         finally:
221             self.db.close()
```

References core.client.client.Client.__add_trax(), core.client.client.Client.db, and client.client.Client.db.

Here is the call graph for this function:



7.3.3.3 __ledger_timestamp() [1/2]

```
def core.client.client.Client.__ledger_timestamp (
              self )  [private]
```

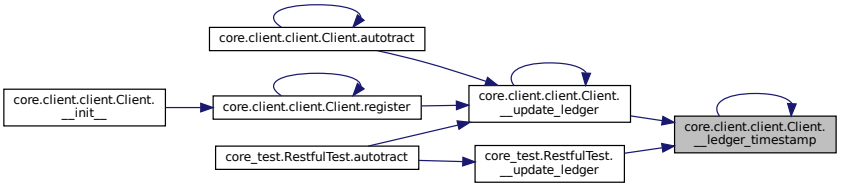
Definition at line 228 of file client.py.

```
228     def __ledger_timestamp(self):
229         dt=None
230         self.db.init()
231         try:
232             dt=self.db.gets.get_last_timestamp()
233             self.db.commmit()
234         except psycopg2.DatabaseError as error:
235             logger.critical("failed to retrieve ledger time stamp")
236             self.db.rollback()
237         finally:
238             self.db.close()
239         print("timestamp: ", dt)
240 
241 
242         return dt if not dt==None else datetime.datetime.now().strftime(TIMESTAMP_FORMAT)
```

References client.client.Client.db, and core.client.client.Client.db.

Referenced by core.client.client.Client.__ledger_timestamp(), core.client.client.Client.__update_ledger(), and core_test.RestfulTest.__update_ledger().

Here is the caller graph for this function:



7.3.3.4 __ledger_timestamp() [2/2]

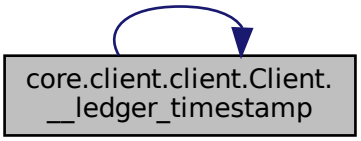
```
def core.client.client.Client.__ledger_timestamp (
              self )  [private]
```

Definition at line 228 of file client.py.

```
228     def __ledger_timestamp(self):
229         dt=None
230         self.db.init()
231         try:
232             dt=self.db.gets.get_last_timestamp()
233             self.db.commmit()
234         except psycopg2.DatabaseError as error:
235             logger.critical("failed to retrieve ledger time stamp")
236             self.db.rollback()
237         finally:
238             self.db.close()
239         print("timestamp: ", dt)
240 
241 
242         return dt if not dt==None else datetime.datetime.now().strftime(TIMESTAMP_FORMAT)
```

References core.client.client.Client.__ledger_timestamp(), core.client.client.Client.db, and client.client.Client.db.

Here is the call graph for this function:



7.3.3.5 __register_bank_account() [1/2]

```
def core.client.client.Client.__register_bank_account (
              self,
              bank_name = None,
              branch_number = None,
              account_number = None,
              name_reference = None )  [private]
```

check if this client has credentials, if not register

Definition at line 143 of file client.py.

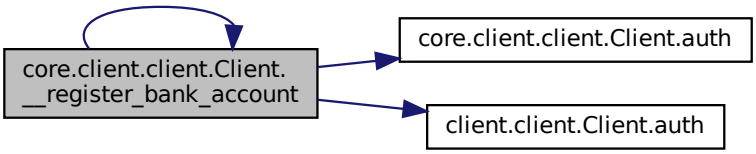
```
143     def __register_bank_account(self, bank_name=None, branch_number=None, account_number=None,
        name_reference=None):
144         """ check if this client has credentials, if not register
145 
146         """
147         #check if user have credentials
148         BRANCH_NUM_MAX=100
149         ACCOUNT_NUMBER_MAX=10000000000
150         self.db.init()
```

```
151         try:
152             #register user
153             bank_name = self.faker.name() if bank_name==None else bank_name
154             branch_number = random.random()*BRANCH_NUM_MAX if branch_number==None else branch_number
155             account_number= random.random()*ACCOUNT_NUMBER_MAX if account_number==None else
    account_number
156             name_reference= self.faker.name() if name_reference==None else name_reference
157             payload={'bank_name':bank_name,
158                      'branch_number':branch_number,
159                      'account_number':account_number,
160                      'name_reference':name_reference}
161             res=requests.post(ADD_BANK_ACCOUNT_URL, \
162                               data=json.dumps(payload), \
163                               auth=self.auth())
164             response = json.loads(res.text)
165             scode=res.status_code
166             #TODO support multiple accounts
167             #create local client table for banking,
168             #to insert balance, or each account.
169             print('response for balance:  ', response)
170             self.balance=response['balance']
171             self.db.commit()
172 
173         except psycopg2.DatabaseError as error:
174             logger.critical("bank account registration failed!, error: "+ str(error))
175             print("bank account register failed!, error: "+str(error))
176             self.db.rollback()
177         finally:
178             self.db.close()
```

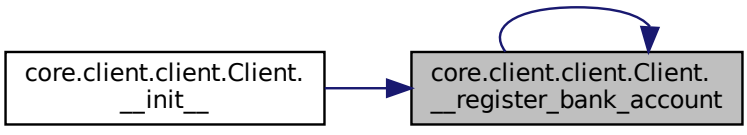
References client.client.Client.auth(), core.client.client.Client.auth(), client.client.Client.db, core.client.client.↩ Client.db, client.client.Client.faker, core.client.client.Client.faker, and core.client.client.Client.name.

Referenced by core.client.client.Client.__init__(), and core.client.client.Client.__register_bank_account().

Here is the call graph for this function:



Here is the caller graph for this function:



7.3.3.6 __register_bank_account() [2/2]

```
def core.client.client.Client.__register_bank_account (
              self,
              bank_name = None,
              branch_number = None,
              account_number = None,
              name_reference = None )  [private]
```

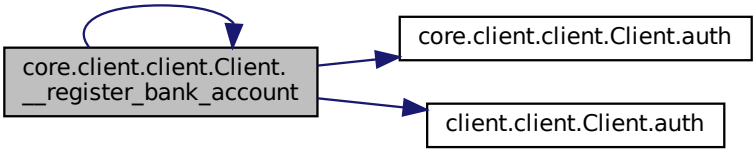
check if this client has credentials, if not register

Definition at line 143 of file client.py.

```
143     def __register_bank_account(self, bank_name=None, branch_number=None, account_number=None,
        name_reference=None):
144         """ check if this client has credentials, if not register
145
146         """
147         #check if user have credentials
148         BRANCH_NUM_MAX=100
149         ACCOUNT_NUMBER_MAX=10000000000
150         self.db.init()
151         try:
152             #register user
153             bank_name = self.faker.name() if bank_name==None else bank_name
154             branch_number = random.random()*BRANCH_NUM_MAX if branch_number==None else branch_number
155             account_number= random.random()*ACCOUNT_NUMBER_MAX if account_number==None else
        account_number
156             name_reference= self.faker.name() if name_reference==None else name_reference
157             payload={'bank_name':bank_name,
158                      'branch_number':branch_number,
159                      'account_number':account_number,
160                      'name_reference':name_reference}
161             res=requests.post(ADD_BANK_ACCOUNT_URL, \
162                               data=json.dumps(payload), \
163                               auth=self.auth())
164             response = json.loads(res.text)
165             scode=res.status_code
166             #TODO support multiple accounts
167             #create local client table for banking,
168             #to insert balance, or each account.
169             print('response for balance:  ', response)
170             self.balance=response['balance']
171             self.db.commit()
172
173         except psycopg2.DatabaseError as error:
174             logger.critical("bank account registration failed!, error: "+ str(error))
175             print("bank account register failed!, error: "+str(error))
176             self.db.rollback()
177         finally:
178             self.db.close()
```

References core.client.client.Client.__register_bank_account(), core.client.client.Client.auth(), client.client.Client.↩ auth(), client.client.Client.balance, core.client.client.Client.balance, client.client.Client.db, core.client.client.Client.↩ db, core.client.client.Client.faker, client.client.Client.faker, and core.client.client.Client.name.

Here is the call graph for this function:



7.3.3.7 __transact() [1/2]

```
def core.client.client.Client.__transact (
                self,
                credid,
                amount,
                currency = 'USD' )  [private]
```

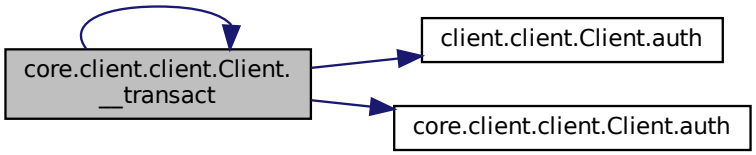
Definition at line 202 of file client.py.

```
202     def __transact(self, credid, amount, currency='USD'):
203         pur_item = {'credid': credid,
204                     'amount':amount,
205                     'currency': 'USD'}
206         ret=requests.post(PURCHASE_URL, data=json.dumps(pur_item), auth=self.auth())
207         response=ret.text#ret.text.decode('utf8')#.replace('"', "'")
208         trx=json.loads(response)
209         return trx
```

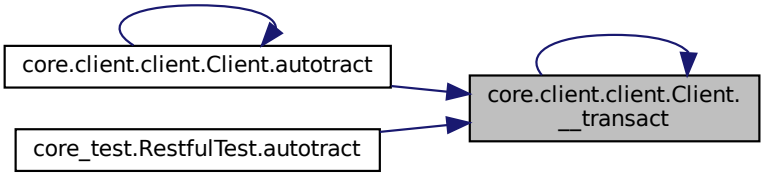
References client.client.Client.auth(), and core.client.client.Client.auth().

Referenced by core.client.client.Client.__transact(), core.client.client.Client.autotract(), and core_test.RestfulTest.↩ autotract().

Here is the call graph for this function:



Here is the caller graph for this function:



7.3.3.8 __transact() [2/2]

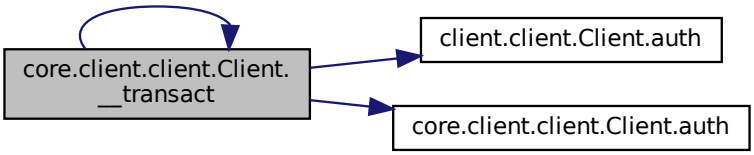
```
def core.client.client.Client.__transact (
              self,
              credid,
              amount,
              currency = 'USD' )  [private]
```

Definition at line 202 of file client.py.

```
202     def __transact(self, credid, amount, currency='USD'):
203         pur_item = {'credid': credid,
204                     'amount':amount,
205                     'currency': 'USD'}
206         ret=requests.post(PURCHASE_URL, data=json.dumps(pur_item), auth=self.auth())
207         response=ret.text#ret.text.decode('utf8')#.replace('"', "'")
208         trx=json.loads(response)
209         return trx
```

References core.client.client.Client.__transact(), core.client.client.Client.auth(), and client.client.Client.auth().

Here is the call graph for this function:



7.3.3.9 __update_balance() [1/2]

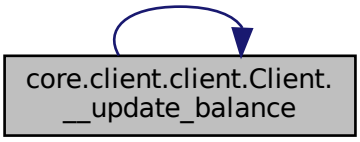
```
def core.client.client.Client.__update_balance (
              self,
              balance )  [private]
```

Definition at line 222 of file client.py.

```
222     def __update_balance(self, balance):
223         #TODO fix
224         if type(balance)==dict:
225             self.balance=balance['balance']
226             return
227         self.balance=balance
```

References core.client.client.Client.__update_balance(), core.client.client.Client.balance, client.client.Client.↩ balance, and core.client.client.type.

Here is the call graph for this function:



7.3.3.10 __update_balance() [2/2]

```
def core.client.client.Client.__update_balance (
            self,
            balance )  [private]
```

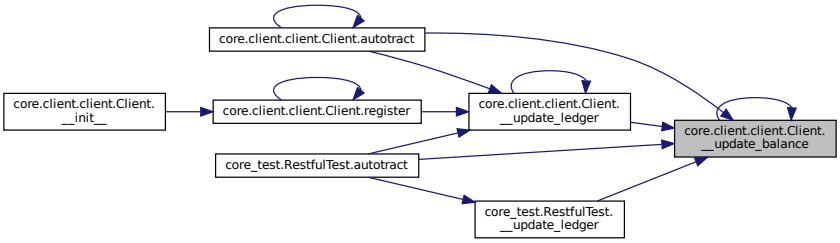
Definition at line 222 of file client.py.

```
222     def __update_balance(self, balance):
223         #TODO fix
224         if type(balance)==dict:
225             self.balance=balance['balance']
226             return
227         self.balance=balance
```

References client.client.Client.balance, core.client.client.Client.balance, and core.client.client.type.

Referenced by core.client.client.Client.__update_balance(), core.client.client.Client.__update_ledger(), core_↩ test.RestfulTest.__update_ledger(), core.client.client.Client.autotract(), and core_test.RestfulTest.autotract().

Here is the caller graph for this function:



7.3.3.11 __update_ledger() [1/2]

```
def core.client.client.Client.__update_ledger (
            self )  [private]
```

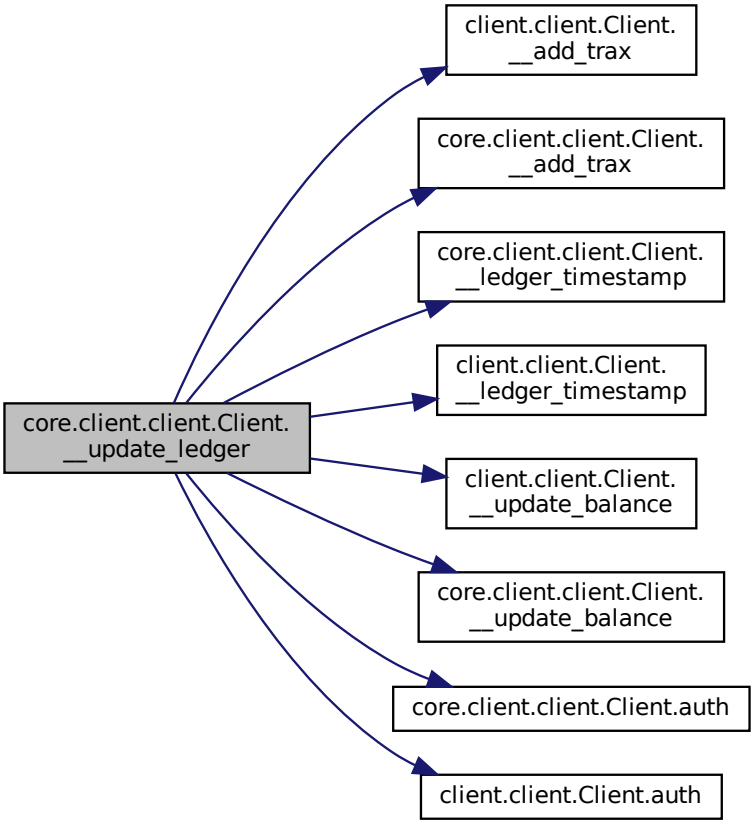
update ledger with sold goods, and update balance

Definition at line 243 of file client.py.

```
243     def __update_ledger(self):
244         """update ledger with sold goods, and update balance
245 
246         """
247         ret=requests.post(LEDGER_URL, data=json.dumps({'trx_dt': self.__ledger_timestamp()}),
    auth=self.auth())
248         #TODO always process the status code!
249         print('status code: ', ret.status_code)
250         res = json.loads(ret.text) #self.__get_dict(ret.text)
251         print('res: ', res)
252         print("new balance: ", res['balance'])
253         self.__update_balance(res['balance'])
254         self.db.init()
255         try:
256             new_trxs=res['transactions']
257             print("trxs: ", new_trxs)
258             transactions=pd.read_json(new_trxs)
259             print("trxs pandas: ", transactions)
260             if len(transactions)==0:
261                 logger.info("ledger is up to date!")
262                 return
263             for i in range(len(transactions)-1):
264                 self.__add_trax(transactions.iloc[i])
265             db.commit()
266         except psycopg2.DatabaseError as error:
267             print('FAILURE LEDGER UPDATE: ', str(error))
268             self.db.rollback()
269         except Exception as error:
270             print('FAILURE LEDGER UPDATE: ', str(error))
271             self.db.rollback()
272         finally:
273             self.db.close()
```

References client.client.Client.__add_trax(), core.client.client.Client.__add_trax(), client.client.Client.__ledger_timestamp(), core.client.client.Client.__ledger_timestamp(), client.client.Client.__update_balance(), core.client.client.Client.__update_balance(), core.client.client.Client.__update_ledger(), core.client.client.Client.auth(), client.client.Client.auth(), client.client.Client.db, and core.client.client.Client.db.

Here is the call graph for this function:



7.3.3.12 __update_ledger() [2/2]

```
def core.client.client.Client.__update_ledger (
              self )  [private]
```

update ledger with sold goods, and update balance

Definition at line 243 of file client.py.

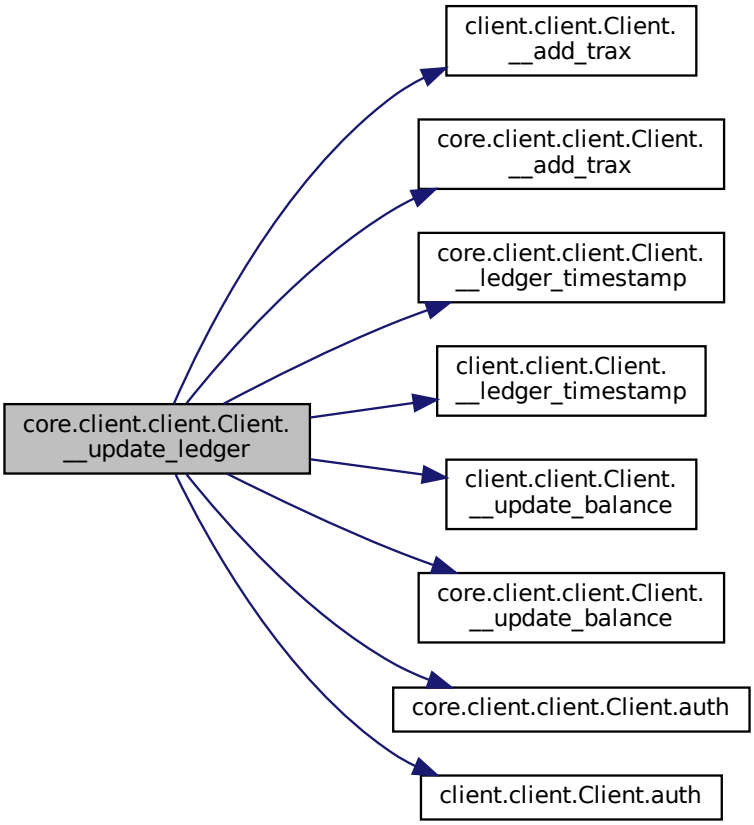
```
243     def __update_ledger(self):
244         """update ledger with sold goods, and update balance
245 
246         """
247         ret=requests.post(LEDGER_URL, data=json.dumps({'trx_dt': self.__ledger_timestamp()}),
        auth=self.auth())
248         #TODO always process the status code!
249         print('status code: ', ret.status_code)
250         res = json.loads(ret.text) #self.__get_dict(ret.text)
251         print('res: ', res)
252         print("new balance: ", res['balance'])
253         self.__update_balance(res['balance'])
254         self.db.init()
255         try:
```

```
256             new_trxs=res['transactions']
257             print("trxs: ", new_trxs)
258             transactions=pd.read_json(new_trxs)
259             print("trxs pandas: ", transactions)
260             if len(transactions)==0:
261                 logger.info("ledger is up to date!")
262                 return
263             for i in range(len(transactions)-1):
264                 self.__add_trax(transactions.iloc[i])
265             db.commit()
266         except psycopg2.DatabaseError as error:
267             print('FAILURE LEDGER UPDATE: ', str(error))
268             self.db.rollback()
269         except Exception as error:
270             print('FAILURE LEDGER UPDATE: ', str(error))
271             self.db.rollback()
272         finally:
273             self.db.close()
```

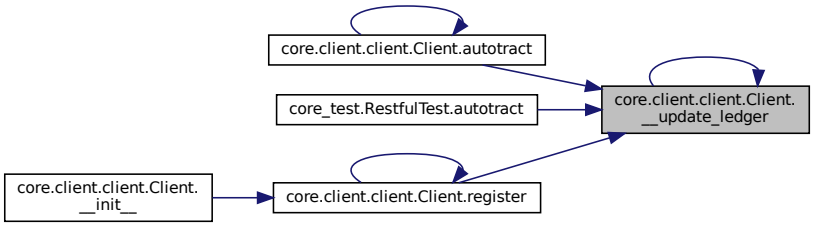
References client.client.Client.__add_trax(), core.client.client.Client.__add_trax(), client.client.Client.__ledger_↩ timestamp(), core.client.client.Client.__ledger_timestamp(), client.client.Client.__update_balance(), core.client.↩ client.Client.__update_balance(), core.client.client.Client.auth(), client.client.Client.auth(), client.client.Client.db, and core.client.client.Client.db.

Referenced by core.client.client.Client.__update_ledger(), core.client.client.Client.autotract(), core_test.Restful↩ Test.autotract(), and core.client.client.Client.register().

Here is the call graph for this function:



Here is the caller graph for this function:



7.3.3.13 add_contact() [1/2]

```
def core.client.client.Client.add_contact (
                self,
                email,
                name )


Fetch the client with given email/name

@param email: contact email
@param name: contact name
```

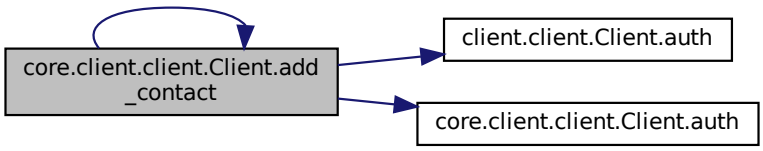
Definition at line 179 of file client.py.

```
179     def add_contact(self, email, name):
180         """Fetch the client with given email/name
181 
182         @param email: contact email
183         @param name: contact name
184         """
185         #get client credential id
186         cred={'email': email}
187         ret=requests.post(CONTACTS_URL, \
188                           data=json.dumps(cred), \
189                           auth=self.auth())
190         response=ret.text#.decode('utf8')#.replace('"', "'")
191         print('add contact response: ', response)
192         res = json.loads(response)
193         credid = res['credid']
194         self.db.init()
195         try:
196             self.db.inserts.insert_contact(email, name, credid)
197             self.db.commit()
198         except psycopg2.DatabaseError as error:
199             self.db.rollback()
200         finally:
201             self.db.close()
```

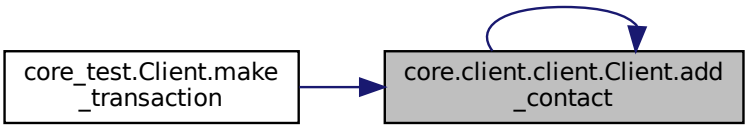
References client.client.Client.auth(), core.client.client.Client.auth(), client.client.Client.db, and core.client.client.↩ Client.db.

Referenced by core.client.client.Client.add_contact(), and core_test.Client.make_transaction().

Here is the call graph for this function:



Here is the caller graph for this function:



7.3.3.14 add_contact() [2/2]

```
def core.client.client.Client.add_contact (
              self,
              email,
              name )


Fetch the client with given email/name

@param email: contact email
@param name: contact name
```

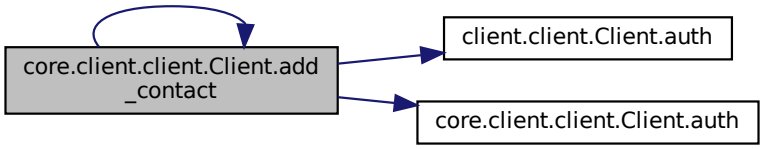
Definition at line 179 of file client.py.

```
179     def add_contact(self, email, name):
180         """Fetch the client with given email/name
181 
182         @param email: contact email
183         @param name: contact name
184         """
185         #get client credential id
186         cred={'email': email}
187         ret=requests.post(CONTACTS_URL, \
188                           data=json.dumps(cred), \
189                           auth=self.auth())
190         response=ret.text#.decode('utf8')#.replace('"', "'")
191         print('add contact response: ', response)
192         res = json.loads(response)
193         credid = res['credid']
194         self.db.init()
195         try:
```

```
196             self.db.inserts.insert_contact(email, name, credid)
197             self.db.commit()
198         except psycopg2.DatabaseError as error:
199             self.db.rollback()
200         finally:
201             self.db.close()
```

References core.client.client.Client.add_contact(), client.client.Client.auth(), core.client.client.Client.auth(), client.↩client.Client.db, and core.client.client.Client.db.

Here is the call graph for this function:



7.3.3.15 auth() [1/2]

```
def core.client.client.Client.auth (
                self )
```

Definition at line 97 of file client.py.

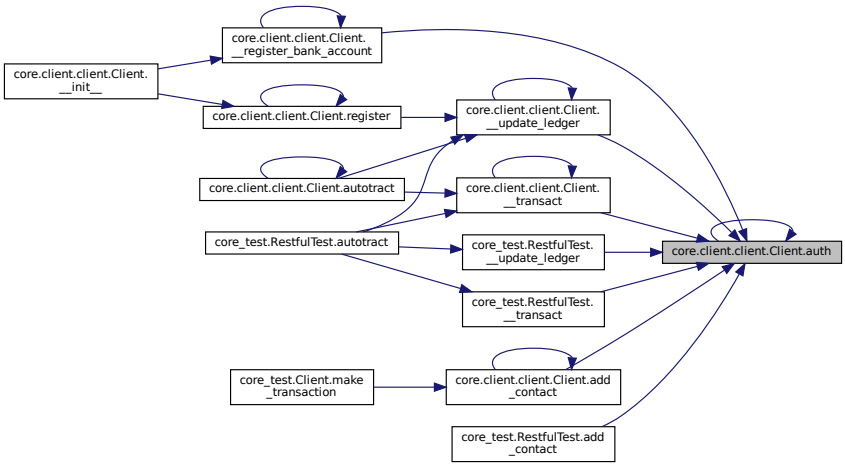
```
97     def auth(self):
98         logger.info("auth ("+str(self.email)+":"+str(self.passcode)+")")
99         return HTTPBasicAuth(self.email, self.passcode)
100
```

References core.client.client.Client.auth(), client.client.Client.email, core.client.client.Client.email, client.client.↩Client.passcode, and core.client.client.Client.passcode.

Referenced by core_test.RestfulTest.__transact(), core_test.RestfulTest.__update_ledger(), and core_test.↩RestfulTest.add_contact().

Here is the call graph for this function:

Here is the caller graph for this function:



7.3.3.16 auth() [2/2]

```
def core.client.client.Client.auth (
              self )
```

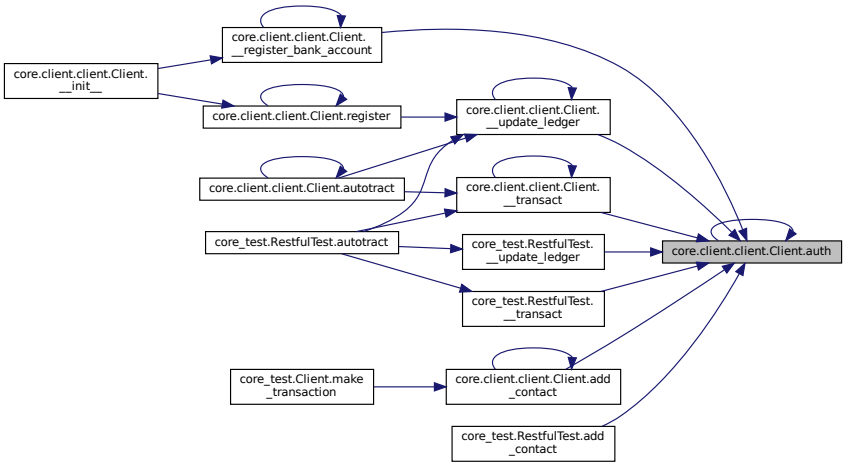
Definition at line 97 of file client.py.

```
97     def auth(self):
98         logger.info("auth ("+str(self.email)+":"+str(self.passcode)+")")
99         return HTTPBasicAuth(self.email, self.passcode)
100
```

References client.client.Client.email, core.client.client.Client.email, client.client.Client.passcode, and core.client.↩ client.Client.passcode.

Referenced by core.client.client.Client.__register_bank_account(), core.client.client.Client.__transact(), core_↩ test.RestfulTest.__transact(), core.client.client.Client.__update_ledger(), core_test.RestfulTest.__update_ledger(), core.client.client.Client.add_contact(), core_test.RestfulTest.add_contact(), and core.client.client.Client.auth().

Here is the caller graph for this function:



7.3.3.17 autotract() [1/2]

```
def core.client.client.Client.autotract (
              self )

Stochastic fake auto-tracting

trade with randomly with maintained contacts with 0.5 probability for each, and 0.1 probability for all contac
```

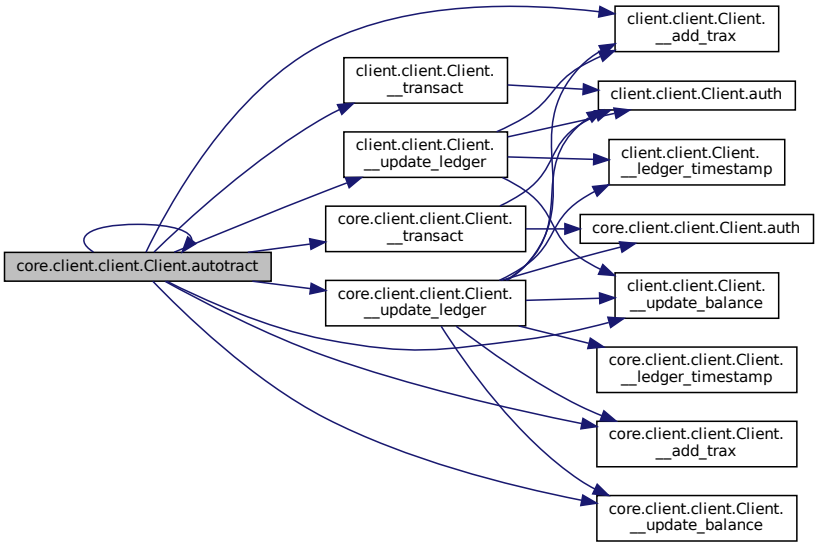
Definition at line 274 of file client.py.

```
274     def autotract(self):
275         """Stochastic fake auto-tracting
276 
277         trade with randomly with maintained contacts with 0.5 probability for each, and 0.1 probability
      for all contacts goods, update balance, and goods for each contact

278         """
279         #update balance, and add new transactions
280         MAX_TRACT_BALANCE=100000
281         self.__update_ledger()
282         for i in range(len(contacts_df-1)):
283             contact_credid=contacts_df.iloc[i]['contact_id']
284             contact_name=contacts_df.iloc[i]['contact_name']
285             contact_email=contacts_df.iloc[i]['contact_email']
286             amount=random.random()*MAX_TRACT_BALANCE
287             logger.info("making transaction with {}[{}] by amount: {}".\
288                         format(contact_name, contact_email, amount))
289             #if random.random()> STOCHASTIC_TRADE_THRESHOLD and good.cost <= self.balance:
290             #TODO (fix) doesn't work!
291             #trx=self.__purchase(good.gid)
292             trx=self.__transact(contact_credid, amount)
293             self.__add_trax(trx)
294             self.__update_balance(trx)
```

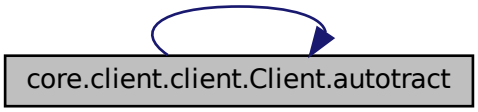
References client.client.Client.__add_trax(), core.client.client.Client.__add_trax(), core.client.client.Client.__transact(), client.client.Client.__transact(), client.client.Client.__update_balance(), core.client.client.Client.__update_balance(), core.client.client.Client.__update_ledger(), client.client.Client.__update_ledger(), and core.client.client.format.

Referenced by core.client.client.Client.autotract().

Here is the call graph for this function:



Here is the caller graph for this function:



7.3.3.18 autotract() [2/2]

```
def core.client.client.Client.autotract (
              self )
```

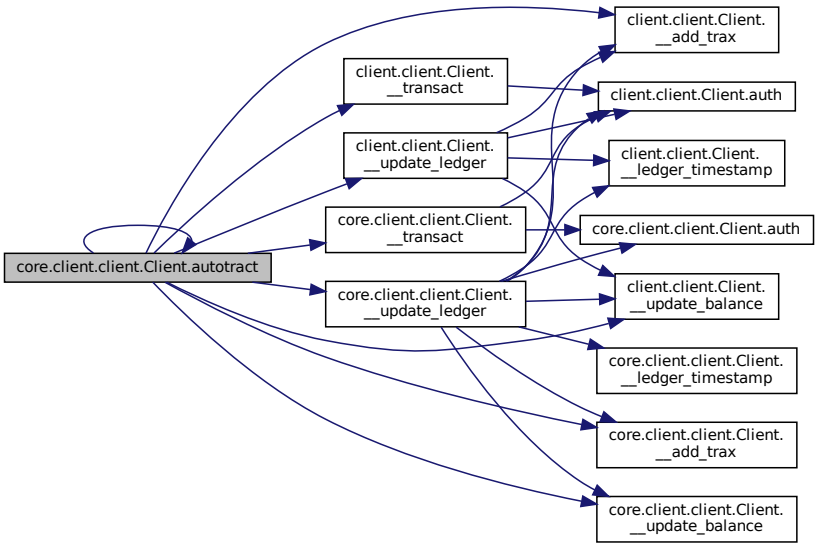
```
Stochastic fake auto-tracting

trade with randomly with maintained contacts with 0.5 probability for each, and 0.1 probability for all contac
```

Definition at line 274 of file client.py.

```
274     def autotract(self):
275         """Stochastic fake auto-tracting
276 
277         trade with randomly with maintained contacts with 0.5 probability for each, and 0.1 probability
       for all contacts goods, update balance, and goods for each contact

278         """
279         #update balance, and add new transactions
280         MAX_TRACT_BALANCE=100000
281         self.__update_ledger()
282         for i in range(len(contacts_df-1)):
283             contact_credid=contacts_df.iloc[i]['contact_id']
284             contact_name=contacts_df.iloc[i]['contact_name']
285             contact_email=contacts_df.iloc[i]['contact_email']
286             amount=random.random()*MAX_TRACT_BALANCE
287             logger.info("making transaction with {}[{}] by amount: {}".\
288                         format(contact_name, contact_email, amount))
289             #if random.random()> STOCHASTIC_TRADE_THRESHOLD and good.cost <= self.balance:
290             #TODO (fix) doesn't work!
291             #trx=self.__purchase(good.gid)
292             trx=self.__transact(contact_credid, amount)
293             self.__add_trax(trx)
294             self.__update_balance(trx)
```

References client.client.Client.__add_trax(), core.client.client.Client.__add_trax(), client.client.Client.__transact(), core.client.client.Client.__transact(), client.client.Client.__update_balance(), core.client.client.Client.__update_balance(), client.client.Client.__update_ledger(), core.client.client.Client.__update_ledger(), core.client.client.Client.autotract(), and core.client.client.format.

Here is the call graph for this function:



7.3.3.19 register() [1/2]

```
def core.client.client.Client.register (
              self )
```

check if this client has credentials, if not register

Definition at line 101 of file client.py.

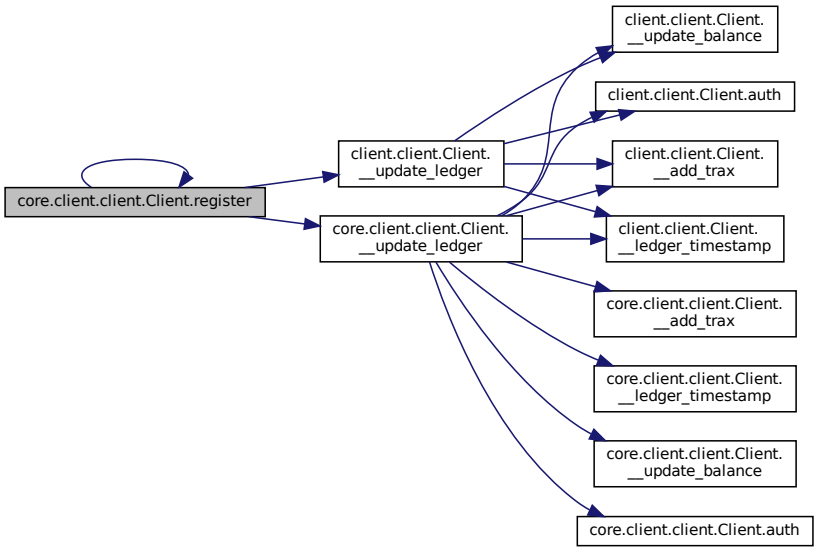
```
101     def register(self):
102         """ check if this client has credentials, if not register
103 
104         """
105         #check if user have credentials
106         self.db.init()
107         try:
108             if not self.db.exists.credential_exists(1):
109                 #register user
110                 #my_goods = [good(self.faker) for i in range(math.ceil(random.random()*MAX_GOODS))]
111                 #TODO check repose status_code makes sure it's request.codes.ok
112                 payload={'name':self.name, \
113                          'email': self.email, \
114                          'passcode': self.passcode}
115                 print('payload: ', payload)
116                 res=requests.post(REGISTER_URL, data=json.dumps(payload))
117                 response = json.loads(res.text)
118                 assert res.status_code==201, "status code is error {} ".format(res.status_code)
119                 print(response)
120                 credid=response['cred_id']
121                 #cid set to 0, since it never matters in the client side
122 
123                 self.db.inserts.register(0, self.passcode, credid)
124                 logger.debug("user registered with credentials username: {}, email: {}, passcode: {}"\
125                              .format(name, email, passcode))
126 
132             else:
133                 self.__update_ledger()
134             self.db.commit()
135             self.uname=email
```

```
136             self.pas=passcode
137         except psycopg2.DatabaseError as error:
138             logger.critical("registration failed!, error: "+ str(error))
139             print("register failed!, error: "+str(error))
140             self.db.rollback()
141         finally:
142             self.db.close()
```

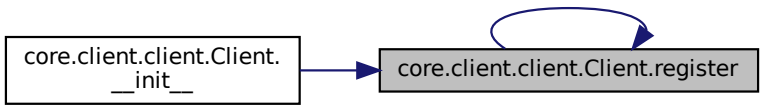
References client.client.Client.__update_ledger(), core.client.client.Client.__update_ledger(), client.client.Client.↩ db, core.client.client.Client.db, client.client.Client.email, core.client.client.Client.email, core.client.client.format, core.client.client.Client.name, client.client.Client.name, client.client.Client.passcode, and core.client.client.Client.↩ passcode.

Referenced by core.client.client.Client.__init__(), and core.client.client.Client.register().

Here is the call graph for this function:



Here is the caller graph for this function:



7.3.3.20 register() [2/2]

```
def core.client.client.Client.register (
            self )
```

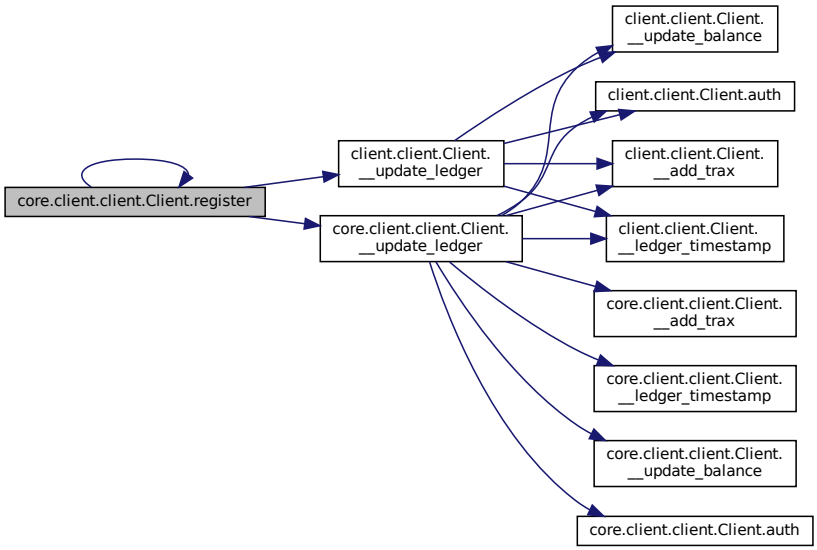
check if this client has credentials, if not register

Definition at line 101 of file client.py.

```
101     def register(self):
102         """ check if this client has credentials, if not register
103 
104         """
105         #check if user have credentials
106         self.db.init()
107         try:
108             if not self.db.exists.credential_exists(1):
109                 #register user
110                 #my_goods = [good(self.faker) for i in range(math.ceil(random.random()*MAX_GOODS))]
111                 #TODO check repose status_code makes sure it's request.codes.ok
112                 payload={'name':self.name, \
113                          'email': self.email, \
114                          'passcode': self.passcode}
115                 print('payload: ', payload)
116                 res=requests.post(REGISTER_URL, data=json.dumps(payload))
117                 response = json.loads(res.text)
118                 assert res.status_code==201, "status code is error {} ".format(res.status_code)
119                 print(response)
120                 credid=response['cred_id']
121                 #cid set to 0, since it never matters in the client side
122 
123                 self.db.inserts.register(0, self.passcode, credid)
124                 logger.debug("user registered with credentials username: {}, email: {}, passcode: {}"\
125                              .format(name, email, passcode))
126 
132             else:
133                 self.__update_ledger()
134             self.db.commit()
135             self.uname=email
136             self.pas=passcode
137         except psycopg2.DatabaseError as error:
138             logger.critical("registration failed!, error: "+ str(error))
139             print("register failed!, error: "+str(error))
140             self.db.rollback()
141         finally:
142             self.db.close()
```

References client.client.Client.__update_ledger(), core.client.client.Client.__update_ledger(), client.client.Client.db, core.client.client.Client.db, client.client.Client.email, core.client.client.Client.email, core.client.client.format, client.client.Client.name, core.client.client.Client.name, core.client.client.Client.pas, client.client.Client.pas, core.client.client.Client.passcode, client.client.Client.passcode, core.client.client.Client.register(), client.client.Client.uname, and core.client.client.Client.uname.

Here is the call graph for this function:



7.3.4 Field Documentation

7.3.4.1 account_number

`core.client.client.Client.account_number`

Definition at line 83 of file client.py.

Referenced by core.client.client.Client.__init__(), core.utils.PaymentGate.__init__(), and core_test.Client.register↩
_bank_account().

7.3.4.2 balance

`core.client.client.Client.balance`

Definition at line 170 of file client.py.

Referenced by core.client.client.Client.__register_bank_account(), core.client.client.Client.__update_balance(), core_test.RestfulTest.__update_balance(), core.utils.PaymentGate.authenticated(), core.utils.PaymentGate.get_↩
balance(), and core_test.Client.get_balance().

7.3.4.3 bank_name

core.client.client.Client.bank_name

Definition at line 81 of file client.py.

Referenced by core.client.client.Client.__init__(), core.utils.PaymentGate.__init__(), and core_test.Client.register_bank_account().

7.3.4.4 branch_number

core.client.client.Client.branch_number

Definition at line 82 of file client.py.

Referenced by core.client.client.Client.__init__(), core.utils.PaymentGate.__init__(), and core_test.Client.register_bank_account().

7.3.4.5 db

core.client.client.Client.db

Definition at line 86 of file client.py.

Referenced by core.client.client.Client.__add_trax(), core_test.RestfulTest.__add_trax(), core.client.client.Client.__init__(), core.client.client.Client.__ledger_timestamp(), core_test.RestfulTest.__ledger_timestamp(), core.client.client.Client.__register_bank_account(), core_test.RestfulTest.__register_bank_account(), core.client.client.Client.__update_ledger(), core_test.RestfulTest.__update_ledger(), core.client.client.Client.add_contact(), core_test.RestfulTest.add_contact(), and core.client.client.Client.register().

7.3.4.6 email

core.client.client.Client.email

Definition at line 80 of file client.py.

Referenced by core.client.client.Client.__init__(), core.client.client.Client.auth(), core_test.Client.basic_auth(), core_test.Client.get_balance(), core_test.Client.register(), and core.client.client.Client.register().

7.3.4.7 faker

core.client.client.Client.faker

Definition at line 74 of file client.py.

Referenced by core.client.client.Client.__init__(), core.client.client.Client.__register_bank_account(), and core_test.RestfulTest.__register_bank_account().

7.3.4.8 my_contacts

core.client.client.Client.my_contacts

Definition at line 87 of file client.py.

Referenced by core.client.client.Client.__init__().

7.3.4.9 name

core.client.client.Client.name

Definition at line 78 of file client.py.

Referenced by core.client.client.Client.__init__(), core.client.client.Client.__register_bank_account(), core_test.Client.get_balance(), core_test.Client.register(), and core.client.client.Client.register().

7.3.4.10 name_reference

core.client.client.Client.name_reference

Definition at line 84 of file client.py.

Referenced by core.client.client.Client.__init__(), core.utils.PaymentGate.__init__(), and core_test.Client.register_bank_account().

7.3.4.11 pas

core.client.client.Client.pas

Definition at line 136 of file client.py.

Referenced by core_test.RestfulTest.__auth(), core.client.client.Client.register(), and core_test.RestfulTest.test_register().

7.3.4.12 passcode

`core.client.client.Client.passcode`

Definition at line 79 of file client.py.

Referenced by core.client.client.Client.__init__(), core.client.client.Client.auth(), core_test.Client.basic_auth(), core_test.Client.get_balance(), core_test.Client.register(), and core.client.client.Client.register().

7.3.4.13 uname

`core.client.client.Client.uname`

TODO credid.

post goods #

payload=json.dumps({'goods': [(g.name, g.cost, g.quality) for g in my_goods]}) requests.post(GOODS_URL, data=payload, auth=self.auth()) TODO assert that requests.text is equivalent to payload

Definition at line 135 of file client.py.

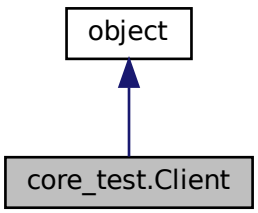
Referenced by core_test.RestfulTest.__auth(), core.client.client.Client.register(), and core_test.RestfulTest.test_↩ register().

The documentation for this class was generated from the following file:

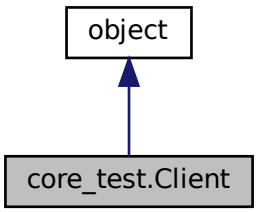
- core/client/client.py

7.4 core_test.Client Class Reference

Inheritance diagram for core_test.Client:



Collaboration diagram for core_test.Client:



Public Member Functions

- def __init__ (self, app)
- def basic_auth (self)
- def headers (self)
- def register (self)
- def register_bank_account (self)
- def add_contact (self, email, name)
- def get_balance (self)
- def make_transaction (self, email, name, amount)

Data Fields

- faker
- app
- name
- passcode
- email
- bank_name
- branch_number
- account_number
- name_reference
- currency_pref
- db
- my_contacts
- credid
- balance

7.4.1 Detailed Description

Definition at line 60 of file core_test.py.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 __init__()

```
def core_test.Client.__init__ (
               self,
               app )
```

Definition at line 61 of file core_test.py.

```
61     def __init__(self, app):
62         seed=int.from_bytes(os.urandom(2), 'big')
63         self.faker = Faker()
64         random.seed(seed)
65         Faker.seed(seed)
66         #
67         self.app=app
68         #
69         self.name=get_name()
70         self.passcode=get_rand_pass()
71         self.email=get_email()
72         self.bank_name=get_bank_name()
73         self.branch_number=get_branch_number()
74         self.account_number=get_account_number()
75         self.name_reference=get_name_reference()
76         self.currency_pref=EUR
77         #
78         self.db = database(db_configs)
79         self.my_contacts = []
80         # register
81         self.register()
82         self.register_bank_account()
83         logger.info("client initialized")
84
```

7.4.3 Member Function Documentation

7.4.3.1 add_contact()

```
def core_test.Client.add_contact (
               self,
               email,
               name )


Fetch the client with given email/name

@param email: contact email
@param name: contact name
```

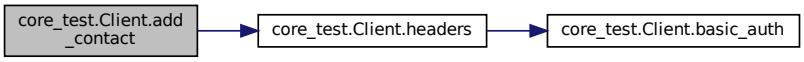
Definition at line 133 of file core_test.py.

```
133     def add_contact(self, email, name):
134         """Fetch the client with given email/name
135
136         @param email: contact email
137         @param name: contact name
138         """
139         #get client credential id
140         cred={'email': email}
141         ret=self.app.post(CONTACTS_URL, \
142                           data=json.dumps(cred), \
143                           headers=self.headers())
144         assert ret.status_code==201, 'adding contact failed!'
145         response=ret.json#json.loads(ret.data)
146         print('add contact response: ', response)
147         return response['credid']
```

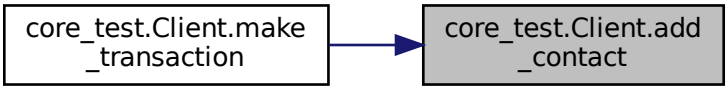
References core_test.Client.app, and core_test.Client.headers().

Referenced by core_test.Client.make_transaction().

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.3.2 basic_auth()

```
def core_test.Client.basic_auth (
                self )
```

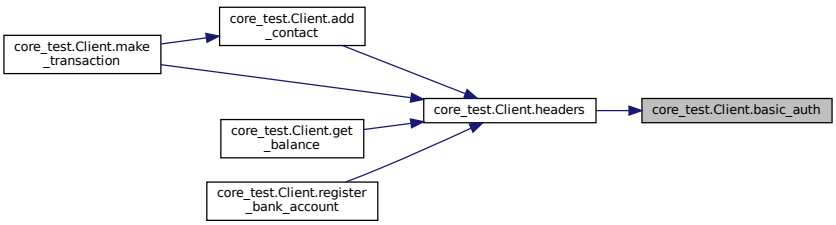
Definition at line 90 of file core_test.py.

```
90     def basic_auth(self):
91         #using the emails doesn't succeed! i will try the byname instead
92         valid_cred = base64.b64encode(bytes("{}:{}".format(self.email, self.passcode),
       'utf-8')).decode("utf-8")
93         #valid_cred = base64.b64encode(bytes("{}:{}".format(self.name, self.passcode),
       'utf-8')).decode("utf-8")
94         return valid_cred
```

References core_test.Client.email, client.client.Client.email, core.client.client.Client.email, core_test.format, core↩
_test.Client.passcode, core.client.client.Client.passcode, and client.client.Client.passcode.

Referenced by core_test.Client.headers().

Here is the caller graph for this function:



7.4.3.3 get_balance()

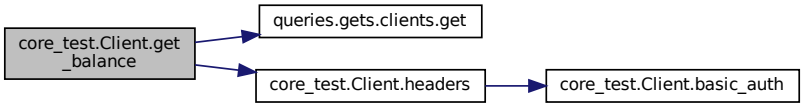
```
def core_test.Client.get_balance (
               self )
```

Definition at line 148 of file core_test.py.

```
148     def get_balance(self):
149         logger.info('verifying balance for {}/{} + {} with balance {}'.format(self.name, self.email,
        self.passcode, self.balance))
150         res=self.app.get(BALANCE,headers=self.headers())
151         assert res.status_code== 201
152         balance = res.json['balance']
153         base = res.json['base']
154         return balance, base
```

References core_test.Client.app, core_test.Client.balance, core.client.client.Client.balance, client.client.Client.↩ balance, core_test.Client.email, client.client.Client.email, core.client.client.Client.email, core_test.format, queries.↩ gets.clients.get(), core_test.Client.headers(), core_test.Client.name, client.client.Client.name, core.client.client.↩ Client.name, core_test.Client.passcode, client.client.Client.passcode, and core.client.client.Client.passcode.

Here is the call graph for this function:



7.4.3.4 headers()

```
def core_test.Client.headers (
               self )
```

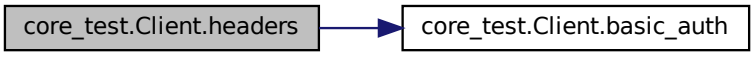
Definition at line 95 of file core_test.py.

```
95     def headers(self):
96         return {"Authorization": "Basic "+self.basic_auth()}
```

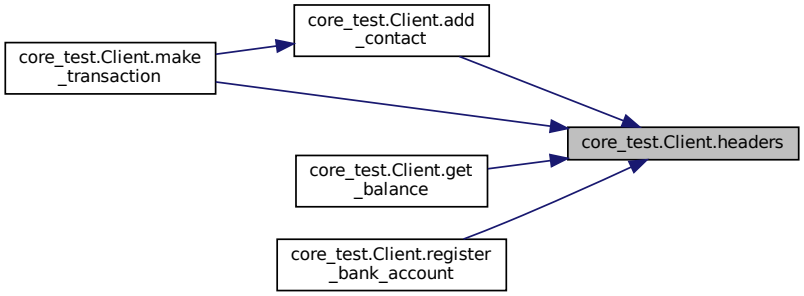
References core_test.Client.basic_auth().

Referenced by core_test.Client.add_contact(), core_test.Client.get_balance(), core_test.Client.make_transaction(), and core_test.Client.register_bank_account().

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.3.5 make_transaction()

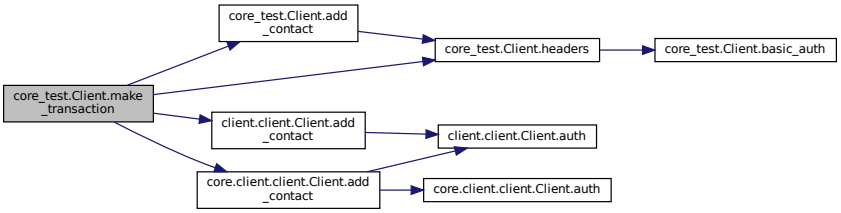
```
def core_test.Client.make_transaction (
                self,
                email,
                name,
                amount )
```

Definition at line 156 of file core_test.py.

```
156     def make_transaction(self, email, name, amount):
157         credid=self.add_contact(email, name)
158         payload={'credid':credid,\
159                  'amount': amount}
160         res=self.app.post(TRANSACTION,\
161                           data=json.dumps(payload),\
162                           headers=self.headers())
163         response=res.json
164         assert res.status_code==201
165         print('make transaction response: {}'.format(response))
166         balance_eq=response['balance']
167         trxs=response['transactions']
168         return (balance_eq, trxs)
169
```

References core_test.Client.add_contact(), core.client.client.Client.add_contact(), client.client.Client.add_contact(), core_test.Client.app, core_test.format, and core_test.Client.headers().

Here is the call graph for this function:



7.4.3.6 register()

```
def core_test.Client.register (
              self )
```

check if this client has credentials, if not register

Definition at line 97 of file core_test.py.

```
97     def register(self):
98         """ check if this client has credentials, if not register
99 
100         """
101         payload={'name':self.name, \
102                  'email': self.email, \
103                  'passcode': self.passcode,\
104                  'cur_pref':self.currency_pref}
105         print('payload: ', payload)
106         res=self.app.post(REGISTER_URL, data=json.dumps(payload))
107         #response = json.loads(res.text)
108         response = res.json#json.loads(res.data)
109         print(response)
110         assert res.status_code==201, "status code is error {} ".format(res.status_code)
111         credid=response['cred_id']
112         self.credid=credid
113         logger.debug("user registered with credentials username: {}, email: {}, passcode:
       {}".format(self.name, self.email, self.passcode))
114 
```

References core_test.Client.app, core_test.Client.currency_pref, core_test.Client.email, core.client.client.Client.email, client.client.Client.email, core_test.format, core_test.Client.name, core.client.client.Client.name, client.client.Client.name, core_test.Client.passcode, client.client.Client.passcode, and core.client.client.Client.passcode.

7.4.3.7 register_bank_account()

```
def core_test.Client.register_bank_account (
              self )
```

check if this client has credentials, if not register

Definition at line 115 of file core_test.py.

```
115     def register_bank_account(self):
116         """ check if this client has credentials, if not register
117 
118         """
119         #check if user have credentials
120         payload={'bank_name':self.bank_name, \
121                  'branch_number':self.branch_number, \
122                  'account_number':self.account_number,\
123                  'name_reference':self.name_reference}
124         res=self.app.post(ADD_BANK_ACCOUNT_URL, \
125                           data=json.dumps(payload), \
126                           headers=self.headers())
127         #response = json.loads(res.text)
128         response = res.json#json.loads(res.data)
129         print('response', response)
130         assert res.status_code == 201, 'registering bank account failed!'
131         self.balance=response['balance']
132 
```

References core_test.Client.account_number, client.client.Client.account_number, core.client.client.Client.account_number, core_test.Client.app, core_test.Client.bank_name, core.client.client.Client.bank_name, client.client.Client.bank_name, core_test.Client.branch_number, client.client.Client.branch_number, core.client.client.Client.branch_number, core_test.Client.headers(), core_test.Client.name_reference, core.client.client.Client.name_reference, and client.client.Client.name_reference.

Here is the call graph for this function:

7.4.4 Field Documentation

7.4.4.1 account_number

`core_test.Client.account_number`

Definition at line 74 of file core_test.py.

Referenced by core.utils.PaymentGate.__init__(), and core_test.Client.register_bank_account().

7.4.4.2 app

`core_test.Client.app`

Definition at line 67 of file core_test.py.

Referenced by core_test.Client.add_contact(), core_test.Client.get_balance(), core_test.Client.make_transaction(), core_test.Client.register(), core_test.Client.register_bank_account(), core_test.RestfulTest.test_add_contact(), core_test.RestfulTest.test_get_balance(), and core_test.RestfulTest.test_register().

7.4.4.3 balance

`core_test.Client.balance`

Definition at line 131 of file core_test.py.

Referenced by core_test.RestfulTest.__update_balance(), core.utils.PaymentGate.authenticated(), core.utils.↩ PaymentGate.get_balance(), and core_test.Client.get_balance().

7.4.4.4 bank_name

core_test.Client.bank_name

Definition at line 72 of file core_test.py.

Referenced by core.utils.PaymentGate.__init__(), and core_test.Client.register_bank_account().

7.4.4.5 branch_number

core_test.Client.branch_number

Definition at line 73 of file core_test.py.

Referenced by core.utils.PaymentGate.__init__(), and core_test.Client.register_bank_account().

7.4.4.6 credid

core_test.Client.credid

Definition at line 112 of file core_test.py.

7.4.4.7 currency_pref

core_test.Client.currency_pref

Definition at line 76 of file core_test.py.

Referenced by core_test.Client.register().

7.4.4.8 db

core_test.Client.db

Definition at line 78 of file core_test.py.

Referenced by core_test.RestfulTest.__add_trax(), core_test.RestfulTest.__ledger_timestamp(), core_test.↩ RestfulTest.__register_bank_account(), core_test.RestfulTest.__update_ledger(), and core_test.RestfulTest.↩ add_contact().

7.4.4.9 email

core_test.Client.email

Definition at line 71 of file core_test.py.

Referenced by core_test.Client.basic_auth(), core_test.Client.get_balance(), and core_test.Client.register().

7.4.4.10 faker

core_test.Client.faker

Definition at line 63 of file core_test.py.

Referenced by core_test.RestfulTest.__register_bank_account().

7.4.4.11 my_contacts

core_test.Client.my_contacts

Definition at line 79 of file core_test.py.

7.4.4.12 name

core_test.Client.name

Definition at line 69 of file core_test.py.

Referenced by core_test.Client.get_balance(), and core_test.Client.register().

7.4.4.13 name_reference

core_test.Client.name_reference

Definition at line 75 of file core_test.py.

Referenced by core.utils.PaymentGate.__init__(), and core_test.Client.register_bank_account().

7.4.4.14 passcode

`core_test.Client.passcode`

Definition at line 70 of file core_test.py.

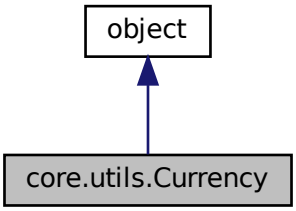
Referenced by core_test.Client.basic_auth(), core_test.Client.get_balance(), and core_test.Client.register().

The documentation for this class was generated from the following file:

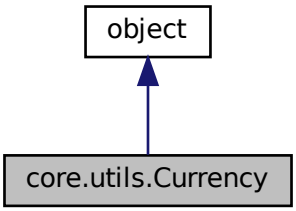
- core/tests/core_test.py

7.5 core.utils.Currency Class Reference

Inheritance diagram for core.utils.Currency:



Collaboration diagram for core.utils.Currency:



Public Member Functions

- def __init__ (self, preference, base=EUR)
- def valid (self)
- def exchange (self, amount=1)
- def exchange_back (self, amount=1)
- def __init__ (self, preference, base=EUR)
- def valid (self)
- def exchange (self, amount=1)
- def exchange_back (self, amount=1)

Data Fields

- base
- pref
- rate

7.5.1 Detailed Description

Definition at line 94 of file utils.py.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 __init__() [1/2]

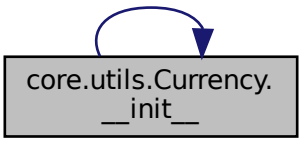
```
def core.utils.Currency.__init__ (
                self,
                preference,
                base = EUR )
```

Definition at line 95 of file utils.py.

```
95     def __init__(self, preference, base=EUR):
96         self.base=base
97         self.pref=preference
98         self.rate = exchange(self.base, self.pref)
99         log.info("exchange with rate {}".format(self.rate))
100         if self.rate==0:
101             log.critical("currence base {}, pref {}".format(self.base, self.pref))
102
```

Referenced by core.utils.Currency.__init__().

Here is the caller graph for this function:



7.5.2.2 __init__() [2/2]

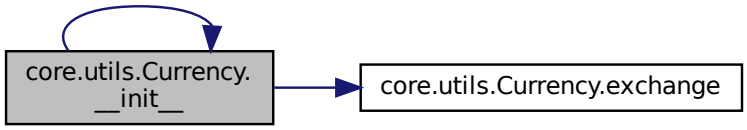
```
def core.utils.Currency.__init__ (
               self,
               preference,
               base = EUR )
```

Definition at line 95 of file utils.py.

```
95     def __init__(self, preference, base=EUR):
96         self.base=base
97         self.pref=preference
98         self.rate = exchange(self.base, self.pref)
99         log.info("exchange with rate {}".format(self.rate))
100         if self.rate==0:
101             log.critical("currence base {}, pref {}".format(self.base, self.pref))
102
```

References core.utils.Currency.__init__(), core.utils.Currency.base, core.utils.Currency.exchange(), core.utils.↩ format, core.utils.Currency.pref, and core.utils.Currency.rate.

Here is the call graph for this function:



7.5.3 Member Function Documentation

7.5.3.1 exchange() [1/2]

```
def core.utils.Currency.exchange (
               self,
               amount = 1 )
```

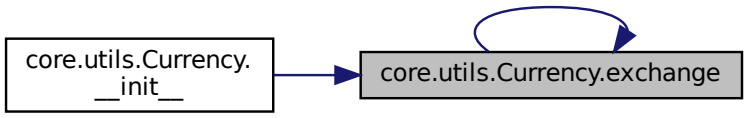
Definition at line 107 of file utils.py.

```
107     def exchange(self, amount=1):
108         print("making exchange with rate {}".format(self.rate))
109
110         return amount*self.rate
```

References core.utils.format, and core.utils.Currency.rate.

Referenced by core.utils.Currency.__init__(), and core.utils.Currency.exchange().

Here is the caller graph for this function:



7.5.3.2 exchange() [2/2]

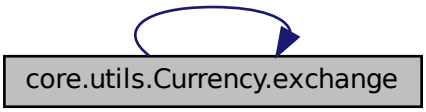
```
def core.utils.Currency.exchange (
                self,
                amount = 1 )
```

Definition at line 107 of file utils.py.

```
107     def exchange(self, amount=1):
108         print("making exchange with rate {}".format(self.rate))
109 
110         return amount*self.rate
```

References core.utils.Currency.exchange(), core.utils.format, and core.utils.Currency.rate.

Here is the call graph for this function:



7.5.3.3 exchange_back() [1/2]

```
def core.utils.Currency.exchange_back (
                self,
                amount = 1 )
```

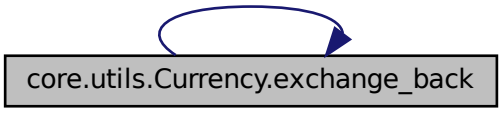
Definition at line 111 of file utils.py.

```
111     def exchange_back(self, amount=1):
112         print("making back exchange with rate {}".format(self.rate))
113         return amount/self.rate
```

```
114
```

References core.utils.Currency.exchange_back(), core.utils.format, and core.utils.Currency.rate.

Here is the call graph for this function:



7.5.3.4 exchange_back() [2/2]

```
def core.utils.Currency.exchange_back (
              self,
              amount = 1 )
```

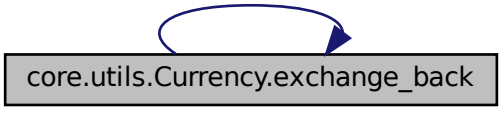
Definition at line 111 of file utils.py.

```
111     def exchange_back(self, amount=1):
112         print("making back exchange with rate {}".format(self.rate))
113         return amount/self.rate
114
```

References core.utils.format, and core.utils.Currency.rate.

Referenced by core.utils.Currency.exchange_back().

Here is the caller graph for this function:



7.5.3.5 valid() [1/2]

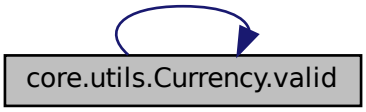
```
def core.utils.Currency.valid (
              self )

wither the given preference currency is available
```

Definition at line 103 of file utils.py.

```
103     def valid(self):
104         """ wither the given preference currency is available
105         """
106         return not self.rate==0
```

References core.utils.Currency.rate, and core.utils.Currency.valid().

Here is the call graph for this function:



7.5.3.6 valid() [2/2]

```
def core.utils.Currency.valid (
              self )

wither the given preference currency is available
```

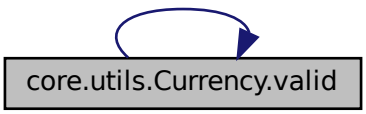
Definition at line 103 of file utils.py.

```
103     def valid(self):
104         """ wither the given preference currency is available
105         """
106         return not self.rate==0
```

References core.utils.Currency.rate.

Referenced by core.utils.Currency.valid().

Here is the caller graph for this function:



7.5.4 Field Documentation

7.5.4.1 base

`core.utils.Currency.base`

Definition at line 96 of file utils.py.

Referenced by core.utils.Currency.__init__(), core.utils.PaymentGate.authenticated(), and core.utils.PaymentGate.get_balance().

7.5.4.2 pref

`core.utils.Currency.pref`

Definition at line 97 of file utils.py.

Referenced by core.utils.Currency.__init__().

7.5.4.3 rate

`core.utils.Currency.rate`

Definition at line 98 of file utils.py.

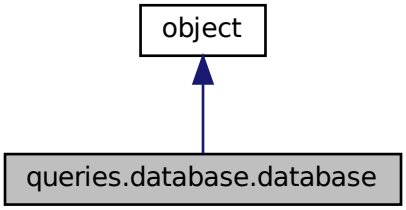
Referenced by core.utils.Currency.__init__(), core.utils.Currency.exchange(), core.utils.Currency.exchange_back(), and core.utils.Currency.valid().

The documentation for this class was generated from the following file:

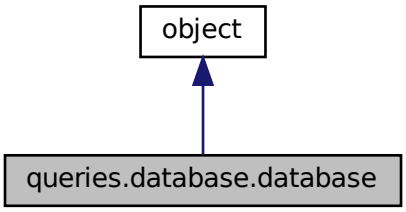
- core/utils.py

7.6 queries.database.database Class Reference

Inheritance diagram for queries.database.database:



Collaboration diagram for queries.database.database:



Public Member Functions

- def __init__ (self, dbconfigs)
- def commit (self, lock=None)
- def rollback (self, lock=None)
- def init (self)
- def repeatable_read (self)
- def committed_read (self)
- def lock_advisory (self, lock)
- def unlock_advisory (self, lock)
- def close (self)

Data Fields

- db_configs
- conn
- cur
- logger
- exists
- gets
- inserts
- updates

7.6.1 Detailed Description

Definition at line 611 of file database.py.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 __init__()

```
def queries.database.database.__init__ (
              self,
              dbconfigs )
```

Definition at line 612 of file database.py.

```
612     def __init__(self, dbconfigs):
613         #cursor
614         self.db_configs=dbconfigs
615         self.conn = psycopg2.connect(self.db_configs)
616         self.cur = self.conn.cursor()
617         #logger
618         logging.basicConfig(filename="db.log", \
619                             format='%(asctime)s %(message)s', \
620                             filemode='w')
621         self.logger=logging.getLogger()
622         self.logger.setLevel(logging.DEBUG)
623         self.exists=exists(self.conn, self.cur)
624         self.gets=gets(self.conn, self.cur, self.logger)
625         self.inserts=inserts(self.conn, self.cur, self.logger)
626         self.updates=updates(self.conn, self.cur, self.logger)
```

7.6.3 Member Function Documentation

7.6.3.1 close()

```
def queries.database.database.close (
              self )
```

Definition at line 652 of file database.py.

```
652     def close(self):
653         self.conn.close()
```

References queries.database.exists.conn, queries.database.gets.conn, queries.database.inserts.conn, queries.↩database.updates.conn, and queries.database.database.conn.

7.6.3.2 commit()

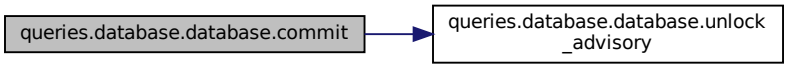
```
def queries.database.database.commit (
              self,
              lock = None )
```

Definition at line 627 of file database.py.

```
627     def commit(self, lock=None):
628         self.conn.commit()
629         self.logger.info("database committed")
630         #self.repeatable_read()
631         if not lock==None:
632             self.unlock_advisory(lock)
```

References queries.database.exists.conn, queries.database.gets.conn, queries.database.inserts.conn, queries.database.updates.conn, queries.database.database.conn, queries.database.database.logger, and queries.database.database.unlock_advisory().

Here is the call graph for this function:



7.6.3.3 committed_read()

```
def queries.database.database.committed_read (
              self )
```

Definition at line 644 of file database.py.

```
644     def committed_read(self):
645         self.cur.execute("SET TRANSACTION ISOLATION LEVEL READ COMMITTED;")
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, and queries.database.database.cur.

7.6.3.4 init()

```
def queries.database.database.init (
              self )
```

Definition at line 638 of file database.py.

```
638     def init(self):
639         self.conn=psycopg2.connect(self.db_configs) #TODO (res) should it be called?!
640         self.cur=self.conn.cursor()
641         self.logger.info("database initialized")
```

References queries.database.exists.conn, queries.database.gets.conn, queries.database.inserts.conn, queries.database.updates.conn, queries.database.database.conn, queries.database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.database.cur, queries.database.database.db_configs, and queries.database.database.logger.

7.6.3.5 lock_advisory()

```
def queries.database.database.lock_advisory (
              self,
              lock )
```

Definition at line 646 of file database.py.

```
646     def lock_advisory(self, lock):
647         stat="SELECT pg_advisory_lock({});".format(lock)
648         self.cur.execute(stat)
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.↩database.updates.cur, queries.database.database.cur, and client.client.format.

7.6.3.6 repeatable_read()

```
def queries.database.database.repeatable_read (
              self )
```

Definition at line 642 of file database.py.

```
642     def repeatable_read(self):
643         self.cur.execute("SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;")
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.↩database.updates.cur, and queries.database.database.cur.

7.6.3.7 rollback()

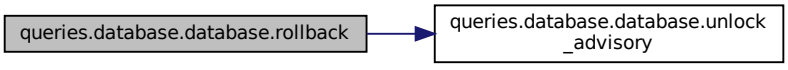
```
def queries.database.database.rollback (
              self,
              lock = None )
```

Definition at line 633 of file database.py.

```
633     def rollback(self, lock=None):
634         self.conn.rollback()
635         self.logger.info("database rollback")
636         if not lock==None:
637             self.unlock_advisory(lock)
```

References queries.database.exists.conn, queries.database.gets.conn, queries.database.inserts.conn, queries.↩database.updates.conn, queries.database.database.conn, queries.database.database.logger, and queries.↩database.database.unlock_advisory().

Here is the call graph for this function:



7.6.3.8 unlock_advisory()

```
def queries.database.database.unlock_advisory (
              self,
              lock )
```

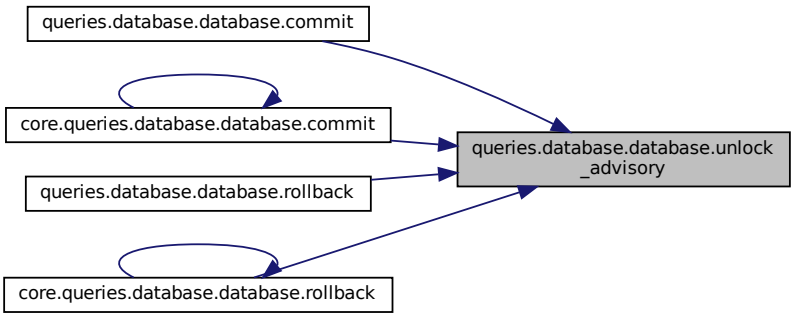
Definition at line 649 of file database.py.

```
649     def unlock_advisory(self, lock):
650         stat="SELECT pg_advisory_unlock({});".format(lock)
651         self.cur.execute(stat)
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.database.cur, and client.client.format.

Referenced by queries.database.database.commit(), core.queries.database.database.commit(), queries.database.database.rollback(), and core.queries.database.database.rollback().

Here is the caller graph for this function:



7.6.4 Field Documentation

7.6.4.1 conn

```
queries.database.database.conn
```

Definition at line 615 of file database.py.

Referenced by core.queries.database.exists.__init__(), core.queries.database.gets.__init__(), core.queries.database.inserts.__init__(), core.queries.database.updates.__init__(), core.queries.database.database.__init__(), queries.database.database.close(), core.queries.database.database.close(), queries.database.database.commit(), core.queries.database.database.commit(), core.queries.database.gets.get_all_clients(), core.queries.database.gets.get_all_contacts(), core.queries.database.gets.get_all_credentials(), core.queries.database.gets.get_credential(), core.queries.database.gets.get_sells(), core.queries.database.gets.get_transactions(), core.queries.database.gets.get_transactions_sum(), queries.database.database.init(), core.queries.database.database.init(), queries.database.database.rollback(), and core.queries.database.database.rollback().

7.6.4.2 cur

queries.database.database.cur

Definition at line 616 of file database.py.

Referenced by core.queries.database.exists.__init__(), core.queries.database.gets.__init__(), core.queries.database.inserts.__init__(), core.queries.database.updates.__init__(), core.queries.database.database.__init__(), core.queries.database.exists.account_byemail(), core.queries.database.exists.account_byname(), core.queries.database.inserts.add_bank_account(), core.queries.database.inserts.add_client(), core.queries.database.inserts.add_currency(), core.queries.database.exists.bank_account_bycid(), core.queries.database.gets.cid2credid(), core.queries.database.exists.client_exists(), queries.database.database.committed_read(), core.queries.database.database.committed_read(), core.queries.database.exists.contact_exists(), core.queries.database.exists.credential_exists(), core.queries.database.gets.credid2cid(), core.queries.database.exists.currency(), core.queries.database.updates.currency_preference(), core.queries.database.gets.get(), core.queries.database.gets.get_balance_by_cid(), core.queries.database.gets.get_balance_by_credid(), core.queries.database.gets.get_banking_id(), core.queries.database.gets.get_client_id(), core.queries.database.gets.get_client_id_byemail(), core.queries.database.gets.get_client_id_byname(), core.queries.database.gets.get_currency_id(), core.queries.database.gets.get_currency_name(), core.queries.database.gets.get_last_timestamp(), core.queries.database.gets.get_password(), core.queries.database.gets.get_preference_currency_bycid(), queries.database.database.init(), core.queries.database.database.init(), core.queries.database.inserts.insert_contact(), core.queries.database.inserts.insert_trx(), queries.database.database.lock_advisory(), core.queries.database.database.lock_advisory(), core.queries.database.inserts.register(), queries.database.database.repeatable_read(), core.queries.database.database.repeatable_read(), core.queries.database.gets.to_euro(), queries.database.database.unlock_advisory(), core.queries.database.database.unlock_advisory(), and core.queries.database.updates.update_account().

7.6.4.3 db_configs

queries.database.database.db_configs

Definition at line 614 of file database.py.

Referenced by core.queries.database.database.__init__(), queries.database.database.init(), and core.queries.database.database.init().

7.6.4.4 exists

queries.database.database.exists

Definition at line 623 of file database.py.

Referenced by core.queries.database.database.__init__().

7.6.4.5 gets

queries.database.database.gets

Definition at line 624 of file database.py.

Referenced by core.queries.database.database.__init__().

7.6.4.6 inserts

queries.database.database.inserts

Definition at line 625 of file database.py.

Referenced by core.queries.database.database.__init__().

7.6.4.7 logger

queries.database.database.logger

Definition at line 621 of file database.py.

Referenced by core.queries.database.database.__init__(), queries.database.database.commit(), core.queries.database.database.commit(), queries.database.database.init(), core.queries.database.database.init(), queries.database.database.rollback(), and core.queries.database.database.rollback().

7.6.4.8 updates

queries.database.database.updates

Definition at line 626 of file database.py.

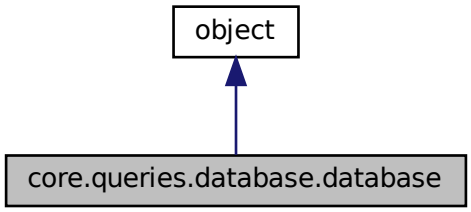
Referenced by core.queries.database.database.__init__().

The documentation for this class was generated from the following file:

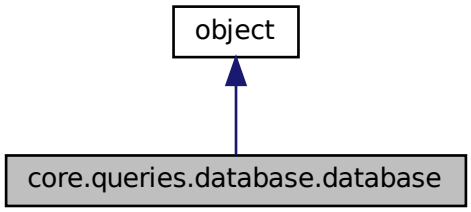
- core/build/lib/queries/database.py

7.7 core.queries.database.database Class Reference

Inheritance diagram for core.queries.database.database:



Collaboration diagram for core.queries.database.database:



Public Member Functions

- def __init__ (self, dbconfigs)
- def commit (self, lock=None)
- def rollback (self, lock=None)
- def init (self)
- def repeatable_read (self)
- def committed_read (self)
- def lock_advisory (self, lock)
- def unlock_advisory (self, lock)
- def close (self)
- def __init__ (self, dbconfigs)
- def commit (self, lock=None)
- def rollback (self, lock=None)
- def init (self)
- def repeatable_read (self)
- def committed_read (self)
- def lock_advisory (self, lock)
- def unlock_advisory (self, lock)
- def close (self)

Data Fields

- db_configs
- conn
- cur
- logger
- exists
- gets
- inserts
- updates

7.7.1 Detailed Description

Definition at line 693 of file database.py.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 __init__() [1/2]

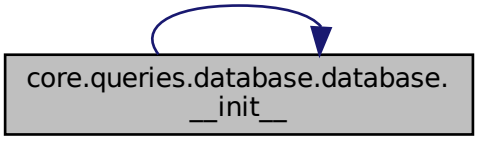
```
def core.queries.database.database.__init__ (
              self,
              dbconfigs )
```

Definition at line 694 of file database.py.

```
694     def __init__(self, dbconfigs):
695         #cursor
696         self.db_configs=dbconfigs
697         self.conn = psycopg2.connect(self.db_configs)
698         self.cur = self.conn.cursor()
699         #logger
700         logging.basicConfig(filename="db.log", \
701                             format='%(asctime)s %(message)s', \
702                             filemode='w')
703         self.logger=logging.getLogger()
704         self.logger.setLevel(logging.DEBUG)
705         self.exists=exists(self.conn, self.cur)
706         self.gets=gets(self.conn, self.cur, self.logger)
707         self.inserts=inserts(self.conn, self.cur, self.logger)
708         self.updates=updates(self.conn, self.cur, self.logger)
```

Referenced by core.queries.database.database.__init__().

Here is the caller graph for this function:



7.7.2.2 __init__() [2/2]

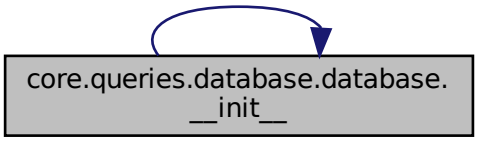
```
def core.queries.database.database.__init__ (
                self,
                dbconfigs )
```

Definition at line 694 of file database.py.

```
694     def __init__(self, dbconfigs):
695         #cursor
696         self.db_configs=dbconfigs
697         self.conn = psycopg2.connect(self.db_configs)
698         self.cur = self.conn.cursor()
699         #logger
700         logging.basicConfig(filename="db.log", \
701                             format='%(asctime)s %(message)s', \
702                             filemode='w')
703         self.logger=logging.getLogger()
704         self.logger.setLevel(logging.DEBUG)
705         self.exists=exists(self.conn, self.cur)
706         self.gets=gets(self.conn, self.cur, self.logger)
707         self.inserts=inserts(self.conn, self.cur, self.logger)
708         self.updates=updates(self.conn, self.cur, self.logger)
```

References core.queries.database.database.__init__(), queries.database.exists.conn, core.queries.database.↩ exists.conn, queries.database.gets.conn, core.queries.database.gets.conn, queries.database.inserts.conn, core.↩ queries.database.inserts.conn, queries.database.updates.conn, queries.database.database.conn, core.queries.↩ database.updates.conn, core.queries.database.database.conn, core.queries.database.exists.cur, queries.↩ database.exists.cur, queries.database.gets.cur, core.queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.updates.cur, queries.database.database.cur, core.queries.↩ database.updates.cur, core.queries.database.database.cur, queries.database.database.db_configs, core.↩ queries.database.database.db_configs, queries.database.database.exists, core.queries.database.database.↩ exists, queries.database.database.gets, core.queries.database.database.gets, queries.database.database.inserts, core.queries.database.database.inserts, queries.database.database.logger, core.queries.database.database.↩ logger, queries.database.database.updates, and core.queries.database.database.updates.

Here is the call graph for this function:



7.7.3 Member Function Documentation

7.7.3.1 close() [1/2]

```
def core.queries.database.database.close (
                self )
```

Definition at line 734 of file database.py.

```
734     def close(self):
735         self.conn.close()
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.↩ queries.database.gets.conn, queries.database.inserts.conn, core.queries.database.inserts.conn, queries.↩ database.updates.conn, queries.database.database.conn, core.queries.database.updates.conn, and core.↩ queries.database.database.conn.

Referenced by core.queries.database.database.close().

Here is the caller graph for this function:

7.7.3.2 close() [2/2]

```
def core.queries.database.database.close (
              self )
```

Definition at line 734 of file database.py.

```
734     def close(self):
735         self.conn.close()
```

References core.queries.database.database.close(), queries.database.exists.conn, core.queries.database.↩ exists.conn, queries.database.gets.conn, core.queries.database.gets.conn, queries.database.inserts.conn, core.↩ queries.database.inserts.conn, queries.database.updates.conn, queries.database.database.conn, core.queries.↩ database.updates.conn, and core.queries.database.database.conn.

Here is the call graph for this function:

7.7.3.3 commit() [1/2]

```
def core.queries.database.database.commit (
                self,
                lock = None )
```

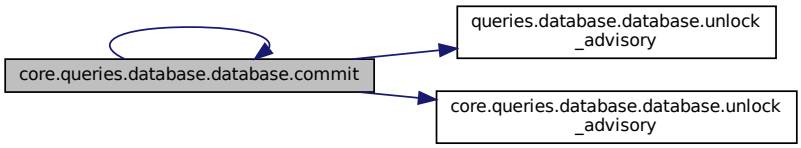
Definition at line 709 of file database.py.

```
709     def commit(self, lock=None):
710         self.conn.commit()
711         self.logger.info("database committed")
712         #self.repeatable_read()
713         if not lock==None:
714             self.unlock_advisory(lock)
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.queries.database.gets.conn, queries.database.inserts.conn, core.queries.database.inserts.conn, queries.database.updates.conn, queries.database.database.conn, core.queries.database.updates.conn, core.queries.database.database.conn, queries.database.database.logger, core.queries.database.database.logger, queries.database.database.unlock_advisory(), and core.queries.database.database.unlock_advisory().

Referenced by core.queries.database.database.commit().

Here is the call graph for this function:



Here is the caller graph for this function:

7.7.3.4 commit() [2/2]

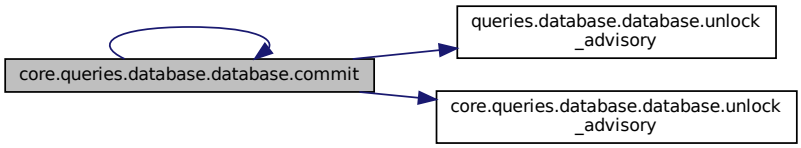
```
def core.queries.database.database.commit (
              self,
              lock = None )
```

Definition at line 709 of file database.py.

```
709     def commit(self, lock=None):
710         self.conn.commit()
711         self.logger.info("database committed")
712         #self.repeatable_read()
713         if not lock==None:
714             self.unlock_advisory(lock)
```

References core.queries.database.database.commit(), queries.database.exists.conn, core.queries.database.↩ exists.conn, queries.database.gets.conn, core.queries.database.gets.conn, queries.database.inserts.conn, core.↩ queries.database.inserts.conn, queries.database.updates.conn, queries.database.database.conn, core.queries.↩ database.updates.conn, core.queries.database.database.conn, queries.database.database.logger, core.queries.↩ database.database.logger, queries.database.database.unlock_advisory(), and core.queries.database.database.↩ unlock_advisory().

Here is the call graph for this function:



7.7.3.5 committed_read() [1/2]

```
def core.queries.database.database.committed_read (
              self )
```

Definition at line 726 of file database.py.

```
726     def committed_read(self):
727         self.cur.execute("SET TRANSACTION ISOLATION LEVEL READ COMMITTED;")
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩ updates.cur, queries.database.database.cur, core.queries.database.updates.cur, and core.queries.database.↩ database.cur.

Referenced by core.queries.database.database.committed_read().

Here is the caller graph for this function:

7.7.3.6 committed_read() [2/2]

```
def core.queries.database.database.committed_read (
              self )
```

Definition at line 726 of file database.py.

```
726     def committed_read(self):
727         self.cur.execute("SET TRANSACTION ISOLATION LEVEL READ COMMITTED;")
```

References core.queries.database.database.committed_read(), queries.database.exists.cur, core.queries.↩database.exists.cur, queries.database.gets.cur, core.queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.updates.cur, queries.database.database.cur, core.queries.↩database.updates.cur, and core.queries.database.database.cur.

Here is the call graph for this function:

7.7.3.7 init() [1/2]

```
def core.queries.database.database.init (
              self )
```

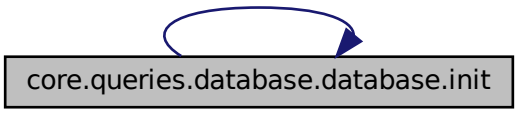
Definition at line 720 of file database.py.

```
720     def init(self):
721         self.conn=psycopg2.connect(self.db_configs) #TODO (res) should it be called?!
722         self.cur=self.conn.cursor()
```

```
723         self.logger.info("database initialized")
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.↩ queries.database.gets.conn, queries.database.inserts.conn, core.queries.database.inserts.conn, queries.↩ database.updates.conn, queries.database.database.conn, core.queries.database.updates.conn, core.queries.↩ database.database.conn, core.queries.database.exists.cur, queries.database.exists.cur, queries.database.gets.↩ cur, core.queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.↩ database.updates.cur, queries.database.database.cur, core.queries.database.updates.cur, core.queries.↩ database.database.cur, queries.database.database.db_configs, core.queries.database.database.db_configs, core.queries.database.database.init(), queries.database.database.logger, and core.queries.database.database.↩ logger.

Here is the call graph for this function:



7.7.3.8 init() [2/2]

```
def core.queries.database.database.init (
              self )
```

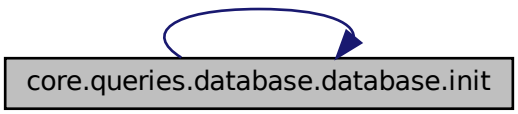
Definition at line 720 of file database.py.

```
720     def init(self):
721         self.conn=psycopg2.connect(self.db_configs) #TODO (res) should it be called?!
722         self.cur=self.conn.cursor()
723         self.logger.info("database initialized")
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.↩ queries.database.gets.conn, queries.database.inserts.conn, core.queries.database.inserts.conn, queries.↩ database.updates.conn, queries.database.database.conn, core.queries.database.updates.conn, core.queries.↩ database.database.conn, core.queries.database.exists.cur, queries.database.exists.cur, queries.database.gets.↩ cur, core.queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.↩ database.updates.cur, queries.database.database.cur, core.queries.database.updates.cur, core.queries.↩ database.database.cur, queries.database.database.db_configs, core.queries.database.database.db_configs, queries.database.database.logger, and core.queries.database.database.logger.

Referenced by core.queries.database.database.init().

Here is the caller graph for this function:



7.7.3.9 lock_advisory() [1/2]

```
def core.queries.database.database.lock_advisory (
              self,
              lock )
```

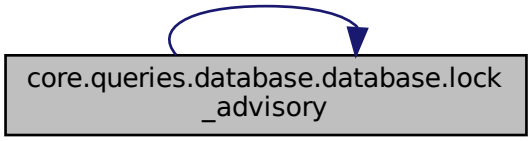
Definition at line 728 of file database.py.

```
728     def lock_advisory(self, lock):
729         stat="SELECT pg_advisory_lock({});".format(lock)
730         self.cur.execute(stat)
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩ updates.cur, queries.database.database.cur, core.queries.database.updates.cur, core.queries.database.↩ database.cur, and client.client.format.

Referenced by core.queries.database.database.lock_advisory().

Here is the caller graph for this function:



7.7.3.10 lock_advisory() [2/2]

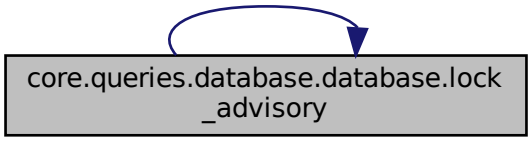
```
def core.queries.database.database.lock_advisory (
              self,
              lock )
```

Definition at line 728 of file database.py.

```
728     def lock_advisory(self, lock):
729         stat="SELECT pg_advisory_lock({});".format(lock)
730         self.cur.execute(stat)
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩ updates.cur, queries.database.database.cur, core.queries.database.updates.cur, core.queries.database.↩ database.cur, client.client.format, and core.queries.database.database.lock_advisory().

Here is the call graph for this function:



7.7.3.11 repeatable_read() [1/2]

```
def core.queries.database.database.repeatable_read (
              self )
```

Definition at line 724 of file database.py.

```
724     def repeatable_read(self):
725         self.cur.execute("SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;")
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩ updates.cur, queries.database.database.cur, core.queries.database.updates.cur, and core.queries.database.↩ database.cur.

Referenced by core.queries.database.database.repeatable_read().

Here is the caller graph for this function:

7.7.3.12 repeatable_read() [2/2]

```
def core.queries.database.database.repeatable_read (
                self )
```

Definition at line 724 of file database.py.

```
724     def repeatable_read(self):
725         self.cur.execute("SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;")
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩ updates.cur, queries.database.database.cur, core.queries.database.updates.cur, core.queries.database.↩ database.cur, and core.queries.database.database.repeatable_read().

Here is the call graph for this function:

7.7.3.13 rollback() [1/2]

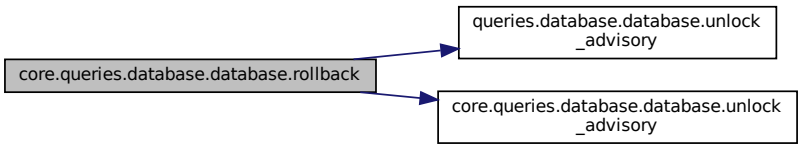
```
def core.queries.database.database.rollback (
                self,
                lock = None )
```

Definition at line 715 of file database.py.

```
715     def rollback(self, lock=None):
716         self.conn.rollback()
717         self.logger.info("database rollback")
718         if not lock==None:
719             self.unlock_advisory(lock)
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.↩ queries.database.gets.conn, queries.database.inserts.conn, core.queries.database.inserts.conn, queries.↩ database.updates.conn, queries.database.database.conn, core.queries.database.updates.conn, core.queries.↩ database.database.conn, queries.database.database.logger, core.queries.database.database.logger, core.↩ queries.database.database.rollback(), queries.database.database.unlock_advisory(), and core.queries.↩ database.database.unlock_advisory().

Here is the call graph for this function:



7.7.3.14 rollback() [2/2]

```
def core.queries.database.database.rollback (
                self,
                lock = None )
```

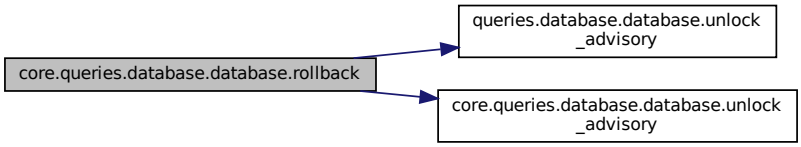
Definition at line 715 of file database.py.

```
715     def rollback(self, lock=None):
716         self.conn.rollback()
717         self.logger.info("database rollback")
718         if not lock==None:
719             self.unlock_advisory(lock)
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.↩ queries.database.gets.conn, queries.database.inserts.conn, core.queries.database.inserts.conn, queries.↩ database.updates.conn, queries.database.database.conn, core.queries.database.updates.conn, core.queries.↩ database.database.conn, queries.database.database.logger, core.queries.database.database.logger, queries.↩ database.database.unlock_advisory(), and core.queries.database.database.unlock_advisory().

Referenced by core.queries.database.database.rollback().

Here is the call graph for this function:



Here is the caller graph for this function:

7.7.3.15 unlock_advisory() [1/2]

```
def core.queries.database.database.unlock_advisory (
                self,
                lock )
```

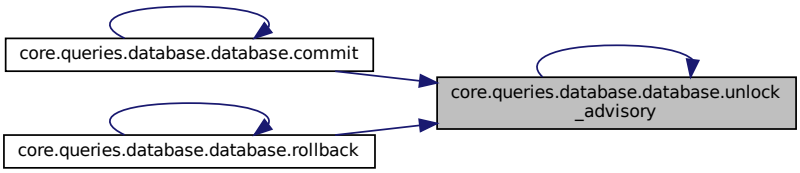
Definition at line 731 of file database.py.

```
731     def unlock_advisory(self, lock):
732         stat="SELECT pg_advisory_unlock({});".format(lock)
733         self.cur.execute(stat)
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩ updates.cur, queries.database.database.cur, core.queries.database.updates.cur, core.queries.database.↩ database.cur, and client.client.format.

Referenced by core.queries.database.database.commit(), core.queries.database.database.rollback(), and core.↩ queries.database.database.unlock_advisory().

Here is the caller graph for this function:



7.7.3.16 unlock_advisory() [2/2]

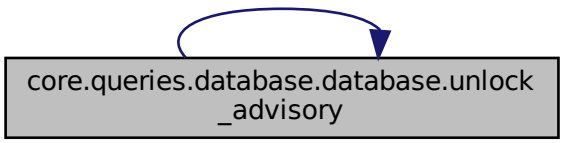
```
def core.queries.database.database.unlock_advisory (
              self,
              lock )
```

Definition at line 731 of file database.py.

```
731     def unlock_advisory(self, lock):
732         stat="SELECT pg_advisory_unlock({});".format(lock)
733         self.cur.execute(stat)
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩ updates.cur, queries.database.database.cur, core.queries.database.updates.cur, core.queries.database.↩ database.cur, client.client.format, and core.queries.database.database.unlock_advisory().

Here is the call graph for this function:



7.7.4 Field Documentation

7.7.4.1 conn

core.queries.database.database.conn

Definition at line 697 of file database.py.

Referenced by core.queries.database.database.__init__(), core.queries.database.database.close(), core.queries.database.database.commit(), core.queries.database.database.init(), and core.queries.database.database.rollback().

7.7.4.2 cur

core.queries.database.database.cur

Definition at line 698 of file database.py.

Referenced by core.queries.database.database.__init__(), core.queries.database.database.committed_read(), core.queries.database.database.init(), core.queries.database.database.lock_advisory(), core.queries.database.database.repeatable_read(), and core.queries.database.database.unlock_advisory().

7.7.4.3 db_configs

core.queries.database.database.db_configs

Definition at line 696 of file database.py.

Referenced by core.queries.database.database.__init__(), and core.queries.database.database.init().

7.7.4.4 exists

core.queries.database.database.exists

Definition at line 705 of file database.py.

Referenced by core.queries.database.database.__init__().

7.7.4.5 gets

`core.queries.database.database.gets`

Definition at line 706 of file database.py.

Referenced by core.queries.database.database.__init__().

7.7.4.6 inserts

`core.queries.database.database.inserts`

Definition at line 707 of file database.py.

Referenced by core.queries.database.database.__init__().

7.7.4.7 logger

`core.queries.database.database.logger`

Definition at line 703 of file database.py.

Referenced by core.queries.database.database.__init__(), core.queries.database.database.commit(), core.↩queries.database.database.init(), and core.queries.database.database.rollback().

7.7.4.8 updates

`core.queries.database.database.updates`

Definition at line 708 of file database.py.

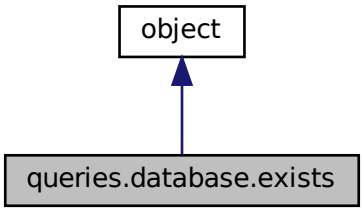
Referenced by core.queries.database.database.__init__().

The documentation for this class was generated from the following file:

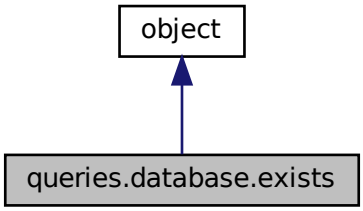
- core/queries/database.py

7.8 queries.database.exists Class Reference

Inheritance diagram for queries.database.exists:



Collaboration diagram for queries.database.exists:



Public Member Functions

- def __init__ (self, conn, cur)
- def account_byemail (self, email)
- def account_byname (self, name, passcode)
- def bank_account_bycid (self, cid)
- def client_exists (self, cid)
- def contact_exists (self, cid)
- def credential_exists (self, cid)

Data Fields

- conn
- cur

7.8.1 Detailed Description

Definition at line 11 of file database.py.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 __init__()

```
def queries.database.exists.__init__ (
              self,
              conn,
              cur )
```

Definition at line 12 of file database.py.

```
12     def __init__(self, conn, cur):
13         self.conn=conn
14         self.cur=cur
```

7.8.3 Member Function Documentation

7.8.3.1 account_byemail()

```
def queries.database.exists.account_byemail (
              self,
              email )


verify that account with corresponding email doesn't exists

@param email: client email
@return boolean for hypothesis test, that it exists
```

Definition at line 15 of file database.py.

```
15     def account_byemail(self, email):
16         """verify that account with corresponding email doesn't exists
17 
18         @param email: client email
19         @return boolean for hypothesis test, that it exists
20         """
21         stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM clients WHERE client_email={email}) FOR UPDATE SKIP
       LOCKED;")\
22             .format(email=sql.Literal(email))
23         self.cur.execute(stat)
24         fet=self.cur.fetchone()
25         print('exists.account_byemail {} fet: {}'.format(email, fet))
26         return fet[0]
```

References queries.database.exists.cur, and client.client.format.

7.8.3.2 account_byname()

```
def queries.database.exists.account_byname (
              self,
              name,
              passcode )
```

verify that account with corresponding email doesn't exists

@param name: client name
@param passcode: client passcode
@return boolean for hypothesis test, that it exists

Definition at line 27 of file database.py.

```
27     def account_byname(self, name, passcode):
28         """verify that account with corresponding email doesn't exists
29 
30         @param name: client name
31         @param passcode: client passcode
32         @return boolean for hypothesis test, that it exists
33         """
34         stat=sql.SQL("SELECT EXISTS(SELECT 1 FROM clients AS c JOIN credentials AS cred ON
       cred.id=c.client_id WHERE c.client_name={name} AND cred.passcode={passcode}) FOR UPDATE SKIP
       LOCKED;")\
35             .format(name=sql.Literal(name),\
36                     passcode=sql.Literal(passcode))
37         self.cur.execute(stat)
38         fet=self.cur.fetchone()
39         print('exists.account_byname {} fet: {}'.format(name, fet))
40         return fet[0]
```

References queries.database.exists.cur, and client.client.format.

7.8.3.3 bank_account_bycid()

```
def queries.database.exists.bank_account_bycid (
              self,
              cid )
```

verify that a banking account with the given client id is available (CALLED AT THE SERVER SIDE)

@param cid: client id
@return boolean wither the banking account for give client exists or note

Definition at line 41 of file database.py.

```
41     def bank_account_bycid(self, cid):
42         """verify that a banking account with the given client id is available (CALLED AT THE SERVER
       SIDE)
43 
44         @param cid: client id
45         @return boolean wither the banking account for give client exists or note
46         """
47         stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM banking WHERE client_id={cid}) FOR UPDATE SKIP
       LOCKED;").\
48             format(cid=sql.Literal(cid))
49         self.cur.execute(stat)
50         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, and client.client.format.

7.8.3.4 client_exists()

```
def queries.database.exists.client_exists (
              self,
              cid )
```

verify that a client with given id is available (CALLED AT THE SERVER SIDE)

@param cid: client id
@return boolean wither the client exists or note

Definition at line 51 of file database.py.

```
51     def client_exists(self, cid):
52         """verify that  a client with given id is available (CALLED AT THE SERVER SIDE)
53 
54         @param cid: client id
55         @return boolean wither the client exists or note
56         """
57         stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM clients WHERE client_id={cid}) FOR UPDATE SKIP
       LOCKED;").\
58             format(cid=sql.Literal(cid))
59         self.cur.execute(stat)
60         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, and client.client.format.

7.8.3.5 contact_exists()

```
def queries.database.exists.contact_exists (
              self,
              cid )
```

verify that a contact with given id is available (CALLED AT THE CLIENT SIDE)

@param cid: contact id
@return boolean wither the contact exists or note

Definition at line 61 of file database.py.

```
61     def contact_exists(self, cid):
62         """verify that a contact with given id is available (CALLED AT THE CLIENT SIDE)
63 
64         @param cid: contact id
65         @return boolean wither the contact exists or note
66         """
67         stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM contacts WHERE contact_id={cid}) FOR UPDATE SKIP
       LOCKED;").\
68             format(cid=sql.Literal(cid))
69         self.cur.execute(stat)
70         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, and client.client.format.

7.8.3.6 credential_exists()

```
def queries.database.exists.credential_exists (
              self,
              cid )
```

```
verify the credential for the client with given cid(CALLED FROM SERVER SIDE),
or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)

@param cid: client id, or 1 (in case of call from client side for it's own credential)
@return boolean for wither the client (with given cid) is registered or not
```

Definition at line 71 of file database.py.

```
71     def credential_exists(self, cid):
72         """ verify the credential for the client with given cid(CALLED FROM SERVER SIDE),
73         or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)
74 
75         @param cid: client id, or 1 (in case of call from client side for it's own credential)
76         @return boolean for wither the client (with given cid) is registered or not
77         """
78         stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM credentials WHERE id={cid}) FOR UPDATE SKIP
       LOCKED;").\
79             format(cid=sql.Literal(cid))
80         self.cur.execute(stat)
81         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, and client.client.format.

7.8.4 Field Documentation

7.8.4.1 conn

```
queries.database.exists.conn
```

Definition at line 13 of file database.py.

Referenced by core.queries.database.exists.__init__(), core.queries.database.gets.__init__(), core.queries.database.inserts.__init__(), core.queries.database.updates.__init__(), core.queries.database.database.__init__(), queries.database.database.close(), core.queries.database.database.close(), queries.database.database.commit(), core.queries.database.database.commit(), queries.database.gets.get_all_clients(), core.queries.database.gets.get_all_clients(), queries.database.gets.get_all_contacts(), core.queries.database.gets.get_all_contacts(), queries.database.gets.get_all_credentials(), core.queries.database.gets.get_all_credentials(), queries.database.gets.get_credential(), core.queries.database.gets.get_credential(), queries.database.gets.get_sells(), core.queries.database.gets.get_sells(), queries.database.gets.get_transactions(), core.queries.database.gets.get_transactions(), core.queries.database.gets.get_transactions_sum(), queries.database.database.init(), core.queries.database.database.init(), queries.database.database.rollback(), core.queries.database.database.rollback(), and queries.database.gets.to_dollar().

7.8.4.2 cur

`queries.database.exists.cur`

Definition at line 14 of file database.py.

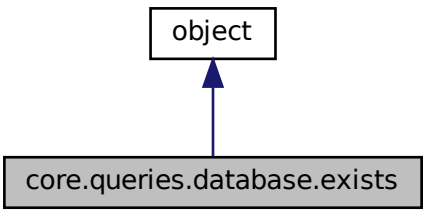
Referenced by core.queries.database.exists.__init__(), core.queries.database.gets.__init__(), core.queries.database.inserts.__init__(), core.queries.database.updates.__init__(), core.queries.database.database.__init__(), queries.database.exists.account_byemail(), core.queries.database.exists.account_byemail(), queries.database.exists.account_byname(), core.queries.database.exists.account_byname(), queries.database.inserts.add_bank_account(), core.queries.database.inserts.add_bank_account(), queries.database.inserts.add_client(), core.queries.database.inserts.add_client(), core.queries.database.inserts.add_currency(), queries.database.exists.bank_account_bycid(), core.queries.database.exists.bank_account_bycid(), queries.database.gets.cid2credid(), core.queries.database.gets.cid2credid(), queries.database.exists.client_exists(), core.queries.database.exists.client_exists(), queries.database.database.committed_read(), core.queries.database.database.committed_read(), queries.database.exists.contact_exists(), core.queries.database.exists.contact_exists(), queries.database.exists.credential_exists(), core.queries.database.exists.credential_exists(), queries.database.gets.credid2cid(), core.queries.database.gets.credid2cid(), core.queries.database.exists.currency(), core.queries.database.updates.currency_preference(), queries.database.gets.get(), core.queries.database.gets.get(), queries.database.gets.get_balance_by_cid(), core.queries.database.gets.get_balance_by_cid(), queries.database.gets.get_balance_by_credid(), core.queries.database.gets.get_balance_by_credid(), queries.database.gets.get_banking_id(), core.queries.database.gets.get_banking_id(), queries.database.gets.get_client_id(), core.queries.database.gets.get_client_id(), queries.database.gets.get_client_id_byemail(), core.queries.database.gets.get_client_id_byemail(), queries.database.gets.get_client_id_byname(), core.queries.database.gets.get_client_id_byname(), core.queries.database.gets.get_currency_id(), core.queries.database.gets.get_currency_name(), queries.database.gets.get_last_timestamp(), core.queries.database.gets.get_last_timestamp(), queries.database.gets.get_password(), core.queries.database.gets.get_password(), core.queries.database.gets.get_preference_currency_bycid(), queries.database.gets.get_transactions_sum(), queries.database.database.init(), core.queries.database.database.init(), queries.database.inserts.insert_contact(), core.queries.database.inserts.insert_contact(), queries.database.inserts.insert_trx(), core.queries.database.inserts.insert_trx(), queries.database.database.lock_advisory(), core.queries.database.database.lock_advisory(), queries.database.inserts.register(), core.queries.database.inserts.register(), queries.database.database.repeatable_read(), core.queries.database.database.repeatable_read(), core.queries.database.gets.to_euro(), queries.database.database.unlock_advisory(), core.queries.database.database.unlock_advisory(), queries.database.updates.update_account(), and core.queries.database.updates.update_account().

The documentation for this class was generated from the following file:

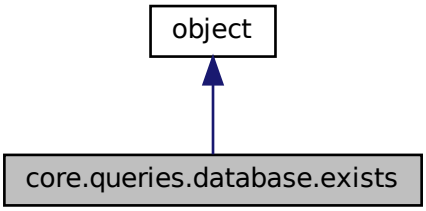
- core/build/lib/queries/database.py

7.9 core.queries.database.exists Class Reference

Inheritance diagram for core.queries.database.exists:



Collaboration diagram for core.queries.database.exists:



Public Member Functions

- def __init__ (self, conn, cur)
- def currency (self, name)
- def account_byemail (self, email)
- def account_byname (self, name, passcode)
- def bank_account_bycid (self, cid)
- def client_exists (self, cid)
- def contact_exists (self, cid)
- def credential_exists (self, cid)
- def __init__ (self, conn, cur)
- def currency (self, name)
- def account_byemail (self, email)
- def account_byname (self, name, passcode)
- def bank_account_bycid (self, cid)
- def client_exists (self, cid)
- def contact_exists (self, cid)
- def credential_exists (self, cid)

Data Fields

- conn
- cur

7.9.1 Detailed Description

Definition at line 11 of file database.py.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 __init__() [1/2]

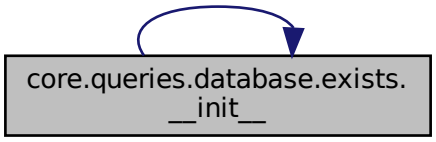
```
def core.queries.database.exists.__init__ (
                self,
                conn,
                cur )
```

Definition at line 12 of file database.py.

```
12     def __init__(self, conn, cur):
13         self.conn=conn
14         self.cur=cur
```

Referenced by core.queries.database.exists.__init__().

Here is the caller graph for this function:



7.9.2.2 __init__() [2/2]

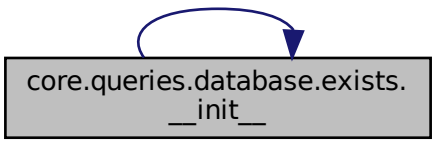
```
def core.queries.database.exists.__init__ (
                self,
                conn,
                cur )
```

Definition at line 12 of file database.py.

```
12     def __init__(self, conn, cur):
13         self.conn=conn
14         self.cur=cur
```

References core.queries.database.exists.__init__(), queries.database.exists.conn, core.queries.database.exists.↩ conn, queries.database.gets.conn, queries.database.inserts.conn, queries.database.updates.conn, queries.↩ database.database.conn, core.queries.database.exists.cur, queries.database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, and queries.database.database.cur.

Here is the call graph for this function:



7.9.3 Member Function Documentation

7.9.3.1 account_byemail() [1/2]

```
def core.queries.database.exists.account_byemail (
              self,
              email )
```

verify that account with corresponding email doesn't exists

@param email: client email
@return boolean for hypothesis test, that it exists

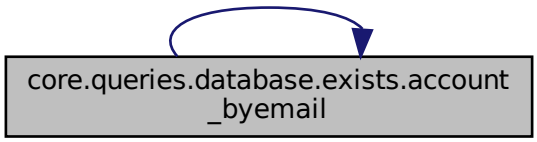
Definition at line 25 of file database.py.

```
25     def account_byemail(self, email):
26         """verify that account with corresponding email doesn't exists
27 
28         @param email: client email
29         @return boolean for hypothesis test, that it exists
30         """
31         stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM clients WHERE client_email={email}) FOR UPDATE SKIP
       LOCKED;")\
32             .format(email=sql.Literal(email))
33         self.cur.execute(stat)
34         fet=self.cur.fetchone()
35         print('exists.account_byemail {} fet: {}'.format(email, fet))
36         return fet[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.database.cur, and client.client.format.

Referenced by core.queries.database.exists.account_byemail().

Here is the caller graph for this function:



7.9.3.2 account_byemail() [2/2]

```
def core.queries.database.exists.account_byemail (
              self,
              email )
```

verify that account with corresponding email doesn't exists

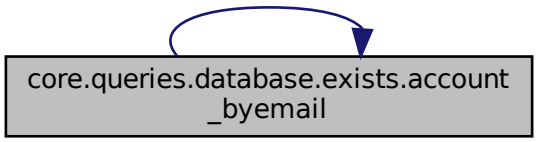
@param email: client email
@return boolean for hypothesis test, that it exists

Definition at line 25 of file database.py.

```
25     def account_byemail(self, email):
26         """verify that account with corresponding email doesn't exists
27 
28         @param email: client email
29         @return boolean for hypothesis test, that it exists
30         """
31         stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM clients WHERE client_email={email}) FOR UPDATE SKIP
       LOCKED;")\
32             .format(email=sql.Literal(email))
33         self.cur.execute(stat)
34         fet=self.cur.fetchone()
35         print('exists.account_byemail {} fet: {}'.format(email, fet))
36         return fet[0]
```

References core.queries.database.exists.account_byemail(), core.queries.database.exists.cur, queries.database.↩exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.↩database.database.cur, and client.client.format.

Here is the call graph for this function:



7.9.3.3 account_byname() [1/2]

```
def core.queries.database.exists.account_byname (
              self,
              name,
              passcode )
```

verify that account with corresponding email doesn't exists

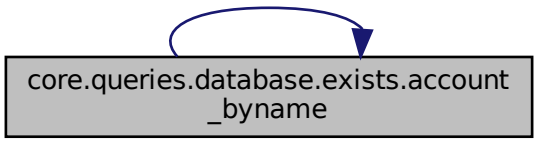
@param name: client name
@param passcode: client passcode
@return boolean for hypothesis test, that it exists

Definition at line 37 of file database.py.

```
37     def account_byname(self, name, passcode):
38         """verify that account with corresponding email doesn't exists
39 
40         @param name: client name
41         @param passcode: client passcode
42         @return boolean for hypothesis test, that it exists
43         """
44         stat=sql.SQL("SELECT EXISTS(SELECT 1 FROM clients AS c JOIN credentials AS cred ON
       cred.id=c.client_id WHERE c.client_name={name} AND cred.passcode={passcode}) FOR UPDATE SKIP
       LOCKED;")\
45             .format(name=sql.Literal(name),\
46                     passcode=sql.Literal(passcode))
47         self.cur.execute(stat)
48         fet=self.cur.fetchone()
49         print('exists.account_byname {} fet: {}'.format(name, fet))
50         return fet[0]
```

References core.queries.database.exists.account_byname(), core.queries.database.exists.cur, queries.↩database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.database.cur, and client.client.format.

Here is the call graph for this function:



7.9.3.4 account_byname() [2/2]

```
def core.queries.database.exists.account_byname (
              self,
              name,
              passcode )


verify that account with corresponding email doesn't exists

@param name: client name
@param passcode: client passcode
@return boolean for hypothesis test, that it exists
```

Definition at line 37 of file database.py.

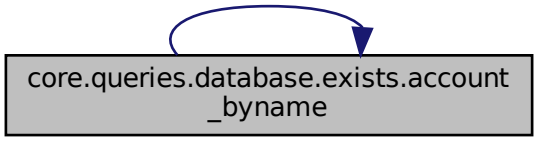
```
37     def account_byname(self, name, passcode):
38         """verify that account with corresponding email doesn't exists
39 
40         @param name: client name
41         @param passcode: client passcode
42         @return boolean for hypothesis test, that it exists
43         """
44         stat=sql.SQL("SELECT EXISTS(SELECT 1 FROM clients AS c JOIN credentials AS cred ON
       cred.id=c.client_id WHERE c.client_name={name} AND cred.passcode={passcode}) FOR UPDATE SKIP
       LOCKED;")\
45             .format(name=sql.Literal(name),\
46                     passcode=sql.Literal(passcode))
```

```
47         self.cur.execute(stat)
48         fet=self.cur.fetchone()
49         print('exists.account_byname {} fet: {}'.format(name, fet))
50         return fet[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.database.cur, and client.client.format.

Referenced by core.queries.database.exists.account_byname().

Here is the caller graph for this function:



7.9.3.5 bank_account_bycid() [1/2]

```
def core.queries.database.exists.bank_account_bycid (
              self,
              cid )


verify that a banking account with the given client id is available (CALLED AT THE SERVER SIDE)

@param cid: client id
@return boolean wither the banking account for give client exists or note
```

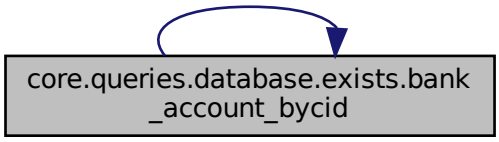
Definition at line 51 of file database.py.

```
51     def bank_account_bycid(self, cid):
52         """verify that a banking account with the given client id is available (CALLED AT THE SERVER
       SIDE)
53
54         @param cid: client id
55         @return boolean wither the banking account for give client exists or note
56         """
57         stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM banking WHERE client_id={cid}) FOR UPDATE SKIP
       LOCKED;").\
58             format(cid=sql.Literal(cid))
59         self.cur.execute(stat)
60         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.database.cur, and client.client.format.

Referenced by core.queries.database.exists.bank_account_bycid().

Here is the caller graph for this function:



7.9.3.6 bank_account_bycid() [2/2]

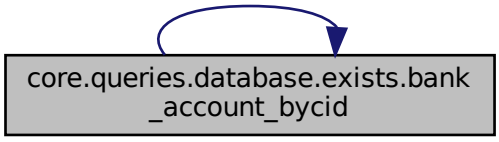
```
def core.queries.database.exists.bank_account_bycid (
              self,
              cid )

verify that a banking account with the given client id is available (CALLED AT THE SERVER SIDE)

@param cid: client id
@return boolean wither the banking account for give client exists or note
```

Definition at line 51 of file database.py.

```
51     def bank_account_bycid(self, cid):
52         """verify that a banking account with the given client id is available (CALLED AT THE SERVER
       SIDE)
53
54         @param cid: client id
55         @return boolean wither the banking account for give client exists or note
56         """
57         stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM banking WHERE client_id={cid}) FOR UPDATE SKIP
       LOCKED;").\
58             format(cid=sql.Literal(cid))
59         self.cur.execute(stat)
60         return self.cur.fetchone()[0]
```

References core.queries.database.exists.bank_account_bycid(), core.queries.database.exists.cur, queries.database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.database.cur, and client.client.format.

Here is the call graph for this function:



7.9.3.7 client_exists() [1/2]

```
def core.queries.database.exists.client_exists (
              self,
              cid )
```

verify that a client with given id is available (CALLED AT THE SERVER SIDE)

@param cid: client id
@return boolean wither the client exists or note

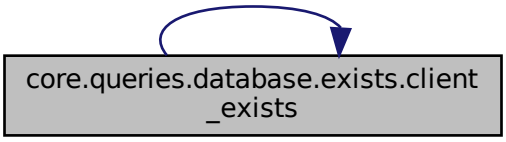
Definition at line 61 of file database.py.

```
61     def client_exists(self, cid):
62         """verify that  a client with given id is available (CALLED AT THE SERVER SIDE)
63 
64         @param cid: client id
65         @return boolean wither the client exists or note
66         """
67         stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM clients WHERE client_id={cid}) FOR UPDATE SKIP
       LOCKED;").\
68             format(cid=sql.Literal(cid))
69         self.cur.execute(stat)
70         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.database.cur, and client.client.format.

Referenced by core.queries.database.exists.client_exists().

Here is the caller graph for this function:



7.9.3.8 client_exists() [2/2]

```
def core.queries.database.exists.client_exists (
              self,
              cid )
```

verify that a client with given id is available (CALLED AT THE SERVER SIDE)

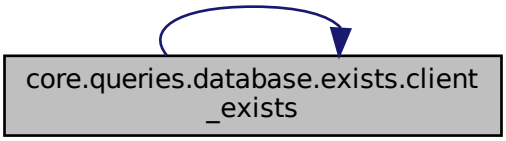
@param cid: client id
@return boolean wither the client exists or note

Definition at line 61 of file database.py.

```
61     def client_exists(self, cid):
62         """verify that  a client with given id is available (CALLED AT THE SERVER SIDE)
63 
64         @param cid: client id
65         @return boolean wither the client exists or note
66         """
67         stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM clients WHERE client_id={cid}) FOR UPDATE SKIP
       LOCKED;").\
68             format(cid=sql.Literal(cid))
69         self.cur.execute(stat)
70         return self.cur.fetchone()[0]
```

References core.queries.database.exists.client_exists(), core.queries.database.exists.cur, queries.database.↩ exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.↩ database.database.cur, and client.client.format.

Here is the call graph for this function:



7.9.3.9 contact_exists() [1/2]

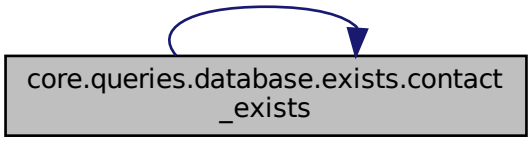
```
def core.queries.database.exists.contact_exists (
              self,
              cid )


verify that a contact with given id is available (CALLED AT THE CLIENT SIDE)

@param cid: contact id
@return boolean wither the contact exists or note
```

Definition at line 71 of file database.py.

```
71     def contact_exists(self, cid):
72         """verify that a contact with given id is available (CALLED AT THE CLIENT SIDE)
73 
74         @param cid: contact id
75         @return boolean wither the contact exists or note
76         """
77         stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM contacts WHERE contact_id={cid}) FOR UPDATE SKIP
       LOCKED;").\
78             format(cid=sql.Literal(cid))
79         self.cur.execute(stat)
80         return self.cur.fetchone()[0]
```

References core.queries.database.exists.contact_exists(), core.queries.database.exists.cur, queries.database.↩ exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.↩ database.database.cur, and client.client.format.

Here is the call graph for this function:



7.9.3.10 contact_exists() [2/2]

```
def core.queries.database.exists.contact_exists (
              self,
              cid )

verify that a contact with given id is available (CALLED AT THE CLIENT SIDE)

@param cid: contact id
@return boolean wither the contact exists or note
```

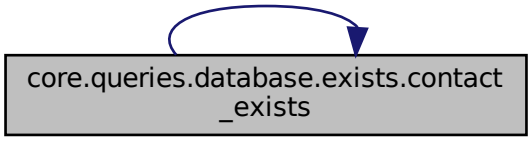
Definition at line 71 of file database.py.

```
71     def contact_exists(self, cid):
72         """verify that a contact with given id is available (CALLED AT THE CLIENT SIDE)
73
74         @param cid: contact id
75         @return boolean wither the contact exists or note
76         """
77         stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM contacts WHERE contact_id={cid}) FOR UPDATE SKIP
       LOCKED;").\
78             format(cid=sql.Literal(cid))
79         self.cur.execute(stat)
80         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.database.cur, and client.client.format.

Referenced by core.queries.database.exists.contact_exists().

Here is the caller graph for this function:



7.9.3.11 credential_exists() [1/2]

```
def core.queries.database.exists.credential_exists (
              self,
              cid )
```

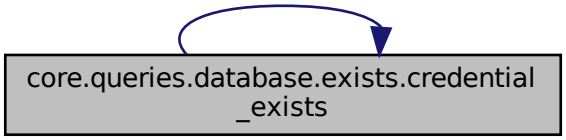
```
verify the credential for the client with given cid(CALLED FROM SERVER SIDE),
or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)

@param cid: client id, or 1 (in case of call from client side for it's own credential)
@return boolean for wither the client (with given cid) is registered or not
```

Definition at line 81 of file database.py.

```
81     def credential_exists(self, cid):
82         """ verify the credential for the client with given cid(CALLED FROM SERVER SIDE),
83         or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)
84
85         @param cid: client id, or 1 (in case of call from client side for it's own credential)
86         @return boolean for wither the client (with given cid) is registered or not
87         """
88         stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM credentials WHERE id={cid}) FOR UPDATE SKIP
       LOCKED;").\
89             format(cid=sql.Literal(cid))
90         self.cur.execute(stat)
91         return self.cur.fetchone()[0]
```

References core.queries.database.exists.credential_exists(), core.queries.database.exists.cur, queries.database.↩exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.↩database.database.cur, and client.client.format.

Here is the call graph for this function:



7.9.3.12 credential_exists() [2/2]

```
def core.queries.database.exists.credential_exists (
              self,
              cid )
```

```
verify the credential for the client with given cid(CALLED FROM SERVER SIDE),
or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)

@param cid: client id, or 1 (in case of call from client side for it's own credential)
@return boolean for wither the client (with given cid) is registered or not
```

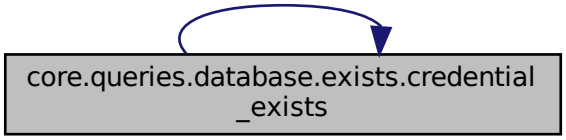
Definition at line 81 of file database.py.

```
81     def credential_exists(self, cid):
82         """ verify the credential for the client with given cid(CALLED FROM SERVER SIDE),
83         or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)
84 
85         @param cid: client id, or 1 (in case of call from client side for it's own credential)
86         @return boolean for wither the client (with given cid) is registered or not
87         """
88         stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM credentials WHERE id={cid}) FOR UPDATE SKIP
       LOCKED;").\
89             format(cid=sql.Literal(cid))
90         self.cur.execute(stat)
91         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, queries.↩database.inserts.cur, queries.database.updates.cur, queries.database.database.cur, and client.client.format.

Referenced by core.queries.database.exists.credential_exists().

Here is the caller graph for this function:



7.9.3.13 currency() [1/2]

```
def core.queries.database.exists.currency (
              self,
              name )


check wither given currency name exists

@param name: currency name
```

Definition at line 15 of file database.py.

```
15     def currency(self, name):
16         """check wither given currency name exists
17 
18         @param name: currency name
19         """
20         stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM currency WHERE currency_name={name}) LIMIT 1 FOR
       UPDATE SKIP LOCKED").format(name=sql.Literal(process_cur(name)))
21         self.cur.execute(stat)
22         ret=self.cur.fetchone()[0]
23         print('assert that currency exists with name {}, ret: {}'.format(name, ret))
24         return ret
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, queries.↩database.inserts.cur, queries.database.updates.cur, queries.database.database.cur, core.queries.database.↩exists.currency(), client.client.format, and core.utils.process_cur().

Here is the call graph for this function:

7.9.3.14 currency() [2/2]

```
def core.queries.database.exists.currency (
              self,
              name )


check wither given currency name exists

@param name: currency name
```

Definition at line 15 of file database.py.

```
15     def currency(self, name):
16         """check wither given currency name exists
17 
18         @param name: currency name
19         """
20         stat=sql.SQL("SELECT EXISTS (SELECT 1 FROM currency WHERE currency_name={name}) LIMIT 1 FOR
       UPDATE SKIP LOCKED").format(name=sql.Literal(process_cur(name)))
21         self.cur.execute(stat)
22         ret=self.cur.fetchone()[0]
23         print('assert that currency exists with name {}, ret: {}'.format(name, ret))
24         return ret
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.database.cur, client.client.format, and core.utils.process_cur().

Referenced by core.queries.database.exists.currency().

Here is the call graph for this function:

Here is the caller graph for this function:

7.9.4 Field Documentation

7.9.4.1 conn

`core.queries.database.exists.conn`

Definition at line 13 of file database.py.

Referenced by core.queries.database.exists.__init__(), core.queries.database.gets.__init__(), core.queries.database.inserts.__init__(), core.queries.database.updates.__init__(), core.queries.database.database.__init__(), core.queries.database.database.close(), core.queries.database.database.commit(), core.queries.database.gets.get_all_clients(), core.queries.database.gets.get_all_contacts(), core.queries.database.gets.get_all_credentials(), core.queries.database.gets.get_credential(), core.queries.database.gets.get_sells(), core.queries.database.gets.get_transactions(), core.queries.database.gets.get_transactions_sum(), core.queries.database.database.init(), and core.queries.database.database.rollback().

7.9.4.2 cur

`core.queries.database.exists.cur`

Definition at line 14 of file database.py.

Referenced by core.queries.database.exists.__init__(), core.queries.database.gets.__init__(), core.queries.database.inserts.__init__(), core.queries.database.updates.__init__(), core.queries.database.database.__init__(), core.queries.database.exists.account_byemail(), core.queries.database.exists.account_byname(), core.queries.database.inserts.add_bank_account(), core.queries.database.inserts.add_client(), core.queries.database.inserts.add_currency(), core.queries.database.exists.bank_account_bycid(), core.queries.database.gets.cid2credid(), core.queries.database.exists.client_exists(), core.queries.database.database.committed_read(), core.queries.database.exists.contact_exists(), core.queries.database.exists.credential_exists(), core.queries.database.gets.credid2cid(), core.queries.database.exists.currency(), core.queries.database.updates.currency_preference(), core.queries.database.gets.get(), core.queries.database.gets.get_balance_by_cid(), core.queries.database.gets.get_balance_by_credid(), core.queries.database.gets.get_banking_id(), core.queries.database.gets.get_client_id(), core.queries.database.gets.get_client_id_byemail(), core.queries.database.gets.get_client_id_byname(), core.queries.database.gets.get_currency_id(), core.queries.database.gets.get_currency_name(),

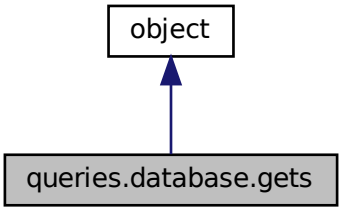
core.queries.database.gets.get_last_timestamp(), core.queries.database.gets.get_password(), core.queries.↩ database.gets.get_preference_currency_bycid(), core.queries.database.database.init(), core.queries.database.↩ inserts.insert_contact(), core.queries.database.inserts.insert_trx(), core.queries.database.database.lock_↩ advisory(), core.queries.database.inserts.register(), core.queries.database.database.repeatable_read(), core.↩ queries.database.gets.to_euro(), core.queries.database.database.unlock_advisory(), and core.queries.database.↩ updates.update_account().

The documentation for this class was generated from the following file:

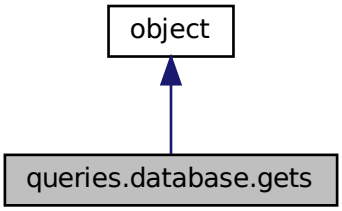
- core/queries/database.py

7.10 queries.database.gets Class Reference

Inheritance diagram for queries.database.gets:



Collaboration diagram for queries.database.gets:



Public Member Functions

- def __init__ (self, conn, cur, logger)
- def get_client_id_byemail (self, email)
- def get_client_id_byname (self, name, passcode)

- def get_client_id (self, bid)
- def get_banking_id (self, cid)
- def get_balance_by_cid (self, cid)
- def get_balance_by_credid (self, cred_id)
- def get_all_clients (self)
- def get (self, cid)
- def get_name (self, cid)
- def get_all_contacts (self)
- def get_all_credentials (self)
- def get_credential (self, cid)
- def credid2cid (self, cred_id)
- def cid2credid (self, cid)
- def get_password (self, cred_id)
- def to_dollar (self, cid)
- def get_transactions (self, st_dt, end_dt=dt.datetime.now())
- def get_transactions_sum (self, trx_with_credid, st_dt=None, end_dt=None)
- def get_sells (self, dest, st_dt, end_dt=None)
- def get_last_timestamp (self)

Data Fields

- conn
- cur
- db_log

7.10.1 Detailed Description

Definition at line 94 of file database.py.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 __init__()

```
def queries.database.gets.__init__ (
                self,
                conn,
                cur,
                logger )
```

Definition at line 95 of file database.py.

```
95     def __init__(self, conn, cur, logger):
96         self.conn=conn
97         self.cur=cur
98         self.db_log=logger
```

7.10.3 Member Function Documentation

7.10.3.1 cid2credid()

```
def queries.database.gets.cid2credid (
              self,
              cid )


get credential id

@param cid: client's id
@return credidid
```

Definition at line 254 of file database.py.

```
254     def cid2credid(self, cid):
255         """ get credential id
256 
257         @param cid: client's id
258         @return credidid
259         """
260         query=sql.SQL("SELECT cred_id FROM credentials WHERE id={cid}  FOR UPDATE SKIP LOCKED;").\
261             format(cid=sql.Literal(cid))
262         self.db_log.debug(query)
263         self.cur.execute(query)
264         fet=self.cur.fetchone()
265         print("cid2credid fet: ", fet)
266         return fet[0]
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.gets.db_log, server.server.↩ debug, and client.client.format.

7.10.3.2 credid2cid()

```
def queries.database.gets.credid2cid (
              self,
              cred_id )


get client id

@param cred_id: credential id
@return the id, or None if doesn't exist
```

Definition at line 241 of file database.py.

```
241     def credid2cid(self, cred_id):
242         """ get client id
243 
244         @param cred_id: credential id
245         @return the id, or None if doesn't exist
246         """
247         query=sql.SQL("SELECT id FROM credentials WHERE cred_id={credid}  FOR UPDATE SKIP LOCKED;").\
248             format(credid=sql.Literal(cred_id))
249         self.db_log.debug(query)
250         self.cur.execute(query)
251         fet=self.cur.fetchone()
252         print("cred2cid fet: ", fet)
253         return fet[0]
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.gets.db_log, server.server.↩ debug, and client.client.format.

7.10.3.3 get()

```
def queries.database.gets.get (
              self,
              cid )

retrieve client into with given client id (cid)

@param cid: client id
@return tuple (id, name, join date)
```

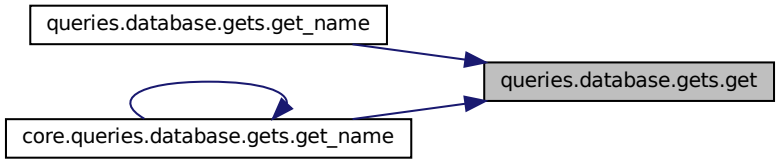
Definition at line 188 of file database.py.

```
188     def get(self, cid):
189         """retrieve client into with given client id (cid)
190
191         @param cid: client id
192         @return tuple (id, name, join date)
193         """
194         query=sql.SQL("SELECT * FROM clients WHERE client_id={cid} LIMIT 1 FOR UPDATE SKIP LOCKED;").\
195             format(cid=sql.Literal(cid))
196         self.db_log.debug(query)
197         self.cur.execute(query)
198         return self.cur.fetchone()
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.gets.db_log, server.server.↩ debug, and client.client.format.

Referenced by queries.database.gets.get_name(), and core.queries.database.gets.get_name().

Here is the caller graph for this function:



7.10.3.4 get_all_clients()

```
def queries.database.gets.get_all_clients (
              self )

retrieve all clients info
```

Definition at line 181 of file database.py.

```
181     def get_all_clients(self):
182         """retrieve all clients info
183
184         """
185         query="SELECT * FROM clients FOR UPDATE NOWAIT;"
186         self.db_log.debug(query)
187         return pd.read_sql(query, self.conn).to_json
```

References queries.database.exists.conn, queries.database.gets.conn, queries.database.gets.db_log, and server.server.debug.

7.10.3.5 get_all_contacts()

```
def queries.database.gets.get_all_contacts (
              self )
```

Definition at line 207 of file database.py.

```
207     def get_all_contacts(self):
208         query = "SELECT * FROM contacts;"
209         #query = "SELECT * FROM contacts FOR UPDATE NOWAIT;"
210         self.db_log.debug(query)
211         return pd.read_sql(query, self.conn).to_json()
```

References queries.database.exists.conn, queries.database.gets.conn, queries.database.gets.db_log, and server.server.debug.

7.10.3.6 get_all_credentials()

```
def queries.database.gets.get_all_credentials (
              self )
```

Definition at line 224 of file database.py.

```
224     def get_all_credentials(self):
225         query = "SELECT * FROM credentials;"
226         self.db_log.debug(query)
227         ret = pd.read_sql(query, self.conn)
228         return ret
229
```

References queries.database.exists.conn, queries.database.gets.conn, queries.database.gets.db_log, and server.server.debug.

7.10.3.7 get_balance_by_cid()

```
def queries.database.gets.get_balance_by_cid (
              self,
              cid )
```

```
called at the server side to retrieve the account balance d of the given client_id (cid)

@param cid: client id
@return bid: banking id
```

Definition at line 153 of file database.py.

```
153     def get_balance_by_cid(self, cid):
154         """called at the server side to retrieve the account balance d of the given client_id (cid)
155
156         @param cid: client id
157         @return bid: banking id
158         """
159         query=sql.SQL("SELECT (balance) FROM banking WHERE client_id={cid} LIMIT 1 FOR UPDATE SKIP
      LOCKED;").\
160             format(cid=sql.Literal(cid))
161         self.db_log.debug(query)
162         self.cur.execute(query)
163         return self.cur.fetchone()[0]
164         #return pd.read_sql(query, self.conn).ix[0]
165
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.gets.db_log, server.server.↩ debug, and client.client.format.

7.10.3.8 get_balance_by_credid()

```
def queries.database.gets.get_balance_by_credid (
              self,
              cred_id )
```

```
get balance of client with given credential id

@param cred_id: client credential id
```

Definition at line 166 of file database.py.

```
166     def get_balance_by_credid(self, cred_id):
167         """ get balance of client with given credential id
168 
169         @param cred_id: client credential id
170         """
171         query=sql.SQL("SELECT (b.balance) FROM banking as b JOIN  credentials AS c ON c.id=b.client_id
        WHERE c.cred_id={credid} FOR UPDATE SKIP LOCKED;").\
172             format(credid=sql.Literal(cred_id))
173         self.db_log.debug(query)
174         self.cur.execute(query)
175         fet=self.cur.fetchone()
176         print("credid fet: ", fet)
177         #TODO handle none! how to handle this in the transaction itself
178         if fet==None:
179             return 0
180         return fet[0]
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.gets.db_log, server.server.↩ debug, and client.client.format.

7.10.3.9 get_banking_id()

```
def queries.database.gets.get_banking_id (
              self,
              cid )
```

```
retrieve the corresponding banking_id of the given  client_id (cid) (called at the server side)

@param cid: client id
@return bid: banking id
```

Definition at line 140 of file database.py.

```
140     def get_banking_id(self, cid):
141         """retrieve the corresponding banking_id of the given  client_id (cid) (called at the server
        side)
142 
143         @param cid: client id
144         @return bid: banking id
145         """
146         query=sql.SQL("SELECT (id) FROM banking WHERE client_id={cid} LIMIT 1 FOR UPDATE SKIP
        LOCKED;").\
147             format(cid=sql.Literal(cid))
148         self.db_log.debug(query)
149         self.cur.execute(query)
150         return self.cur.fetchone()[0]
151         #return pd.read_sql(query, self.conn).ix[0]
152 
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.gets.db_log, server.server.↩ debug, and client.client.format.

7.10.3.10 get_client_id()

```
def queries.database.gets.get_client_id (
              self,
              bid )
```

retrieve the corresponding client_id of the given banking_id (bid) (called at the server side)

@param bid: banking id
@return cid: contact id

Definition at line 127 of file database.py.

```
127     def get_client_id(self, bid):
128         """ retrieve the corresponding client_id of the given banking_id (bid) (called at the server
        side)
129 
130         @param bid: banking id
131         @return cid: contact id
132         """
133         query=sql.SQL("SELECT (client_id) FROM banking WHERE id={bid} LIMIT 1 FOR UPDATE SKIP
        LOCKED;").\
134             format(bid=sql.Literal(bid))
135         self.db_log.debug(query)
136         self.cur.execute(query)
137         return self.cur.fetchone()[0]
138         #return pd.read_sql(query, self.conn).ix[0]
139 
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.gets.db_log, server.server.↩debug, and client.client.format.

7.10.3.11 get_client_id_byemail()

```
def queries.database.gets.get_client_id_byemail (
              self,
              email )
```

retrieve the corresponding client_id of the given banking_id (bid) (called at the server side)

@param bid: banking id
@return cid: contact id

Definition at line 99 of file database.py.

```
99     def get_client_id_byemail(self, email):
100         """ retrieve the corresponding client_id of the given banking_id (bid) (called at the server
        side)
101 
102         @param bid: banking id
103         @return cid: contact id
104         """
105         query=sql.SQL("SELECT (client_id) FROM clients WHERE client_email={email} LIMIT 1 FOR UPDATE
        SKIP LOCKED;").\
106             format(email=sql.Literal(email))
107         self.db_log.debug(query)
108         self.cur.execute(query)
109         ret=self.cur.fetchone()
110         if None:
111             return False
112         return ret[0]
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.gets.db_log, server.server.↩debug, and client.client.format.

7.10.3.12 get_client_id_byname()

```
def queries.database.gets.get_client_id_byname (
              self,
              name,
              passcode )
```

```
retrieve the corresponding client_id of the given banking_id (bid) (called at the server side)

@param bid: banking id
@return cid: contact id
```

Definition at line 114 of file database.py.

```
114     def get_client_id_byname(self, name, passcode):
115         """ retrieve the corresponding client_id of the given banking_id (bid) (called at the server
        side)
116
117         @param bid: banking id
118         @return cid: contact id
119         """
120         query=sql.SQL("SELECT (c.client_id) FROM clients AS c INNER JOIN credentials   ON
        (credentials.id=c.client_id) WHERE c.client_name={name} AND credentials.passcode={passcode} LIMIT 1
        FOR UPDATE SKIP LOCKED;").\
121             format(name=sql.Literal(name),\
122                    passcode=sql.Literal(passcode))
123         self.db_log.debug(query)
124         self.cur.execute(query)
125         return self.cur.fetchone()
126         #return pd.read_sql(query, self.conn).iloc[0]
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.gets.db_log, server.server.↩ debug, and client.client.format.

7.10.3.13 get_credential()

```
def queries.database.gets.get_credential (
              self,
              cid )
```

```
get the credential for the client with given cid(CALLED FROM SERVER SIDE),
or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)

@param cid: client id, or 1 (in case of call from client side for it's own credential)
```

Definition at line 230 of file database.py.

```
230     def get_credential(self, cid):
231         """ get the credential for the client with given cid(CALLED FROM SERVER SIDE),
232         or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)
233
234         @param cid: client id, or 1 (in case of call from client side for it's own credential)
235         """
236         query=sql.SQL("SELECT * FROM credentials WHERE id={cid} LIMIT 1 FOR UPDATE SKIP LOCKED;)").\
237             format(cid=sql.Literal(cid))
238         self.db_log.debug(query)
239         ret = pd.read_sql(query, self.conn)
240
```

References queries.database.exists.conn, queries.database.gets.conn, queries.database.gets.db_log, server.↩ server.debug, and client.client.format.

7.10.3.14 get_last_timestamp()

```
def queries.database.gets.get_last_timestamp (
              self )
```

```
retrieve the timestamp of the last transaction (CALLED FROM THE CLIENT SIDE)

@return timestamp
```

Definition at line 417 of file database.py.

```
417     def get_last_timestamp(self):
418         """ retrieve the timestamp of the last transaction (CALLED FROM THE CLIENT SIDE)
419
420         @return timestamp
421         """
422         query="SELECT currval(pg_get_serial_sequence('ledger', 'trx_id')) FOR UPDATE SKIP LOCKED;"
423         self.db_log.debug(query)
424         self.cur.execute(query)
425         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.gets.db_log, and server.↩ server.debug.

7.10.3.15 get_name()

```
def queries.database.gets.get_name (
              self,
              cid )
```

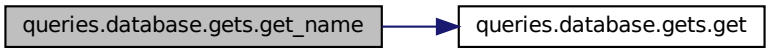
```
retrieve client name corresponding to given client id (cid)

@param cid: client id
@return client name
```

Definition at line 199 of file database.py.

```
199     def get_name(self, cid):
200         """retrieve client name corresponding to given client id (cid)
201
202         @param cid: client id
203         @return client name
204         """
205         return self.get(cid)[1]
206
```

References queries.database.gets.get().

Here is the call graph for this function:



7.10.3.16 get_password()

```
def queries.database.gets.get_password (
              self,
              cred_id )
```

get user's passcode for authentication

@param cred_id: credential id
@return list of the id, or empty list of doesn't exist

Definition at line 267 of file database.py.

```
267     def get_password(self, cred_id):
268         """ get user's passcode for authentication
269 
270         @param cred_id: credential id
271         @return list of the id, or empty list of doesn't exist
272         """
273         query=sql.SQL("SELECT (passcode) FROM credentials WHERE cred_id={credid}  FOR UPDATE SKIP
      LOCKED;").\
274             format(credid=sql.Literal(cred_id))
275         self.db_log.debug(query)
276         self.cur.execute(query)
277         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.gets.db_log, server.server.↩ debug, and client.client.format.

7.10.3.17 get_sells()

```
def queries.database.gets.get_sells (
              self,
              dest,
              st_dt,
              end_dt = None )
```

get sells transaction within the st_dt, end_dt period, while there destined to dest (CALLED AT SERVER SIDE)

@param dest: the destination credential id
@return sells transactions

Definition at line 395 of file database.py.

```
395     def get_sells(self, dest, st_dt, end_dt=None):
396         """ get sells transaction within the st_dt, end_dt period, while there destined to dest (CALLED
      AT SERVER SIDE)
397 
398         @param dest: the destination credential id
399         @return sells transactions
400         """
401 
402         stat = sql.SQL("SELECT * FROM ledger WHERE trx_dest={dest};")\
403                   .format(dest=sql.Literal(dest))
404         if st_dt==None:
405             stat=sql.SQL("SELECT * FROM ledger WHERE trx_dt>{st_dt} AND trx_dt<{end_dt} AND
      trx_dest={dest} FOR UPDATE SKIP LOCKED;").\
406             format(st_dt=sql.Literal(st_dt),\
407                    end_dt=sql.Literal(end_dt),\
408                    dest=dest)
409         self.db_log.debug(stat)
410         return pd.read_sql(stat, self.conn)
411 
412         trx=self.get_transactions(st_dt, end_dt)
413         #trx.apply(lambda x:x['trx_dest']==dest, inplace=True)
414         trx=trx.apply(lambda x:x['trx_dest']==dest)
415         return trx
```

```
416
```

References queries.database.exists.conn, queries.database.gets.conn, queries.database.gets.db_log, server.↩ server.debug, client.client.format, and queries.database.gets.get_transactions().

Here is the call graph for this function:

7.10.3.18 get_transactions()

```
def queries.database.gets.get_transactions (
              self,
              st_dt,
              end_dt = dt.datetime.now() )

get the transactions within the given period exclusively

@param st_dt: the start datetime
@param end_dt: the end datetime
@return dataframe of the transactions
```

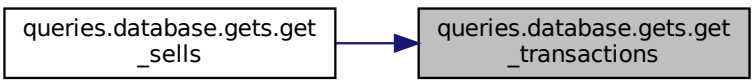
Definition at line 349 of file database.py.

```
349     def get_transactions(self, st_dt, end_dt=dt.datetime.now()):
350         """ get the transactions within the given period exclusively
351
352         @param st_dt: the start datetime
353         @param end_dt: the end datetime
354         @return dataframe of the transactions
355         """
356         stat = "SELECT * FROM ledger;"
357         if st_dt==None:
358             stat=sql.SQL("SELECT * FROM ledger WHERE trx_dt>{st_dt} AND trx_dt<{end_dt}  FOR UPDATE SKIP
      LOCKED;").\
359                 format(st_dt=sql.Literal(st_dt), end_dt=sql.Literal(end_dt))
360         self.db_log.debug(stat)
361         return pd.read_sql(stat, self.conn)
```

References queries.database.exists.conn, queries.database.gets.conn, queries.database.gets.db_log, server.↩ server.debug, and client.client.format.

Referenced by queries.database.gets.get_sells().

Here is the caller graph for this function:



7.10.3.19 get_transactions_sum()

```
def queries.database.gets.get_transactions_sum (
              self,
              trx_with_credid,
              st_dt = None,
              end_dt = None )
```

get the transactions within the given period inclusively

@param trx_with_credid: the credential id of the client of interest
@param st_dt: the start datetime
@param end_dt: the end datetime
@return dataframe of the transactions

Definition at line 363 of file database.py.

```
363     def get_transactions_sum(self, \
364                              trx_with_credid, \
365                              st_dt=None, \
366                              end_dt=None):
367         """ get the transactions within the given period inclusively
368 
369         @param trx_with_credid: the credential id of the client of interest
370         @param st_dt: the start datetime
371         @param end_dt: the end datetime
372         @return dataframe of the transactions
373         """
374         if end_dt==None:
375             end_dt=dt.datetime.now().strftime(TIMESTAMP_FORMAT)
376 
377         stat = "SELECT SUM(trx_cost) FROM ledger WHERE (trx_dest={to_credid} OR
      trx_src={from_credid});".\
378             format(to_credid=sql.Literal(trx_with_credid),\
379                    from_credid=sql.Literal(trx_with_credid))
380 
381         if not st_dt==None:
382             #note! FOR UPDATE is not allowed with aggregate functions
383             stat=sql.SQL("SELECT SUM(trx_cost) FROM ledger WHERE (trx_dt>={st_dt} AND trx_dt<{end_dt}
      AND trx_dest={to_credid}) OR (trx_dt>={st_dt} AND trx_dt<{end_dt} AND trx_src={from_credid});").\
384             format(st_dt=sql.Literal(st_dt),\
385                    end_dt=sql.Literal(end_dt),\
386                    to_credid=sql.Literal(trx_with_credid), \
387                    from_credid=sql.Literal(trx_with_credid))
388         self.db_log.debug(stat)
389         self.cur.execute(stat)
390         fet=self.cur.fetchone()[0]
391         if fet==None:
392             return 0
393         return fet
394         #return pd.read_sql(stat, self.conn)
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.gets.db_log, server.server.↩
debug, and client.client.format.

7.10.3.20 to_dollar()

```
def queries.database.gets.to_dollar (
              self,
              cid )
```

convert currency of the corresponding id to dollar ratio

for example if currency A = 2 dollars, then the conversion would be 0.5,
for another currency B = 0.5 dollar, then the conversion to dollar would be 2
such that for given cost of xA, would be 0.5x$.
@param cid is the id of the corresponding currency
@return transformation ratio to dollar

Definition at line 293 of file database.py.

```
293     def to_dollar(self, cid):
294         """ convert currency of the corresponding id to dollar ratio
295 
296         for example if currency A = 2 dollars, then the conversion would be 0.5,
297         for another currency B = 0.5 dollar, then the conversion to dollar would be 2
298         such that for given cost of xA, would be 0.5x$.
299         @param cid is the id of the corresponding currency
300         @return transformation ratio to dollar
301         """
302         query = sql.SQL("SELECT * FROM currency WHERE id=cid FOR UPDATE SKIP LOCKED;").\
303             format(cid=sql.Literal(cid))
304         self.db_log.debug(query)
305         ratio = 1.0/pd.read_sql(query, self.conn)['currency_value'].ix[0]
306         return ratio
```

References queries.database.exists.conn, queries.database.gets.conn, queries.database.gets.db_log, server.↩ server.debug, and client.client.format.

7.10.4 Field Documentation

7.10.4.1 conn

queries.database.gets.conn

Definition at line 96 of file database.py.

Referenced by core.queries.database.exists.__init__(), core.queries.database.gets.__init__(), core.queries.↩ database.inserts.__init__(), core.queries.database.updates.__init__(), core.queries.database.database.__init__(), queries.database.database.close(), core.queries.database.database.close(), queries.database.database.commit(), core.queries.database.database.commit(), queries.database.gets.get_all_clients(), core.queries.database.↩ gets.get_all_clients(), queries.database.gets.get_all_contacts(), core.queries.database.gets.get_all_contacts(), queries.database.gets.get_all_credentials(), core.queries.database.gets.get_all_credentials(), queries.database.↩ gets.get_credential(), core.queries.database.gets.get_credential(), queries.database.gets.get_sells(), core.↩ queries.database.gets.get_sells(), queries.database.gets.get_transactions(), core.queries.database.gets.get↩ _transactions(), core.queries.database.gets.get_transactions_sum(), queries.database.database.init(), core.↩ queries.database.database.init(), queries.database.database.rollback(), core.queries.database.database.rollback(), and queries.database.gets.to_dollar().

7.10.4.2 cur

queries.database.gets.cur

Definition at line 97 of file database.py.

Referenced by core.queries.database.exists.__init__(), core.queries.database.gets.__init__(), core.queries.↩ database.inserts.__init__(), core.queries.database.updates.__init__(), core.queries.database.database.__init_↩ _(), core.queries.database.exists.account_byemail(), core.queries.database.exists.account_byname(), queries.↩ database.inserts.add_bank_account(), core.queries.database.inserts.add_bank_account(), queries.database.↩ inserts.add_client(), core.queries.database.inserts.add_client(), core.queries.database.inserts.add_currency(), core.queries.database.exists.bank_account_bycid(), queries.database.gets.cid2credid(), core.queries.database.↩ gets.cid2credid(), core.queries.database.exists.client_exists(), queries.database.database.committed_read(), core.queries.database.database.committed_read(), core.queries.database.exists.contact_exists(), core.queries.↩ database.exists.credential_exists(), queries.database.gets.credid2cid(), core.queries.database.gets.credid2cid(),

core.queries.database.exists.currency(), core.queries.database.updates.currency_preference(), queries.database.gets.get(), core.queries.database.gets.get(), queries.database.gets.get_balance_by_cid(), core.queries.database.gets.get_balance_by_cid(), queries.database.gets.get_balance_by_credid(), core.queries.database.gets.get_balance_by_credid(), queries.database.gets.get_banking_id(), core.queries.database.gets.get_banking_id(), queries.database.gets.get_client_id(), core.queries.database.gets.get_client_id(), queries.database.gets.get_client_id_byemail(), core.queries.database.gets.get_client_id_byemail(), queries.database.gets.get_client_id_byname(), core.queries.database.gets.get_client_id_byname(), core.queries.database.gets.get_currency_id(), core.queries.database.gets.get_currency_name(), queries.database.gets.get_last_timestamp(), core.queries.database.gets.get_last_timestamp(), queries.database.gets.get_password(), core.queries.database.gets.get_password(), core.queries.database.gets.get_preference_currency_bycid(), queries.database.gets.get_transactions_sum(), queries.database.database.init(), core.queries.database.database.init(), queries.database.inserts.insert_contact(), core.queries.database.inserts.insert_contact(), queries.database.inserts.insert_trx(), core.queries.database.inserts.insert_trx(), queries.database.database.lock_advisory(), core.queries.database.database.lock_advisory(), queries.database.inserts.register(), core.queries.database.inserts.register(), queries.database.database.repeatable_read(), core.queries.database.database.repeatable_read(), core.queries.database.gets.to_euro(), queries.database.database.unlock_advisory(), core.queries.database.database.unlock_advisory(), queries.database.updates.update_account(), and core.queries.database.updates.update_account().

7.10.4.3 db_log

`queries.database.gets.db_log`

Definition at line 98 of file database.py.

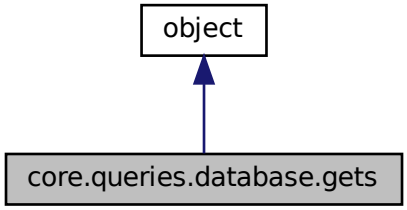
Referenced by core.queries.database.gets.__init__(), core.queries.database.inserts.__init__(), core.queries.database.updates.__init__(), queries.database.inserts.add_bank_account(), core.queries.database.inserts.add_bank_account(), queries.database.gets.cid2credid(), core.queries.database.gets.cid2credid(), queries.database.gets.credid2cid(), core.queries.database.gets.credid2cid(), queries.database.gets.get(), core.queries.database.gets.get(), queries.database.gets.get_all_clients(), core.queries.database.gets.get_all_clients(), queries.database.gets.get_all_contacts(), core.queries.database.gets.get_all_contacts(), queries.database.gets.get_all_credentials(), core.queries.database.gets.get_all_credentials(), queries.database.gets.get_balance_by_cid(), core.queries.database.gets.get_balance_by_cid(), queries.database.gets.get_balance_by_credid(), core.queries.database.gets.get_balance_by_credid(), queries.database.gets.get_banking_id(), core.queries.database.gets.get_banking_id(), queries.database.gets.get_client_id(), core.queries.database.gets.get_client_id(), queries.database.gets.get_client_id_byemail(), core.queries.database.gets.get_client_id_byemail(), queries.database.gets.get_client_id_byname(), core.queries.database.gets.get_client_id_byname(), queries.database.gets.get_credential(), core.queries.database.gets.get_credential(), core.queries.database.gets.get_currency_id(), core.queries.database.gets.get_currency_name(), queries.database.gets.get_last_timestamp(), core.queries.database.gets.get_last_timestamp(), queries.database.gets.get_password(), core.queries.database.gets.get_password(), core.queries.database.gets.get_preference_currency_bycid(), queries.database.gets.get_sells(), core.queries.database.gets.get_sells(), queries.database.gets.get_transactions(), core.queries.database.gets.get_transactions(), queries.database.gets.get_transactions_sum(), core.queries.database.gets.get_transactions_sum(), queries.database.inserts.insert_contact(), core.queries.database.inserts.insert_contact(), queries.database.inserts.insert_trx(), core.queries.database.inserts.insert_trx(), queries.database.inserts.register(), core.queries.database.inserts.register(), queries.database.gets.to_dollar(), and core.queries.database.gets.to_euro().

The documentation for this class was generated from the following file:

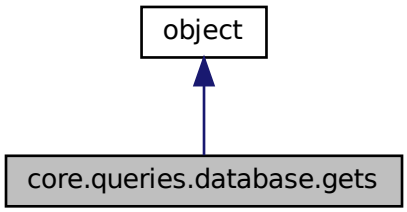
- core/build/lib/queries/database.py

7.11 core.queries.database.gets Class Reference

Inheritance diagram for core.queries.database.gets:



Collaboration diagram for core.queries.database.gets:



Public Member Functions

- def __init__ (self, conn, cur, logger)
- def get_client_id_byemail (self, email)
- def get_client_id_byname (self, name, passcode)
- def get_client_id (self, bid)
- def get_banking_id (self, cid)
- def get_preference_currency_bycid (self, cid)
- def get_currency_id (self, cur_name)
- def get_currency_name (self, id)
- def get_balance_by_cid (self, cid)
- def get_balance_by_credid (self, cred_id)
- def get_all_clients (self)
- def get (self, cid)
- def get_name (self, cid)
- def get_all_contacts (self)
- def get_all_credentials (self)
- def get_credential (self, cid)

- def credid2cid (self, cred_id)
- def cid2credid (self, cid)
- def get_password (self, cred_id)
- def to_euro (self, cid)
- def get_transactions (self, st_dt, end_dt=dt.datetime.now())
- def get_transactions_sum (self, trx_with_credid, st_dt=None, end_dt=None)
- def get_sells (self, dest, st_dt, end_dt=None)
- def get_last_timestamp (self)
- def __init__ (self, conn, cur, logger)
- def get_client_id_byemail (self, email)
- def get_client_id_byname (self, name, passcode)
- def get_client_id (self, bid)
- def get_banking_id (self, cid)
- def get_preference_currency_bycid (self, cid)
- def get_currency_id (self, cur_name)
- def get_currency_name (self, id)
- def get_balance_by_cid (self, cid)
- def get_balance_by_credid (self, cred_id)
- def get_all_clients (self)
- def get (self, cid)
- def get_name (self, cid)
- def get_all_contacts (self)
- def get_all_credentials (self)
- def get_credential (self, cid)
- def credid2cid (self, cred_id)
- def cid2credid (self, cid)
- def get_password (self, cred_id)
- def to_euro (self, cid)
- def get_transactions (self, st_dt, end_dt=dt.datetime.now())
- def get_transactions_sum (self, trx_with_credid, st_dt=None, end_dt=None)
- def get_sells (self, dest, st_dt, end_dt=None)
- def get_last_timestamp (self)

Data Fields

- conn
- cur
- db_log

7.11.1 Detailed Description

Definition at line 104 of file database.py.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 __init__() [1/2]

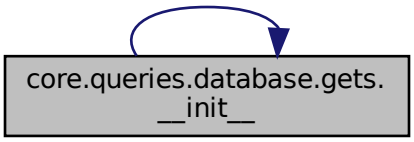
```
def core.queries.database.gets.__init__ (
              self,
              conn,
              cur,
              logger )
```

Definition at line 105 of file database.py.

```
105     def __init__(self, conn, cur, logger):
106         self.conn=conn
107         self.cur=cur
108         self.db_log=logger
```

Referenced by core.queries.database.gets.__init__().

Here is the caller graph for this function:



7.11.2.2 __init__() [2/2]

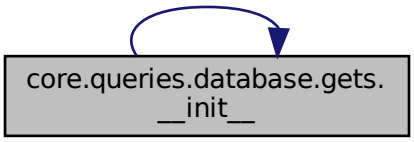
```
def core.queries.database.gets.__init__ (
              self,
              conn,
              cur,
              logger )
```

Definition at line 105 of file database.py.

```
105     def __init__(self, conn, cur, logger):
106         self.conn=conn
107         self.cur=cur
108         self.db_log=logger
```

References core.queries.database.gets.__init__(), queries.database.exists.conn, core.queries.database.exists.↩ conn, queries.database.gets.conn, core.queries.database.gets.conn, queries.database.inserts.conn, queries.↩ database.updates.conn, queries.database.database.conn, queries.database.exists.cur, core.queries.database.↩ exists.cur, queries.database.gets.cur, core.queries.database.gets.cur, queries.database.inserts.cur, queries.↩ database.updates.cur, queries.database.database.cur, queries.database.gets.db_log, core.queries.database.↩ gets.db_log, queries.database.inserts.db_log, and queries.database.updates.db_log.

Here is the call graph for this function:



7.11.3 Member Function Documentation

7.11.3.1 cid2credid() [1/2]

```
def core.queries.database.gets.cid2credid (
              self,
              cid )

get credential id

@param cid: client's id
@return credidid
```

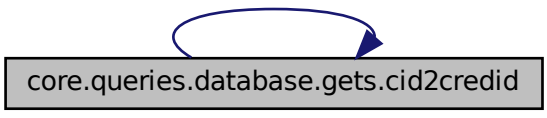
Definition at line 300 of file database.py.

```
300     def cid2credid(self, cid):
301         """ get credential id
302 
303         @param cid: client's id
304         @return credidid
305         """
306         query=sql.SQL("SELECT cred_id FROM credentials WHERE id={cid}  FOR UPDATE SKIP LOCKED;").\
307             format(cid=sql.Literal(cid))
308         self.db_log.debug(query)
309         self.cur.execute(query)
310         fet=self.cur.fetchone()
311         print("cid2credid fet: ", fet)
312         return fet[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.gets.cid2credid().

Here is the caller graph for this function:



7.11.3.2 cid2credid() [2/2]

```
def core.queries.database.gets.cid2credid (
              self,
              cid )


get credential id

@param cid: client's id
@return credidid
```

Definition at line 300 of file database.py.

```
300     def cid2credid(self, cid):
301         """ get credential id
302 
303         @param cid: client's id
304         @return credidid
305         """
306         query=sql.SQL("SELECT cred_id FROM credentials WHERE id={cid}  FOR UPDATE SKIP LOCKED;").\
307             format(cid=sql.Literal(cid))
308         self.db_log.debug(query)
309         self.cur.execute(query)
310         fet=self.cur.fetchone()
311         print("cid2credid fet: ", fet)
312         return fet[0]
```

References core.queries.database.gets.cid2credid(), queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Here is the call graph for this function:

7.11.3.3 credid2cid() [1/2]

```
def core.queries.database.gets.credid2cid (
              self,
              cred_id )


get client id

@param cred_id: credential id
@return the id, or None if doesn't exist
```

Definition at line 287 of file database.py.

```
287     def credid2cid(self, cred_id):
288         """ get client id
289 
290         @param cred_id: credential id
291         @return the id, or None if doesn't exist
292         """
293         query=sql.SQL("SELECT id FROM credentials WHERE cred_id={credid}  FOR UPDATE SKIP LOCKED;").\
294             format(credid=sql.Literal(cred_id))
295         self.db_log.debug(query)
296         self.cur.execute(query)
297         fet=self.cur.fetchone()
298         print("cred2cid fet: ", fet)
299         return fet[0]
```

References core.queries.database.gets.credid2cid(), queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Here is the call graph for this function:

7.11.3.4 credid2cid() [2/2]

```
def core.queries.database.gets.credid2cid (
                self,
                cred_id )


get client id

@param cred_id: credential id
@return the id, or None if doesn't exist
```

Definition at line 287 of file database.py.

```
287     def credid2cid(self, cred_id):
288         """ get client id
289 
290         @param cred_id: credential id
291         @return the id, or None if doesn't exist
292         """
293         query=sql.SQL("SELECT id FROM credentials WHERE cred_id={credid}  FOR UPDATE SKIP LOCKED;").\
294             format(credid=sql.Literal(cred_id))
295         self.db_log.debug(query)
296         self.cur.execute(query)
297         fet=self.cur.fetchone()
298         print("cred2cid fet: ", fet)
299         return fet[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.gets.credid2cid().

Here is the caller graph for this function:

7.11.3.5 get() [1/2]

```
def core.queries.database.gets.get (
                self,
                cid )

retrieve client into with given client id (cid)

@param cid: client id
@return tuple (id, name, join date)
```

Definition at line 234 of file database.py.

```
234     def get(self, cid):
235         """retrieve client into with given client id (cid)
236
237         @param cid: client id
238         @return tuple (id, name, join date)
239         """
240         query=sql.SQL("SELECT * FROM clients WHERE client_id={cid} LIMIT 1 FOR UPDATE SKIP LOCKED;").\
241             format(cid=sql.Literal(cid))
242         self.db_log.debug(query)
243         self.cur.execute(query)
244         return self.cur.fetchone()
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, client.client.format, and core.queries.database.gets.get().

Here is the call graph for this function:

7.11.3.6 get() [2/2]

```
def core.queries.database.gets.get (
              self,
              cid )


retrieve client into with given client id (cid)

@param cid: client id
@return tuple (id, name, join date)
```

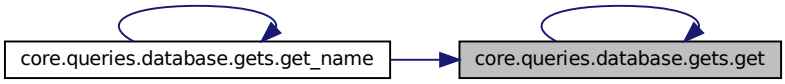
Definition at line 234 of file database.py.

```
234     def get(self, cid):
235         """retrieve client into with given client id (cid)
236 
237         @param cid: client id
238         @return tuple (id, name, join date)
239         """
240         query=sql.SQL("SELECT * FROM clients WHERE client_id={cid} LIMIT 1 FOR UPDATE SKIP LOCKED;").\
241             format(cid=sql.Literal(cid))
242         self.db_log.debug(query)
243         self.cur.execute(query)
244         return self.cur.fetchone()
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.gets.get(), and core.queries.database.gets.get_name().

Here is the caller graph for this function:



7.11.3.7 get_all_clients() [1/2]

```
def core.queries.database.gets.get_all_clients (
              self )


retrieve all clients info
```

Definition at line 227 of file database.py.

```
227     def get_all_clients(self):
228         """retrieve all clients info
229 
230         """
231         query="SELECT * FROM clients FOR UPDATE NOWAIT;"
232         self.db_log.debug(query)
233         return pd.read_sql(query, self.conn).to_json
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.↩ queries.database.gets.conn, queries.database.inserts.conn, queries.database.updates.conn, queries.database.↩ database.conn, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_↩ log, queries.database.updates.db_log, server.server.debug, and core.queries.database.gets.get_all_clients().

Here is the call graph for this function:

7.11.3.8 get_all_clients() [2/2]

```
def core.queries.database.gets.get_all_clients (
                self )
```

retrieve all clients info

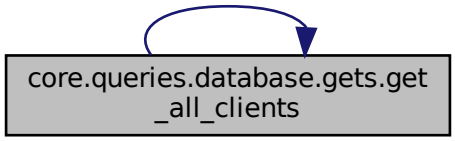
Definition at line 227 of file database.py.

```
227     def get_all_clients(self):
228         """retrieve all clients info
229 
230         """
231         query="SELECT * FROM clients FOR UPDATE NOWAIT;"
232         self.db_log.debug(query)
233         return pd.read_sql(query, self.conn).to_json
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.↩ queries.database.gets.conn, queries.database.inserts.conn, queries.database.updates.conn, queries.database.↩ database.conn, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_↩ log, queries.database.updates.db_log, and server.server.debug.

Referenced by core.queries.database.gets.get_all_clients().

Here is the caller graph for this function:



7.11.3.9 get_all_contacts() [1/2]

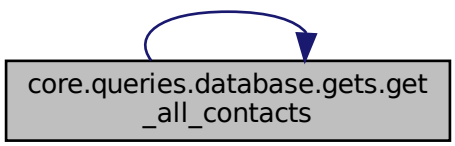
```
def core.queries.database.gets.get_all_contacts (
              self )
```

Definition at line 253 of file database.py.

```
253     def get_all_contacts(self):
254         query = "SELECT * FROM contacts;"
255         #query = "SELECT * FROM contacts FOR UPDATE NOWAIT;"
256         self.db_log.debug(query)
257         return pd.read_sql(query, self.conn).to_json()
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.↩ queries.database.gets.conn, queries.database.inserts.conn, queries.database.updates.conn, queries.database.↩ database.conn, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_↩ log, queries.database.updates.db_log, server.server.debug, and core.queries.database.gets.get_all_contacts().

Here is the call graph for this function:



7.11.3.10 get_all_contacts() [2/2]

```
def core.queries.database.gets.get_all_contacts (
              self )
```

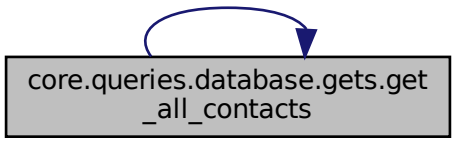
Definition at line 253 of file database.py.

```
253     def get_all_contacts(self):
254         query = "SELECT * FROM contacts;"
255         #query = "SELECT * FROM contacts FOR UPDATE NOWAIT;"
256         self.db_log.debug(query)
257         return pd.read_sql(query, self.conn).to_json()
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.↩ queries.database.gets.conn, queries.database.inserts.conn, queries.database.updates.conn, queries.database.↩ database.conn, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_↩ log, queries.database.updates.db_log, and server.server.debug.

Referenced by core.queries.database.gets.get_all_contacts().

Here is the caller graph for this function:



7.11.3.11 get_all_credentials() [1/2]

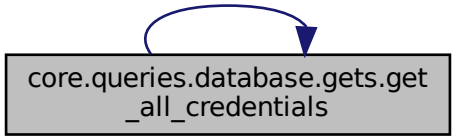
```
def core.queries.database.gets.get_all_credentials (
              self )
```

Definition at line 270 of file database.py.

```
270     def get_all_credentials(self):
271         query = "SELECT * FROM credentials;"
272         self.db_log.debug(query)
273         ret = pd.read_sql(query, self.conn)
274         return ret
275
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.↩ queries.database.gets.conn, queries.database.inserts.conn, queries.database.updates.conn, queries.database.↩ database.conn, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_↩ log, queries.database.updates.db_log, server.server.debug, and core.queries.database.gets.get_all_credentials().

Here is the call graph for this function:



7.11.3.12 get_all_credentials() [2/2]

```
def core.queries.database.gets.get_all_credentials (
              self )
```

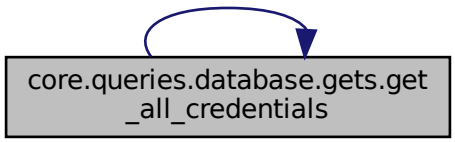
Definition at line 270 of file database.py.

```
270     def get_all_credentials(self):
271         query = "SELECT * FROM credentials;"
272         self.db_log.debug(query)
273         ret = pd.read_sql(query, self.conn)
274         return ret
275
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.↩ queries.database.gets.conn, queries.database.inserts.conn, queries.database.updates.conn, queries.database.↩ database.conn, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_↩ log, queries.database.updates.db_log, and server.server.debug.

Referenced by core.queries.database.gets.get_all_credentials().

Here is the caller graph for this function:



7.11.3.13 get_balance_by_cid() [1/2]

```
def core.queries.database.gets.get_balance_by_cid (
              self,
              cid )
```

```
called at the server side to retrieve the account balance d of the given client_id (cid)

@param cid: client id
@return dict {'balance':balance, 'base': base}
```

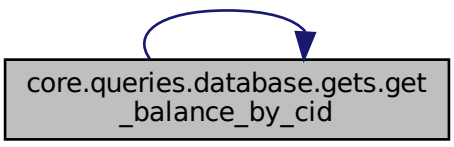
Definition at line 193 of file database.py.

```
193     def get_balance_by_cid(self, cid):
194         """called at the server side to retrieve the account balance d of the given client_id (cid)
195 
196         @param cid: client id
197         @return dict {'balance':balance, 'base': base}
198         """
199         #remove LIMIT 1 FOR UPDATE SKIP LOCKED
200         query=sql.SQL("SELECT (banking.balance, cur.currency_name) FROM banking INNER JOIN currency AS
        cur ON (cur.id=banking.currency_id) WHERE banking.client_id={cid} ;").\
201             format(cid=sql.Literal(cid))
202         self.db_log.debug(query)
203         self.cur.execute(query)
204         fet=self.cur.fetchone()
205         print('fet: ', fet)
206         print('fet[0]: ', fet[0])
207         fet=eval(fet[0])
208         balance=fet[0]
209         base=fet[1]
210         return {'balance':balance, 'base': base}
211         #return pd.read_sql(query, self.conn).ix[0]
212 
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.gets.get_balance_by_cid().

Here is the caller graph for this function:



7.11.3.14 get_balance_by_cid() [2/2]

```
def core.queries.database.gets.get_balance_by_cid (
                self,
                cid )
```

called at the server side to retrieve the account balance d of the given client_id (cid)

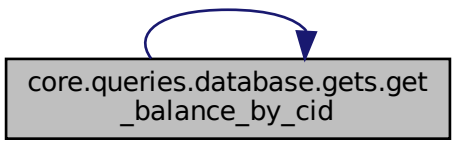
@param cid: client id
@return dict {'balance':balance, 'base': base}

Definition at line 193 of file database.py.

```
193     def get_balance_by_cid(self, cid):
194         """called at the server side to retrieve the account balance d of the given client_id (cid)
195 
196         @param cid: client id
197         @return dict {'balance':balance, 'base': base}
198         """
199         #remove LIMIT 1 FOR UPDATE SKIP LOCKED
200         query=sql.SQL("SELECT (banking.balance, cur.currency_name) FROM banking INNER JOIN currency AS
        cur ON (cur.id=banking.currency_id) WHERE banking.client_id={cid} ;").\
201             format(cid=sql.Literal(cid))
202         self.db_log.debug(query)
203         self.cur.execute(query)
204         fet=self.cur.fetchone()
205         print('fet: ', fet)
206         print('fet[0]: ', fet[0])
207         fet=eval(fet[0])
208         balance=fet[0]
209         base=fet[1]
210         return {'balance':balance, 'base': base}
211         #return pd.read_sql(query, self.conn).ix[0]
212 
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, client.client.format, and core.queries.database.gets.get_↩balance_by_cid().

Here is the call graph for this function:



7.11.3.15 get_balance_by_credid() [1/2]

```
def core.queries.database.gets.get_balance_by_credid (
              self,
              cred_id )
```

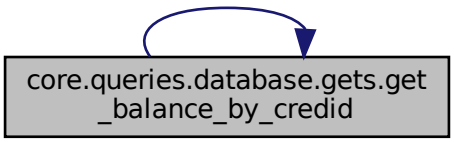
```
get balance of client with given credential id

@param cred_id: client credential id
@return dict {'balance':balance, 'base': base}
```

Definition at line 213 of file database.py.

```
213     def get_balance_by_credid(self, cred_id):
214         """ get balance of client with given credential id
215 
216         @param cred_id: client credential id
217         @return dict {'balance':balance, 'base': base}
218         """
219         query=sql.SQL("SELECT (b.balance, cur.currency_name) FROM banking as b INNER JOIN  credentials
      AS c ON (c.id=b.client_id) INNER JOIN currency AS cur ON (cur.id=b.currency_id) WHERE
      c.cred_id={credid} FOR UPDATE SKIP LOCKED;").\
220             format(credid=sql.Literal(cred_id))
221         self.db_log.debug(query)
222         self.cur.execute(query)
223         fet=eval(self.cur.fetchone()[0])
224         balance=fet[0]
225         base=fet[1]
226         return {'balance':balance, 'base': base}
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, client.client.format, and core.queries.database.gets.get_↩ balance_by_credid().

Here is the call graph for this function:



7.11.3.16 get_balance_by_credid() [2/2]

```
def core.queries.database.gets.get_balance_by_credid (
              self,
              cred_id )
```

```
get balance of client with given credential id

@param cred_id: client credential id
@return dict {'balance':balance, 'base': base}
```

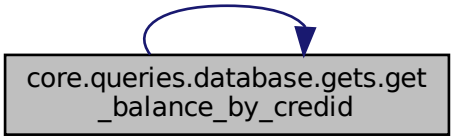
Definition at line 213 of file database.py.

```
213     def get_balance_by_credid(self, cred_id):
214         """ get balance of client with given credential id
215
216         @param cred_id: client credential id
217         @return dict {'balance':balance, 'base': base}
218         """
219         query=sql.SQL("SELECT (b.balance, cur.currency_name) FROM banking as b INNER JOIN  credentials
      AS c ON (c.id=b.client_id) INNER JOIN currency AS cur ON (cur.id=b.currency_id) WHERE
      c.cred_id={credid} FOR UPDATE SKIP LOCKED;").\
220             format(credid=sql.Literal(cred_id))
221         self.db_log.debug(query)
222         self.cur.execute(query)
223         fet=eval(self.cur.fetchone()[0])
224         balance=fet[0]
225         base=fet[1]
226         return {'balance':balance, 'base': base}
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.gets.get_balance_by_credid().

Here is the caller graph for this function:



7.11.3.17 get_banking_id() [1/2]

```
def core.queries.database.gets.get_banking_id (
                self,
                cid )

retrieve the corresponding banking_id of the given  client_id (cid) (called at the server side)

@param cid: client id
@return bid: banking id
```

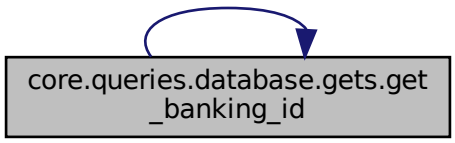
Definition at line 150 of file database.py.

```
150     def get_banking_id(self, cid):
151         """retrieve the corresponding banking_id of the given  client_id (cid) (called at the server
        side)
152 
153         @param cid: client id
154         @return bid: banking id
155         """
156         query=sql.SQL("SELECT (id) FROM banking WHERE client_id={cid} LIMIT 1 FOR UPDATE SKIP
        LOCKED;").\
157             format(cid=sql.Literal(cid))
158         self.db_log.debug(query)
159         self.cur.execute(query)
160         return self.cur.fetchone()[0]
161         #return pd.read_sql(query, self.conn).ix[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.gets.get_banking_id().

Here is the caller graph for this function:



7.11.3.18 get_banking_id() [2/2]

```
def core.queries.database.gets.get_banking_id (
                self,
                cid )
```

```
retrieve the corresponding banking_id of the given  client_id (cid) (called at the server side)

@param cid: client id
@return bid: banking id
```

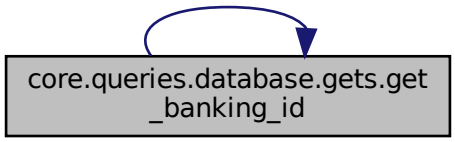
Definition at line 150 of file database.py.

```
150     def get_banking_id(self, cid):
151         """retrieve the corresponding banking_id of the given  client_id (cid) (called at the server
        side)
152 
153         @param cid: client id
154         @return bid: banking id
155         """
156         query=sql.SQL("SELECT (id) FROM banking WHERE client_id={cid} LIMIT 1 FOR UPDATE SKIP
        LOCKED;").\
157             format(cid=sql.Literal(cid))
158         self.db_log.debug(query)
159         self.cur.execute(query)
```

```
160         return self.cur.fetchone()[0]
161         #return pd.read_sql(query, self.conn).ix[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, client.client.format, and core.queries.database.gets.get_↩ banking_id().

Here is the call graph for this function:



7.11.3.19 get_client_id() [1/2]

```
def core.queries.database.gets.get_client_id (
                self,
                bid )


retrieve the corresponding client_id of the given banking_id (bid) (called at the server side)

@param bid: banking id
@return cid: contact id
```

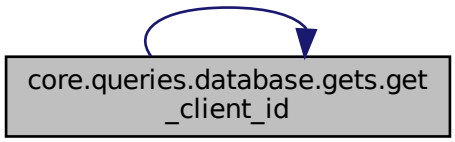
Definition at line 137 of file database.py.

```
137     def get_client_id(self, bid):
138         """ retrieve the corresponding client_id of the given banking_id (bid) (called at the server
        side)
139
140         @param bid: banking id
141         @return cid: contact id
142         """
143         query=sql.SQL("SELECT (client_id) FROM banking WHERE id={bid} LIMIT 1 FOR UPDATE SKIP
        LOCKED;").\
144             format(bid=sql.Literal(bid))
145         self.db_log.debug(query)
146         self.cur.execute(query)
147         return self.cur.fetchone()[0]
148         #return pd.read_sql(query, self.conn).ix[0]
149
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.gets.get_client_id().

Here is the caller graph for this function:



7.11.3.20 get_client_id() [2/2]

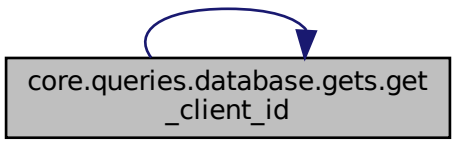
```
def core.queries.database.gets.get_client_id (
              self,
              bid )

retrieve the corresponding client_id of the given banking_id (bid) (called at the server side)

@param bid: banking id
@return cid: contact id
```

Definition at line 137 of file database.py.

```
137     def get_client_id(self, bid):
138         """ retrieve the corresponding client_id of the given banking_id (bid) (called at the server
        side)
139
140         @param bid: banking id
141         @return cid: contact id
142         """
143         query=sql.SQL("SELECT (client_id) FROM banking WHERE id={bid} LIMIT 1 FOR UPDATE SKIP
        LOCKED;").\
144             format(bid=sql.Literal(bid))
145         self.db_log.debug(query)
146         self.cur.execute(query)
147         return self.cur.fetchone()[0]
148         #return pd.read_sql(query, self.conn).ix[0]
149
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, client.client.format, and core.queries.database.gets.get_↩ client_id().

Here is the call graph for this function:



7.11.3.21 get_client_id_byemail() [1/2]

```
def core.queries.database.gets.get_client_id_byemail (
              self,
              email )
```

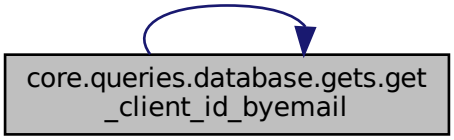
```
retrieve the corresponding client_id of the given banking_id (bid) (called at the server side)

@param bid: banking id
@return cid: contact id
```

Definition at line 109 of file database.py.

```
109     def get_client_id_byemail(self, email):
110         """ retrieve the corresponding client_id of the given banking_id (bid) (called at the server
        side)
111 
112         @param bid: banking id
113         @return cid: contact id
114         """
115         query=sql.SQL("SELECT (client_id) FROM clients WHERE client_email={email} LIMIT 1 FOR UPDATE
        SKIP LOCKED;").\
116             format(email=sql.Literal(email))
117         self.db_log.debug(query)
118         self.cur.execute(query)
119         ret=self.cur.fetchone()
120         if None:
121             return False
122         return ret[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, client.client.format, and core.queries.database.gets.get_↩ client_id_byemail().

Here is the call graph for this function:



7.11.3.22 get_client_id_byemail() [2/2]

```
def core.queries.database.gets.get_client_id_byemail (
              self,
              email )
```

```
retrieve the corresponding client_id of the given banking_id (bid) (called at the server side)

@param bid: banking id
@return cid: contact id
```

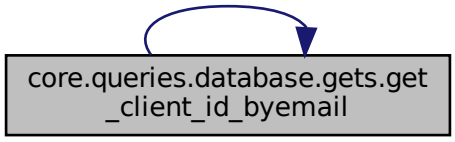
Definition at line 109 of file database.py.

```
109     def get_client_id_byemail(self, email):
110         """ retrieve the corresponding client_id of the given banking_id (bid) (called at the server
        side)
111
112         @param bid: banking id
113         @return cid: contact id
114         """
115         query=sql.SQL("SELECT (client_id) FROM clients WHERE client_email={email} LIMIT 1 FOR UPDATE
        SKIP LOCKED;").\
116             format(email=sql.Literal(email))
117         self.db_log.debug(query)
118         self.cur.execute(query)
119         ret=self.cur.fetchone()
120         if None:
121             return False
122         return ret[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.gets.get_client_id_byemail().

Here is the caller graph for this function:



7.11.3.23 get_client_id_byname() [1/2]

```
def core.queries.database.gets.get_client_id_byname (
              self,
              name,
              passcode )
```

```
retrieve the corresponding client_id of the given banking_id (bid) (called at the server side)

@param bid: banking id
@return cid: contact id
```

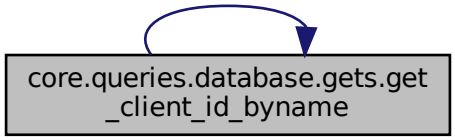
Definition at line 124 of file database.py.

```
124     def get_client_id_byname(self, name, passcode):
125         """ retrieve the corresponding client_id of the given banking_id (bid) (called at the server
        side)
126 
127         @param bid: banking id
128         @return cid: contact id
129         """
130         query=sql.SQL("SELECT (c.client_id) FROM clients AS c INNER JOIN credentials   ON
        (credentials.id=c.client_id) WHERE c.client_name={name} AND credentials.passcode={passcode} LIMIT 1
        FOR UPDATE SKIP LOCKED;").\
131             format(name=sql.Literal(name),\
132                    passcode=sql.Literal(passcode))
133         self.db_log.debug(query)
134         self.cur.execute(query)
135         return self.cur.fetchone()
136         #return pd.read_sql(query, self.conn).iloc[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.gets.get_client_id_byname().

Here is the caller graph for this function:



7.11.3.24 get_client_id_byname() [2/2]

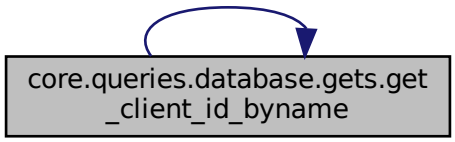
```
def core.queries.database.gets.get_client_id_byname (
                self,
                name,
                passcode )

retrieve the corresponding client_id of the given banking_id (bid) (called at the server side)

@param bid: banking id
@return cid: contact id
```

Definition at line 124 of file database.py.

```
124     def get_client_id_byname(self, name, passcode):
125         """ retrieve the corresponding client_id of the given banking_id (bid) (called at the server
        side)
126 
127         @param bid: banking id
128         @return cid: contact id
129         """
130         query=sql.SQL("SELECT (c.client_id) FROM clients AS c INNER JOIN credentials   ON
        (credentials.id=c.client_id) WHERE c.client_name={name} AND credentials.passcode={passcode} LIMIT 1
        FOR UPDATE SKIP LOCKED;").\
131             format(name=sql.Literal(name),\
132                    passcode=sql.Literal(passcode))
133         self.db_log.debug(query)
134         self.cur.execute(query)
135         return self.cur.fetchone()
136         #return pd.read_sql(query, self.conn).iloc[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, client.client.format, and core.queries.database.gets.get_↩ client_id_byname().

Here is the call graph for this function:



7.11.3.25 get_credential() [1/2]

```
def core.queries.database.gets.get_credential (
                self,
                cid )
```

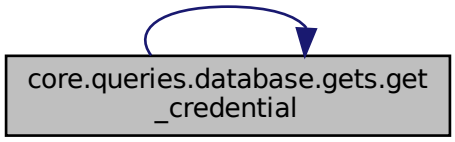
```
get the credential for the client with given cid(CALLED FROM SERVER SIDE),
or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)

@param cid: client id, or 1 (in case of call from client side for it's own credential)
```

Definition at line 276 of file database.py.

```
276     def get_credential(self, cid):
277         """ get the credential for the client with given cid(CALLED FROM SERVER SIDE),
278         or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)
279 
280         @param cid: client id, or 1 (in case of call from client side for it's own credential)
281         """
282         query=sql.SQL("SELECT * FROM credentials WHERE id={cid} LIMIT 1 FOR UPDATE SKIP LOCKED;)").\
283             format(cid=sql.Literal(cid))
284         self.db_log.debug(query)
285         ret = pd.read_sql(query, self.conn)
286 
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.↩
queries.database.gets.conn, queries.database.inserts.conn, queries.database.updates.conn, queries.database.↩
database.conn, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_↩
log, queries.database.updates.db_log, server.server.debug, client.client.format, and core.queries.database.gets.↩
get_credential().

Here is the call graph for this function:



7.11.3.26 get_credential() [2/2]

```
def core.queries.database.gets.get_credential (
                self,
                cid )
```

```
get the credential for the client with given cid(CALLED FROM SERVER SIDE),
or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)

@param cid: client id, or 1 (in case of call from client side for it's own credential)
```

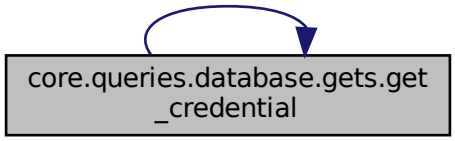
Definition at line 276 of file database.py.

```
276     def get_credential(self, cid):
277         """ get the credential for the client with given cid(CALLED FROM SERVER SIDE),
278         or get the single row for client with cid=1 (CALLED FROM CLIENT SIDE)
279
280         @param cid: client id, or 1 (in case of call from client side for it's own credential)
281         """
282         query=sql.SQL("SELECT * FROM credentials WHERE id={cid} LIMIT 1 FOR UPDATE SKIP LOCKED;)").\
283             format(cid=sql.Literal(cid))
284         self.db_log.debug(query)
285         ret = pd.read_sql(query, self.conn)
286
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.↩
queries.database.gets.conn, queries.database.inserts.conn, queries.database.updates.conn, queries.database.↩
database.conn, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_↩
log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.gets.get_credential().

Here is the caller graph for this function:



7.11.3.27 get_currency_id() [1/2]

```
def core.queries.database.gets.get_currency_id (
              self,
              cur_name )
```

```
get currency id associated with currency_name

@param cur_name: currency_name
@return id: currency id
```

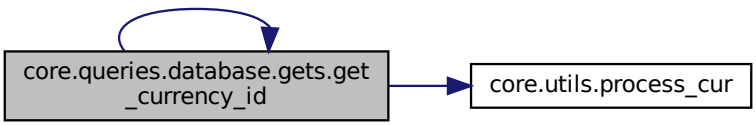
Definition at line 173 of file database.py.

```
173     def get_currency_id(self, cur_name):
174         """ get currency id associated with currency_name
175 
176         @param cur_name: currency_name
177         @return id: currency id
178         """
179         query=sql.SQL("SELECT id FROM currency WHERE
       currency_name={cur_name}").format(cur_name=sql.Literal(process_cur(cur_name)))
180         self.db_log.debug(query)
181         self.cur.execute(query)
182         return self.cur.fetchone()
```

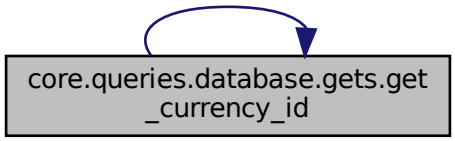
References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, client.client.format, and core.utils.process_cur().

Referenced by core.queries.database.gets.get_currency_id().

Here is the call graph for this function:



Here is the caller graph for this function:



7.11.3.28 get_currency_id() [2/2]

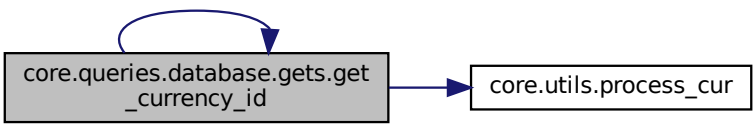
```
def core.queries.database.gets.get_currency_id (
              self,
              cur_name )

get currency id associated with currency_name

@param cur_name: currency_name
@return id: currency id
```

Definition at line 173 of file database.py.

```
173     def get_currency_id(self, cur_name):
174         """ get currency id associated with currency_name
175
176         @param cur_name: currency_name
177         @return id: currency id
178         """
179         query=sql.SQL("SELECT id FROM currency WHERE
       currency_name={cur_name}").format(cur_name=sql.Literal(process_cur(cur_name)))
180         self.db_log.debug(query)
181         self.cur.execute(query)
182         return self.cur.fetchone()
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, client.client.format, core.queries.database.gets.get_↩ currency_id(), and core.utils.process_cur().

Here is the call graph for this function:



7.11.3.29 get_currency_name() [1/2]

```
def core.queries.database.gets.get_currency_name (
              self,
              id )
```

```
get currency name associated with currency id

@param id: currency id
@return cur_name: currency_name
```

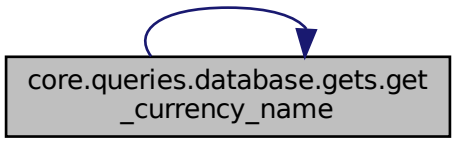
Definition at line 183 of file database.py.

```
183     def get_currency_name(self, id):
184         """ get currency name associated with currency id
185 
186         @param id: currency id
187         @return cur_name: currency_name
188         """
189         query=sql.SQL("SELECT currency_name FROM currency WHERE id={id}").format(id=sql.Literal(id))
190         self.db_log.debug(query)
191         self.cur.execute(query)
192         return self.cur.fetchone()
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.gets.get_currency_name().

Here is the caller graph for this function:



7.11.3.30 get_currency_name() [2/2]

```
def core.queries.database.gets.get_currency_name (
              self,
              id )
```

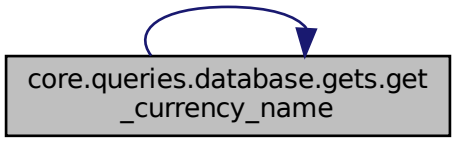
```
get currency name associated with currency id

@param id: currency id
@return cur_name: currency_name
```

Definition at line 183 of file database.py.

```
183     def get_currency_name(self, id):
184         """ get currency name associated with currency id
185 
186         @param id: currency id
187         @return cur_name: currency_name
188         """
189         query=sql.SQL("SELECT currency_name FROM currency WHERE id={id}").format(id=sql.Literal(id))
190         self.db_log.debug(query)
191         self.cur.execute(query)
192         return self.cur.fetchone()
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, client.client.format, and core.queries.database.gets.get_↩ currency_name().

Here is the call graph for this function:



7.11.3.31 get_last_timestamp() [1/2]

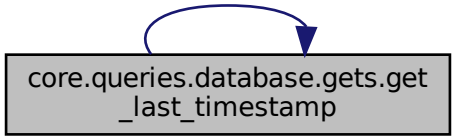
```
def core.queries.database.gets.get_last_timestamp (
                self )


retrieve the timestamp of the last transaction (CALLED FROM THE CLIENT SIDE)

@return timestamp
```

Definition at line 476 of file database.py.

```
476     def get_last_timestamp(self):
477         """ retrieve the timestamp of the last transaction (CALLED FROM THE CLIENT SIDE)
478 
479         @return timestamp
480         """
481         query="SELECT currval(pg_get_serial_sequence('ledger', 'trx_id')) FOR UPDATE SKIP LOCKED;"
482         self.db_log.debug(query)
483         self.cur.execute(query)
484         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and core.queries.database.gets.get_last_timestamp().

Here is the call graph for this function:



7.11.3.32 get_last_timestamp() [2/2]

```
def core.queries.database.gets.get_last_timestamp (
              self )

retrieve the timestamp of the last transaction (CALLED FROM THE CLIENT SIDE)

@return timestamp
```

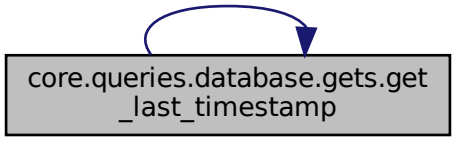
Definition at line 476 of file database.py.

```
476     def get_last_timestamp(self):
477         """ retrieve the timestamp of the last transaction (CALLED FROM THE CLIENT SIDE)
478
479         @return timestamp
480         """
481         query="SELECT currval(pg_get_serial_sequence('ledger', 'trx_id')) FOR UPDATE SKIP LOCKED;"
482         self.db_log.debug(query)
483         self.cur.execute(query)
484         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, and server.server.debug.

Referenced by core.queries.database.gets.get_last_timestamp().

Here is the caller graph for this function:



7.11.3.33 get_name() [1/2]

```
def core.queries.database.gets.get_name (
                self,
                cid )
```

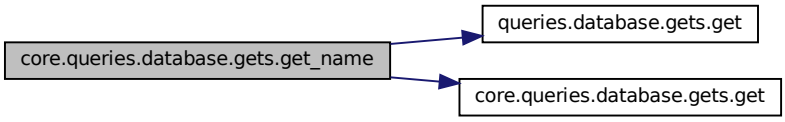
```
retrieve client name corresponding to given client id (cid)

@param cid: client id
@return client name
```

Definition at line 245 of file database.py.

```
245     def get_name(self, cid):
246         """retrieve client name corresponding to given client id (cid)
247 
248         @param cid: client id
249         @return client name
250         """
251         return self.get(cid)[1]
252 
```

References queries.database.gets.get(), core.queries.database.gets.get(), and core.queries.database.gets.get_↩ name().

Here is the call graph for this function:



7.11.3.34 get_name() [2/2]

```
def core.queries.database.gets.get_name (
                self,
                cid )
```

```
retrieve client name corresponding to given client id (cid)

@param cid: client id
@return client name
```

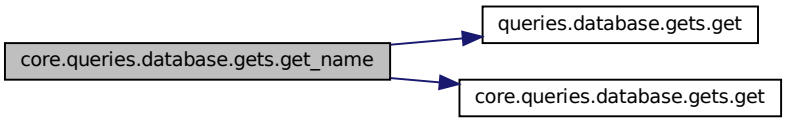
Definition at line 245 of file database.py.

```
245     def get_name(self, cid):
246         """retrieve client name corresponding to given client id (cid)
247 
248         @param cid: client id
249         @return client name
250         """
251         return self.get(cid)[1]
252 
```

References queries.database.gets.get(), and core.queries.database.gets.get().

Referenced by core.queries.database.gets.get_name().

Here is the call graph for this function:



Here is the caller graph for this function:

7.11.3.35 get_password() [1/2]

```
def core.queries.database.gets.get_password (
              self,
              cred_id )


get user's passcode for authentication

@param cred_id: credential id
@return list of the id, or empty list of doesn't exist
```

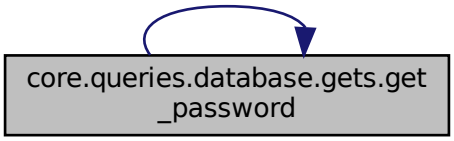
Definition at line 313 of file database.py.

```
313     def get_password(self, cred_id):
314         """ get user's passcode for authentication
315 
316         @param cred_id: credential id
317         @return list of the id, or empty list of doesn't exist
318         """
319         query=sql.SQL("SELECT (passcode) FROM credentials WHERE cred_id={credid}  FOR UPDATE SKIP
      LOCKED;").\
320             format(credid=sql.Literal(cred_id))
321         self.db_log.debug(query)
322         self.cur.execute(query)
323         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.gets.get_password().

Here is the caller graph for this function:



7.11.3.36 get_password() [2/2]

```
def core.queries.database.gets.get_password (
              self,
              cred_id )
```

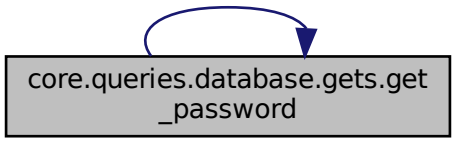
```
get user's passcode for authentication

@param cred_id: credential id
@return list of the id, or empty list of doesn't exist
```

Definition at line 313 of file database.py.

```
313     def get_password(self, cred_id):
314         """ get user's passcode for authentication
315 
316         @param cred_id: credential id
317         @return list of the id, or empty list of doesn't exist
318         """
319         query=sql.SQL("SELECT (passcode) FROM credentials WHERE cred_id={credid}  FOR UPDATE SKIP
      LOCKED;").\
320             format(credid=sql.Literal(cred_id))
321         self.db_log.debug(query)
322         self.cur.execute(query)
323         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, client.client.format, and core.queries.database.gets.get_↩ password().

Here is the call graph for this function:



7.11.3.37 get_preference_currency_bycid() [1/2]

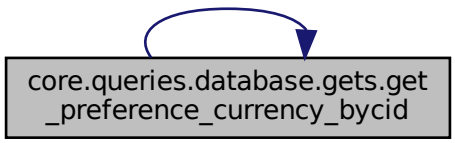
```
def core.queries.database.gets.get_preference_currency_bycid (
              self,
              cid )

get the preference currency for client with client id (cid)

@param cid: client id
@return string: of the preference currency name
```

Definition at line 162 of file database.py.

```
162     def get_preference_currency_bycid(self, cid):
163         """get the preference currency for client with client id (cid)
164 
165         @param cid: client id
166         @return string: of the preference currency name
167         """
168         query=sql.SQL("SELECT (currency.currency_name) FROM currency  JOIN clients  ON
       (clients.currency_id=currency.id) WHERE clients.client_id={cid} LIMIT 1 FOR UPDATE SKIP LOCKED;").\
169             format(cid=sql.Literal(cid))
170         self.db_log.debug(query)
171         self.cur.execute(query)
172         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, client.client.format, and core.queries.database.gets.get_↩ preference_currency_bycid().

Here is the call graph for this function:



7.11.3.38 get_preference_currency_bycid() [2/2]

```
def core.queries.database.gets.get_preference_currency_bycid (
              self,
              cid )
```

```
get the preference currency for client with client id (cid)

@param cid: client id
@return string: of the preference currency name
```

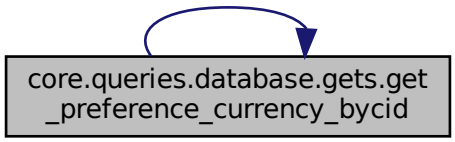
Definition at line 162 of file database.py.

```
162     def get_preference_currency_bycid(self, cid):
163         """get the preference currency for client with client id (cid)
164 
165         @param cid: client id
166         @return string: of the preference currency name
167         """
168         query=sql.SQL("SELECT (currency.currency_name) FROM currency  JOIN clients  ON
       (clients.currency_id=currency.id) WHERE clients.client_id={cid} LIMIT 1 FOR UPDATE SKIP LOCKED;").\
169             format(cid=sql.Literal(cid))
170         self.db_log.debug(query)
171         self.cur.execute(query)
172         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.gets.get_preference_currency_bycid().

Here is the caller graph for this function:



7.11.3.39 get_sells() [1/2]

```
def core.queries.database.gets.get_sells (
              self,
              dest,
              st_dt,
              end_dt = None )
```

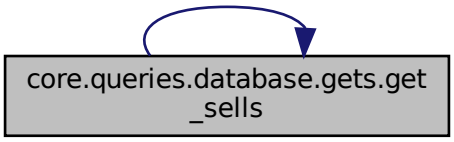
```
get sells transaction within the st_dt, end_dt period, while there destined to dest (CALLED AT SERVER SIDE)

@param dest: the destination credential id
@return sells transactions
```

Definition at line 452 of file database.py.

```
452     def get_sells(self, dest, st_dt, end_dt=None):
453         """ get sells transaction within the st_dt, end_dt period, while there destined to dest (CALLED
        AT SERVER SIDE)
454 
455         @param dest: the destination credential id
456         @return sells transactions
457         """
458 
459         stat = sql.SQL("SELECT * FROM ledger WHERE trx_dest={dest};")\
460                   .format(dest=sql.Literal(dest))
461         if st_dt==None:
462             stat=sql.SQL("SELECT * FROM ledger WHERE trx_dt>{st_dt} AND trx_dt<{end_dt} AND
        trx_dest={dest} FOR UPDATE SKIP LOCKED;").\
463             format(st_dt=sql.Literal(st_dt),\
464                    end_dt=sql.Literal(end_dt),\
465                    dest=dest)
466         self.db_log.debug(stat)
467         return pd.read_sql(stat, self.conn).to_json()
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.↩ queries.database.gets.conn, queries.database.inserts.conn, queries.database.updates.conn, queries.database.↩ database.conn, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_↩ log, queries.database.updates.db_log, server.server.debug, client.client.format, and core.queries.database.gets.↩ get_sells().

Here is the call graph for this function:



7.11.3.40 get_sells() [2/2]

```
def core.queries.database.gets.get_sells (
              self,
              dest,
              st_dt,
              end_dt = None )

get sells transaction within the st_dt, end_dt period, while there destined to dest (CALLED AT SERVER SIDE)

@param dest: the destination credential id
@return sells transactions
```

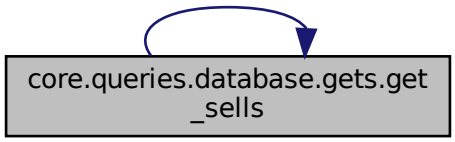
Definition at line 452 of file database.py.

```
452     def get_sells(self, dest, st_dt, end_dt=None):
453         """ get sells transaction within the st_dt, end_dt period, while there destined to dest (CALLED
        AT SERVER SIDE)
454 
455         @param dest: the destination credential id
456         @return sells transactions
457         """
458 
459         stat = sql.SQL("SELECT * FROM ledger WHERE trx_dest={dest};")\
460                   .format(dest=sql.Literal(dest))
461         if st_dt==None:
462             stat=sql.SQL("SELECT * FROM ledger WHERE trx_dt>{st_dt} AND trx_dt<{end_dt} AND
        trx_dest={dest} FOR UPDATE SKIP LOCKED;").\
463             format(st_dt=sql.Literal(st_dt),\
464                    end_dt=sql.Literal(end_dt),\
465                    dest=dest)
466         self.db_log.debug(stat)
467         return pd.read_sql(stat, self.conn).to_json()
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.↩ queries.database.gets.conn, queries.database.inserts.conn, queries.database.updates.conn, queries.database.↩ database.conn, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_↩ log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.gets.get_sells().

Here is the caller graph for this function:



7.11.3.41 get_transactions() [1/2]

```
def core.queries.database.gets.get_transactions (
                self,
                st_dt,
                end_dt = dt.datetime.now() )


get the transactions within the given period exclusively

@param st_dt: the start datetime
@param end_dt: the end datetime
@return dataframe of the transactions
```

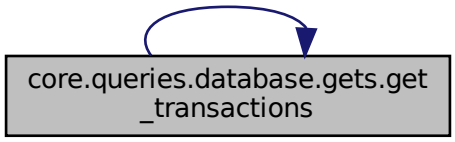
Definition at line 397 of file database.py.

```
397     def get_transactions(self, st_dt, end_dt=dt.datetime.now()):
398         """ get the transactions within the given period exclusively
399 
400         @param st_dt: the start datetime
401         @param end_dt: the end datetime
402         @return dataframe of the transactions
```

```
403         """
404         stat = "SELECT * FROM ledger;"
405         if st_dt==None:
406             stat=sql.SQL("SELECT * FROM ledger WHERE trx_dt>{st_dt} AND trx_dt<{end_dt}  FOR UPDATE SKIP
        LOCKED;").\
407             format(st_dt=sql.Literal(st_dt), end_dt=sql.Literal(end_dt))
408         self.db_log.debug(stat)
409         return pd.read_sql(stat, self.conn)
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.↩ queries.database.gets.conn, queries.database.inserts.conn, queries.database.updates.conn, queries.database.↩ database.conn, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_↩ log, queries.database.updates.db_log, server.server.debug, client.client.format, and core.queries.database.gets.↩ get_transactions().

Here is the call graph for this function:



7.11.3.42 get_transactions() [2/2]

```
def core.queries.database.gets.get_transactions (
                self,
                st_dt,
                end_dt = dt.datetime.now() )


get the transactions within the given period exclusively

@param st_dt: the start datetime
@param end_dt: the end datetime
@return dataframe of the transactions
```

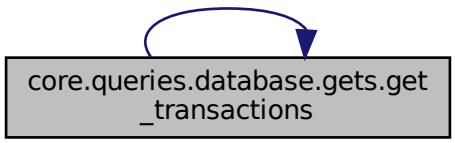
Definition at line 397 of file database.py.

```
397     def get_transactions(self, st_dt, end_dt=dt.datetime.now()):
398         """ get the transactions within the given period exclusively
399
400         @param st_dt: the start datetime
401         @param end_dt: the end datetime
402         @return dataframe of the transactions
403         """
404         stat = "SELECT * FROM ledger;"
405         if st_dt==None:
406             stat=sql.SQL("SELECT * FROM ledger WHERE trx_dt>{st_dt} AND trx_dt<{end_dt}  FOR UPDATE SKIP
        LOCKED;").\
407             format(st_dt=sql.Literal(st_dt), end_dt=sql.Literal(end_dt))
408         self.db_log.debug(stat)
409         return pd.read_sql(stat, self.conn)
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.↩ queries.database.gets.conn, queries.database.inserts.conn, queries.database.updates.conn, queries.database.↩ database.conn, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_↩ log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.gets.get_transactions().

Here is the caller graph for this function:



7.11.3.43 get_transactions_sum() [1/2]

```
def core.queries.database.gets.get_transactions_sum (
              self,
              trx_with_credid,
              st_dt = None,
              end_dt = None )


get the transactions within the given period inclusively

@param trx_with_credid: the credential id of the client of interest
@param st_dt: the start datetime
@param end_dt: the end datetime
@return dataframe of the transactions
```

Definition at line 410 of file database.py.

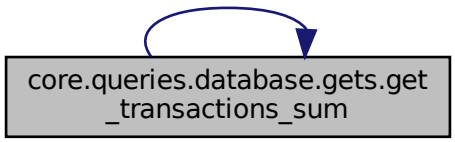
```
410     def get_transactions_sum(self, \
411                              trx_with_credid, \
412                              st_dt=None, \
413                              end_dt=None):
414         """ get the transactions within the given period inclusively
415
416         @param trx_with_credid: the credential id of the client of interest
417         @param st_dt: the start datetime
418         @param end_dt: the end datetime
419         @return dataframe of the transactions
420         """
421         if end_dt==None:
422             end_dt=dt.datetime.now().strftime(TIMESTAMP_FORMAT)
423
424         stat = "SELECT (ledger.trx_cost, cur.currency_name) FROM ledger INNER JOIN currency AS cur ON
       (cur.id=ledger.trx_cur_id) WHERE (trx_dest={to_credid} OR trx_src={from_credid});".\
425             format(to_credid=sql.Literal(trx_with_credid),\
426                    from_credid=sql.Literal(trx_with_credid))
427
428         if not st_dt==None:
429             #note! FOR UPDATE is not allowed with aggregate functions
430             stat=sql.SQL("SELECT (ledger.trx_cost, cur.currency_name) FROM ledger INNER JOIN currency AS
       cur ON (cur.id=ledger.trx_cur_id) WHERE (trx_dt>={st_dt} AND trx_dt<{end_dt} AND
       trx_dest={to_credid}) OR (trx_dt>={st_dt} AND trx_dt<{end_dt} AND trx_src={from_credid});").\
```

```
431             format(st_dt=sql.Literal(st_dt),\
432                    end_dt=sql.Literal(end_dt),\
433                    to_credid=sql.Literal(trx_with_credid), \
434                    from_credid=sql.Literal(trx_with_credid))
435         self.db_log.debug(stat)
436         #self.cur.execute(stat)
437         #fet=self.cur.fetchone()[0]
438         #if fet==None:
439             #return 0
440         #return fet
441         trxs_df=pd.read_sql(stat, self.conn)
442         #the transaction sum in euros
443         sum=0
444         for i in range(len(trxs_df)):
445             row=eval(trxs_df.iloc[i][0])
446             value=row[0]
447             base=row[1]
448             currency = Currency(EUR, base)
449             ineuro_cost=currency.exchange(value)
450             sum+=float(ineuro_cost)
451         return sum
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.↩ queries.database.gets.conn, queries.database.inserts.conn, queries.database.updates.conn, queries.database.↩ database.conn, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_↩ log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.gets.get_transactions_sum().

Here is the caller graph for this function:



7.11.3.44 get_transactions_sum() [2/2]

```
def core.queries.database.gets.get_transactions_sum (
                self,
                trx_with_credid,
                st_dt = None,
                end_dt = None )
```

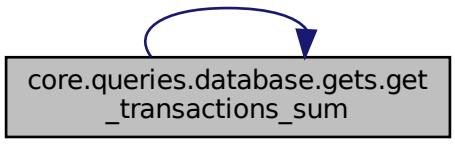
```
get the transactions within the given period inclusively

@param trx_with_credid: the credential id of the client of interest
@param st_dt: the start datetime
@param end_dt: the end datetime
@return dataframe of the transactions
```

Definition at line 410 of file database.py.

```
410     def get_transactions_sum(self, \
411                              trx_with_credid, \
412                              st_dt=None, \
413                              end_dt=None):
414         """ get the transactions within the given period inclusively
415 
416         @param trx_with_credid: the credential id of the client of interest
417         @param st_dt: the start datetime
418         @param end_dt: the end datetime
419         @return dataframe of the transactions
420         """
421         if end_dt==None:
422             end_dt=dt.datetime.now().strftime(TIMESTAMP_FORMAT)
423 
424         stat = "SELECT (ledger.trx_cost, cur.currency_name) FROM ledger INNER JOIN currency AS cur ON
    (cur.id=ledger.trx_cur_id) WHERE (trx_dest={to_credid} OR trx_src={from_credid});".\
425             format(to_credid=sql.Literal(trx_with_credid),\
426                    from_credid=sql.Literal(trx_with_credid))
427 
428         if not st_dt==None:
429             #note! FOR UPDATE is not allowed with aggregate functions
430             stat=sql.SQL("SELECT (ledger.trx_cost, cur.currency_name) FROM ledger INNER JOIN currency AS
    cur ON (cur.id=ledger.trx_cur_id) WHERE (trx_dt>={st_dt} AND trx_dt<{end_dt} AND
    trx_dest={to_credid}) OR (trx_dt>={st_dt} AND trx_dt<{end_dt} AND trx_src={from_credid});").\
431             format(st_dt=sql.Literal(st_dt),\
432                    end_dt=sql.Literal(end_dt),\
433                    to_credid=sql.Literal(trx_with_credid), \
434                    from_credid=sql.Literal(trx_with_credid))
435         self.db_log.debug(stat)
436         #self.cur.execute(stat)
437         #fet=self.cur.fetchone()[0]
438         #if fet==None:
439             #return 0
440         #return fet
441         trxs_df=pd.read_sql(stat, self.conn)
442         #the transaction sum in euros
443         sum=0
444         for i in range(len(trxs_df)):
445             row=eval(trxs_df.iloc[i][0])
446             value=row[0]
447             base=row[1]
448             currency = Currency(EUR, base)
449             ineuro_cost=currency.exchange(value)
450             sum+=float(ineuro_cost)
451         return sum
```

References queries.database.exists.conn, core.queries.database.exists.conn, queries.database.gets.conn, core.↩ queries.database.gets.conn, queries.database.inserts.conn, queries.database.updates.conn, queries.database.↩ database.conn, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_↩ log, queries.database.updates.db_log, server.server.debug, client.client.format, and core.queries.database.gets.↩ get_transactions_sum().

Here is the call graph for this function:



7.11.3.45 to_euro() [1/2]

```
def core.queries.database.gets.to_euro (
              self,
              cid )
```

```
convert currency of the corresponding id to euro ratio

for example if currency A = 2 euros, then the conversion would be 0.5,
for another currency B = 0.5 euros, then the conversion to euro would be 2
such that for given cost of xA, would be 0.5x$.
@param cid is the id of the corresponding currency
@return transformation ratio to euros
```

Definition at line 324 of file database.py.

```
324     def to_euro(self, cid):
325         """ convert currency of the corresponding id to euro ratio
326 
327         for example if currency A = 2 euros, then the conversion would be 0.5,
328         for another currency B = 0.5 euros, then the conversion to euro would be 2
329         such that for given cost of xA, would be 0.5x$.
330         @param cid is the id of the corresponding currency
331         @return transformation ratio to euros
332         """
333         query = sql.SQL("SELECT currency_value FROM currency WHERE id={cid} LIMIT 1 FOR UPDATE SKIP
      LOCKED;").\
334             format(cid=sql.Literal(cid))
335         self.db_log.debug(query)
336         self.cur.execute(query)
337         fet=self.cur.fetchone()
338         #ratio = 1.0/pd.read_sql(query, self.conn)['currency_value'].ix[0]
339         return fet[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, client.client.format, and core.queries.database.gets.to_↩ euro().

Here is the call graph for this function:

7.11.3.46 to_euro() [2/2]

```
def core.queries.database.gets.to_euro (
              self,
              cid )
```

```
convert currency of the corresponding id to euro ratio

for example if currency A = 2 euros, then the conversion would be 0.5,
for another currency B = 0.5 euros, then the conversion to euro would be 2
such that for given cost of xA, would be 0.5x$.
@param cid is the id of the corresponding currency
@return transformation ratio to euros
```

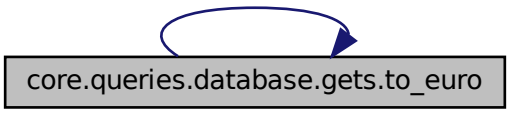
Definition at line 324 of file database.py.

```
324     def to_euro(self, cid):
325         """ convert currency of the corresponding id to euro ratio
326 
327         for example if currency A = 2 euros, then the conversion would be 0.5,
328         for another currency B = 0.5 euros, then the conversion to euro would be 2
329         such that for given cost of xA, would be 0.5x$.
330         @param cid is the id of the corresponding currency
331         @return transformation ratio to euros
332         """
333         query = sql.SQL("SELECT currency_value FROM currency WHERE id={cid} LIMIT 1 FOR UPDATE SKIP
    LOCKED;").\
334             format(cid=sql.Literal(cid))
335         self.db_log.debug(query)
336         self.cur.execute(query)
337         fet=self.cur.fetchone()
338         #ratio = 1.0/pd.read_sql(query, self.conn)['currency_value'].ix[0]
339         return fet[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, queries.database.updates.cur, queries.database.↩ database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.gets.to_euro().

Here is the caller graph for this function:



7.11.4 Field Documentation

7.11.4.1 conn

```
core.queries.database.gets.conn
```

Definition at line 106 of file database.py.

Referenced by core.queries.database.gets.__init__(), core.queries.database.inserts.__init__(), core.queries.↩ database.updates.__init__(), core.queries.database.database.__init__(), core.queries.database.database.close(), core.queries.database.database.commit(), core.queries.database.gets.get_all_clients(), core.queries.database.↩ gets.get_all_contacts(), core.queries.database.gets.get_all_credentials(), core.queries.database.gets.get_↩ credential(), core.queries.database.gets.get_sells(), core.queries.database.gets.get_transactions(), core.queries.↩ database.gets.get_transactions_sum(), core.queries.database.database.init(), and core.queries.database.↩ database.rollback().

7.11.4.2 cur

`core.queries.database.gets.cur`

Definition at line 107 of file database.py.

Referenced by core.queries.database.gets.__init__(), core.queries.database.inserts.__init__(), core.queries.database.updates.__init__(), core.queries.database.database.__init__(), core.queries.database.inserts.add_bank_account(), core.queries.database.inserts.add_client(), core.queries.database.inserts.add_currency(), core.queries.database.gets.cid2credid(), core.queries.database.database.committed_read(), core.queries.database.gets.credid2cid(), core.queries.database.updates.currency_preference(), core.queries.database.gets.get(), core.queries.database.gets.get_balance_by_cid(), core.queries.database.gets.get_balance_by_credid(), core.queries.database.gets.get_banking_id(), core.queries.database.gets.get_client_id(), core.queries.database.gets.get_client_id_byemail(), core.queries.database.gets.get_client_id_byname(), core.queries.database.gets.get_currency_id(), core.queries.database.gets.get_currency_name(), core.queries.database.gets.get_last_timestamp(), core.queries.database.gets.get_password(), core.queries.database.gets.get_preference_currency_bycid(), core.queries.database.database.init(), core.queries.database.inserts.insert_contact(), core.queries.database.inserts.insert_trx(), core.queries.database.database.lock_advisory(), core.queries.database.inserts.register(), core.queries.database.database.repeatable_read(), core.queries.database.gets.to_euro(), core.queries.database.database.unlock_advisory(), and core.queries.database.updates.update_account().

7.11.4.3 db_log

`core.queries.database.gets.db_log`

Definition at line 108 of file database.py.

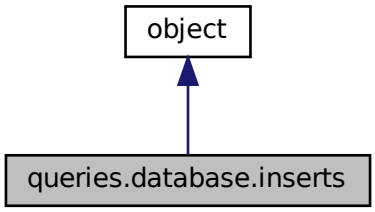
Referenced by core.queries.database.gets.__init__(), core.queries.database.inserts.__init__(), core.queries.database.updates.__init__(), core.queries.database.inserts.add_bank_account(), core.queries.database.gets.cid2credid(), core.queries.database.gets.credid2cid(), core.queries.database.gets.get(), core.queries.database.gets.get_all_clients(), core.queries.database.gets.get_all_contacts(), core.queries.database.gets.get_all_credentials(), core.queries.database.gets.get_balance_by_cid(), core.queries.database.gets.get_balance_by_credid(), core.queries.database.gets.get_banking_id(), core.queries.database.gets.get_client_id(), core.queries.database.gets.get_client_id_byemail(), core.queries.database.gets.get_client_id_byname(), core.queries.database.gets.get_credential(), core.queries.database.gets.get_currency_id(), core.queries.database.gets.get_currency_name(), core.queries.database.gets.get_last_timestamp(), core.queries.database.gets.get_password(), core.queries.database.gets.get_preference_currency_bycid(), core.queries.database.gets.get_sells(), core.queries.database.gets.get_transactions(), core.queries.database.gets.get_transactions_sum(), core.queries.database.inserts.insert_contact(), core.queries.database.inserts.insert_trx(), core.queries.database.inserts.register(), and core.queries.database.gets.to_euro().

The documentation for this class was generated from the following file:

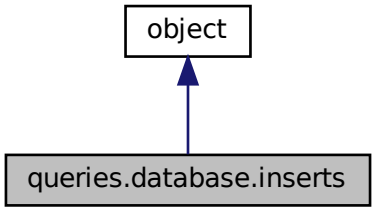
- core/queries/database.py

7.12 queries.database.inserts Class Reference

Inheritance diagram for queries.database.inserts:



Collaboration diagram for queries.database.inserts:



Public Member Functions

- def __init__ (self, conn, cur, logger)
- def add_bank_account (self, cid, balance, bank_name, branch_number, account_number, name_reference)
- def add_client (self, name, email)
- def insert_contact (self, email, name, credid)
- def register (self, cid, passcode, cred_id)
- def insert_trx (self, des, src, cost)

Data Fields

- conn
- cur
- db_log

7.12.1 Detailed Description

Definition at line 453 of file database.py.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 __init__()

```
def queries.database.inserts.__init__ (
              self,
              conn,
              cur,
              logger )
```

Definition at line 454 of file database.py.

```
454     def __init__(self, conn, cur, logger):
455         self.conn=conn
456         self.cur=cur
457         self.db_log=logger
```

7.12.3 Member Function Documentation

7.12.3.1 add_bank_account()

```
def queries.database.inserts.add_bank_account (
              self,
              cid,
              balance,
              bank_name,
              branch_number,
              account_number,
              name_reference )
```

```
give the client with the given id (cid) banking account (CALLED AT SERVER SIDE)

@param cid: client id
@param balance: client account balance
```

Definition at line 458 of file database.py.

```
458     def add_bank_account(self, cid, balance, bank_name, branch_number, account_number, name_reference):
459         """ give the client with the given id (cid) banking account (CALLED AT SERVER SIDE)
460 
461         @param cid: client id
462         @param balance: client account balance
463         """
464         stat=sql.SQL("INSERT INTO banking (client_id, balance,  bank_name, branch_number,
       account_number, name_reference) VALUES ({cid}, {balance}, {bname}, {bnum}, {anum}, {nr});"). \
465             format(cid=sql.Literal(cid), \
466                    balance=sql.Literal(balance), \
467                    bname=sql.Literal(bank_name),
468                    bnum=sql.Literal(branch_number),
```

```
469                     anum=sql.Literal(account_number),
470                     nr=sql.Literal(name_reference)
471                     )
472         self.db_log.debug(stat)
473         self.cur.execute(stat)
474         stat="SELECT currval(pg_get_serial_sequence('banking', 'id'));"
475         self.db_log.debug(stat)
476         self.cur.execute(stat);
477         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.↩ database.gets.db_log, queries.database.inserts.db_log, server.server.debug, and client.client.format.

7.12.3.2 add_client()

```
def queries.database.inserts.add_client (
              self,
              name,
              email )
```

```
add new client to the network (CALLED AT THE SERVER SIDE),

note that some clients might not have banking id yet
@param name: client name
@param email: client email
```

Definition at line 496 of file database.py.

```
496     def add_client(self, name, email):
497         """ add new client to the network (CALLED AT THE SERVER SIDE),
498
499         note that some clients might not have banking id yet
500         @param name: client name
501         @param email: client email
502         """
503         stat=sql.SQL("INSERT INTO clients (client_name, client_email) VALUES ({name}, {email})").\
504             format(name=sql.Literal(name),\
505                    email=sql.Literal(email))
506         self.cur.execute(stat)
507         self.cur.execute("SELECT currval(pg_get_serial_sequence('clients', 'client_id'));")
508         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, and client.↩ client.format.

7.12.3.3 insert_contact()

```
def queries.database.inserts.insert_contact (
              self,
              email,
              name,
              credid )
```

```
insert new contact (CALLED AT THE CLIENT SIDE)

@param email: contact's email
@param name: contact's name
@param credid: contact's credid
```

Definition at line 509 of file database.py.

```
509     def insert_contact(self, email, name, credid):
510         """ insert new contact (CALLED AT THE CLIENT SIDE)
511 
512 
513         @param email: contact's email
514         @param name: contact's name
515         @param credid: contact's credid
516         """
517         stat=sql.SQL("INSERT INTO contacts (contact_id, contact_name, contact_email) VALUES ({credid},
    {email}, {name})").\
518             format(credid=sql.Literal(credid), \
519                    email=sql.Literal(email), \
520                    name=sql.Literal(name))
521         self.db_log.debug(stat)
522         self.cur.execute(stat)
523 
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.↩database.gets.db_log, queries.database.inserts.db_log, server.server.debug, and client.client.format.

7.12.3.4 insert_trx()

```
def queries.database.inserts.insert_trx (
              self,
              des,
              src,
              cost )
```

```
insert transaction from 'src' to 'des' for good with 'gid'

@param des: the transaction destination
@param src: the transaction source
@param cost: the transaction amount
```

Definition at line 558 of file database.py.

```
558     def insert_trx(self, des, src, cost):
559         """ insert transaction from 'src' to 'des' for good with 'gid'
560 
561         @param des: the transaction destination
562         @param src: the transaction source
563         @param cost: the transaction amount
564         """
565         stat=sql.SQL("INSERT INTO ledger (trx_dest, trx_src, trx_cost) VALUES ({des}, {src},
    {cost});").\
566             format(des=sql.Literal(des), \
567                    src=sql.Literal(src), \
568                    cost=sql.Literal(cost))
569         self.db_log.debug(stat)
570         self.cur.execute(stat)
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.↩database.gets.db_log, queries.database.inserts.db_log, server.server.debug, and client.client.format.

7.12.3.5 register()

```
def queries.database.inserts.register (
              self,
              cid,
              passcode,
              cred_id )

add client credentials returned from the server

@param cid: client id
@param passcode: client password
@param cred_id: credential id
```

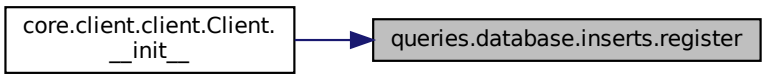
Definition at line 524 of file database.py.

```
524     def register(self, cid, passcode, cred_id):
525         """add client credentials returned from the server
526 
527         @param cid: client id
528         @param passcode: client password
529         @param cred_id: credential id
530         """
531         stat=sql.SQL("INSERT INTO credentials (id, passcode, cred_id) VALUES ({cid}, {passcode},
      {credid});").\
532             format(cid=sql.Literal(cid),\
533                    passcode=sql.Literal(passcode), \
534                    credid=sql.Literal(cred_id))
535         self.db_log.debug(stat)
536         self.cur.execute(stat)
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.database.gets.db_log, queries.database.inserts.db_log, server.server.debug, and client.client.format.

Referenced by core.client.client.Client.__init__().

Here is the caller graph for this function:



7.12.4 Field Documentation

7.12.4.1 conn

```
queries.database.inserts.conn
```

Definition at line 455 of file database.py.

Referenced by core.queries.database.exists.__init__(), core.queries.database.gets.__init__(), core.queries.database.inserts.__init__(), core.queries.database.updates.__init__(), core.queries.database.database.__init__(), queries.database.database.close(), core.queries.database.database.close(), queries.database.database.commit(), core.queries.database.database.commit(), core.queries.database.gets.get_all_clients(), core.queries.database.gets.get_all_contacts(), core.queries.database.gets.get_all_credentials(), core.queries.database.gets.get_credential(), core.queries.database.gets.get_sells(), core.queries.database.gets.get_transactions(), core.queries.database.gets.get_transactions_sum(), queries.database.database.init(), core.queries.database.database.init(), queries.database.database.rollback(), and core.queries.database.database.rollback().

7.12.4.2 cur

queries.database.inserts.cur

Definition at line 456 of file database.py.

Referenced by core.queries.database.exists.__init__(), core.queries.database.gets.__init__(), core.queries.database.inserts.__init__(), core.queries.database.updates.__init__(), core.queries.database.database.__init__(), core.queries.database.exists.account_byemail(), core.queries.database.exists.account_byname(), queries.database.inserts.add_bank_account(), core.queries.database.inserts.add_bank_account(), queries.database.inserts.add_client(), core.queries.database.inserts.add_client(), core.queries.database.inserts.add_currency(), core.queries.database.exists.bank_account_bycid(), core.queries.database.gets.cid2credid(), core.queries.database.exists.client_exists(), queries.database.database.committed_read(), core.queries.database.database.committed_read(), core.queries.database.exists.contact_exists(), core.queries.database.exists.credential_exists(), core.queries.database.gets.credid2cid(), core.queries.database.exists.currency(), core.queries.database.updates.currency_preference(), core.queries.database.gets.get(), core.queries.database.gets.get_balance_by_cid(), core.queries.database.gets.get_balance_by_credid(), core.queries.database.gets.get_banking_id(), core.queries.database.gets.get_client_id(), core.queries.database.gets.get_client_id_byemail(), core.queries.database.gets.get_client_id_byname(), core.queries.database.gets.get_currency_id(), core.queries.database.gets.get_currency_name(), core.queries.database.gets.get_last_timestamp(), core.queries.database.gets.get_password(), core.queries.database.gets.get_preference_currency_bycid(), queries.database.database.init(), core.queries.database.database.init(), queries.database.inserts.insert_contact(), core.queries.database.inserts.insert_contact(), queries.database.inserts.insert_trx(), core.queries.database.inserts.insert_trx(), queries.database.database.lock_advisory(), core.queries.database.database.lock_advisory(), queries.database.inserts.register(), core.queries.database.inserts.register(), queries.database.database.repeatable_read(), core.queries.database.database.repeatable_read(), core.queries.database.gets.to_euro(), queries.database.database.unlock_advisory(), core.queries.database.database.unlock_advisory(), queries.database.updates.update_account(), and core.queries.database.updates.update_account().

7.12.4.3 db_log

queries.database.inserts.db_log

Definition at line 457 of file database.py.

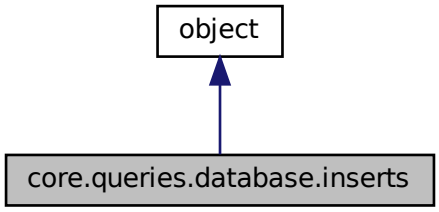
Referenced by core.queries.database.gets.__init__(), core.queries.database.inserts.__init__(), core.queries.database.updates.__init__(), queries.database.inserts.add_bank_account(), core.queries.database.inserts.add_bank_account(), core.queries.database.gets.cid2credid(), core.queries.database.gets.credid2cid(), core.queries.database.gets.get(), core.queries.database.gets.get_all_clients(), core.queries.database.gets.get_all_contacts(), core.queries.database.gets.get_all_credentials(), core.queries.database.gets.get_balance_by_cid(), core.queries.database.gets.get_balance_by_credid(), core.queries.database.gets.get_banking_id(), core.queries.database.gets.get_client_id(), core.queries.database.gets.get_client_id_byemail(), core.queries.database.gets.get_client_id_byname(), core.queries.database.gets.get_credential(), core.queries.database.gets.get_currency_id(), core.queries.database.gets.get_currency_name(), core.queries.database.gets.get_last_timestamp(), core.queries.database.gets.get_password(), core.queries.database.gets.get_preference_currency_bycid(), core.queries.database.gets.get_sells(), core.queries.database.gets.get_transactions(), core.queries.database.gets.get_transactions_sum(), queries.database.inserts.insert_contact(), core.queries.database.inserts.insert_contact(), queries.database.inserts.insert_trx(), core.queries.database.inserts.insert_trx(), queries.database.inserts.register(), core.queries.database.inserts.register(), and core.queries.database.gets.to_euro().

The documentation for this class was generated from the following file:

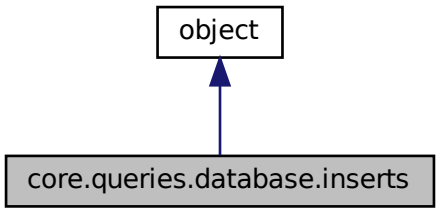
- core/build/lib/queries/database.py

7.13 core.queries.database.inserts Class Reference

Inheritance diagram for core.queries.database.inserts:



Collaboration diagram for core.queries.database.inserts:



Public Member Functions

- def __init__ (self, conn, cur, logger)
- def add_currency (self, cur_name, cur_rate)
- def add_bank_account (self, cid, balance, bank_name, branch_number, account_number, name_reference, base_currency_id)
- def add_client (self, name, email, cur_pref_id)
- def insert_contact (self, email, name, credid)
- def register (self, cid, passcode, cred_id)
- def insert_trx (self, des, src, cost, cur_id, name)
- def __init__ (self, conn, cur, logger)
- def add_currency (self, cur_name, cur_rate)
- def add_bank_account (self, cid, balance, bank_name, branch_number, account_number, name_reference, base_currency_id)
- def add_client (self, name, email, cur_pref_id)
- def insert_contact (self, email, name, credid)
- def register (self, cid, passcode, cred_id)
- def insert_trx (self, des, src, cost, cur_id, name)

Data Fields

- conn
- cur
- db_log

7.13.1 Detailed Description

Definition at line 512 of file database.py.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 __init__() [1/2]

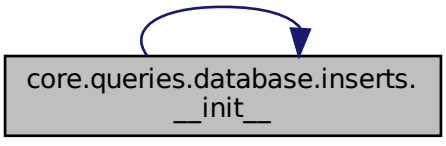
```
def core.queries.database.inserts.__init__ (
                self,
                conn,
                cur,
                logger )
```

Definition at line 513 of file database.py.

```
513     def __init__(self, conn, cur, logger):
514         self.conn=conn
515         self.cur=cur
516         self.db_log=logger
```

Referenced by core.queries.database.inserts.__init__().

Here is the caller graph for this function:



7.13.2.2 __init__() [2/2]

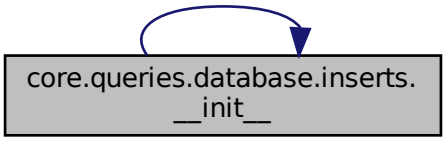
```
def core.queries.database.inserts.__init__ (
              self,
              conn,
              cur,
              logger )
```

Definition at line 513 of file database.py.

```
513     def __init__(self, conn, cur, logger):
514         self.conn=conn
515         self.cur=cur
516         self.db_log=logger
```

References core.queries.database.inserts.__init__(), queries.database.exists.conn, core.queries.database.↩ exists.conn, queries.database.gets.conn, core.queries.database.gets.conn, queries.database.inserts.conn, core.↩ queries.database.inserts.conn, queries.database.updates.conn, queries.database.database.conn, queries.↩ database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.queries.database.gets.↩ cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.updates.cur, queries.↩ database.database.cur, queries.database.gets.db_log, core.queries.database.gets.db_log, queries.database.↩ inserts.db_log, core.queries.database.inserts.db_log, and queries.database.updates.db_log.

Here is the call graph for this function:



7.13.3 Member Function Documentation

7.13.3.1 add_bank_account() [1/2]

```
def core.queries.database.inserts.add_bank_account (
              self,
              cid,
              balance,
              bank_name,
              branch_number,
              account_number,
              name_reference,
              base_currency_id )
```

```
give the client with the given id (cid) banking account (CALLED AT SERVER SIDE)

@param cid: client id
@param balance: client account balance
```

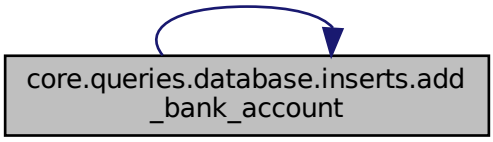
Definition at line 528 of file database.py.

```
528     def add_bank_account(self, cid, balance, bank_name, branch_number, account_number, name_reference,
        base_currency_id):
529         """ give the client with the given id (cid) banking account (CALLED AT SERVER SIDE)
530 
531         @param cid: client id
532         @param balance: client account balance
533         """
534         stat=sql.SQL("INSERT INTO banking (client_id, balance,  bank_name, branch_number,
        account_number, name_reference, currency_id) VALUES ({cid}, {balance}, {bname}, {bnum}, {anum}, {nr},
        {base_currency});").\
535             format(cid=sql.Literal(cid), \
536                    balance=sql.Literal(balance), \
537                    bname=sql.Literal(bank_name), \
538                    bnum=sql.Literal(branch_number), \
539                    anum=sql.Literal(account_number), \
540                    nr=sql.Literal(name_reference), \
541                    base_currency=sql.Literal(base_currency_id))
542         self.db_log.debug(stat)
543         self.cur.execute(stat)
544         stat="SELECT currval(pg_get_serial_sequence('banking', 'id'));"
545         self.db_log.debug(stat)
546         self.cur.execute(stat)
547         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩ updates.cur, queries.database.database.cur, queries.database.gets.db_log, core.queries.database.gets.db_↩ log, queries.database.inserts.db_log, core.queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.inserts.add_bank_account().

Here is the caller graph for this function:



7.13.3.2 add_bank_account() [2/2]

```
def core.queries.database.inserts.add_bank_account (
                self,
                cid,
                balance,
                bank_name,
                branch_number,
                account_number,
                name_reference,
                base_currency_id )
```

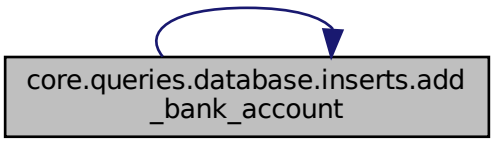
```
give the client with the given id (cid) banking account (CALLED AT SERVER SIDE)

@param cid: client id
@param balance: client account balance
```

Definition at line 528 of file database.py.

```
528     def add_bank_account(self, cid, balance, bank_name, branch_number, account_number, name_reference,
      base_currency_id):
529         """ give the client with the given id (cid) banking account (CALLED AT SERVER SIDE)
530 
531         @param cid: client id
532         @param balance: client account balance
533         """
534         stat=sql.SQL("INSERT INTO banking (client_id, balance,  bank_name, branch_number,
      account_number, name_reference, currency_id) VALUES ({cid}, {balance}, {bname}, {bnum}, {anum}, {nr},
      {base_currency});").\
535             format(cid=sql.Literal(cid), \
536                    balance=sql.Literal(balance), \
537                    bname=sql.Literal(bank_name), \
538                    bnum=sql.Literal(branch_number), \
539                    anum=sql.Literal(account_number), \
540                    nr=sql.Literal(name_reference), \
541                    base_currency=sql.Literal(base_currency_id))
542         self.db_log.debug(stat)
543         self.cur.execute(stat)
544         stat="SELECT currval(pg_get_serial_sequence('banking', 'id'));"
545         self.db_log.debug(stat)
546         self.cur.execute(stat)
547         return self.cur.fetchone()[0]
```

References core.queries.database.inserts.add_bank_account(), queries.database.exists.cur, core.queries.↩ database.exists.cur, queries.database.gets.cur, core.queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.updates.cur, queries.database.database.cur, queries.↩ database.gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, core.queries.↩ database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Here is the call graph for this function:



7.13.3.3 add_client() [1/2]

```
def core.queries.database.inserts.add_client (
                self,
                name,
                email,
                cur_pref_id )
```

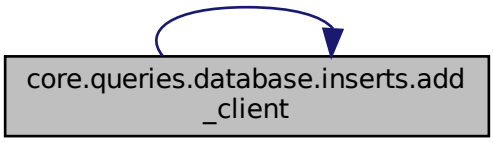
```
add new client to the network (CALLED AT THE SERVER SIDE),

note that some clients might not have banking id yet
@param name: client name
@param email: client email
@param cur_pref: currency preference id
```

Definition at line 548 of file database.py.

```
548     def add_client(self, name, email, cur_pref_id):
549         """ add new client to the network (CALLED AT THE SERVER SIDE),
550 
551         note that some clients might not have banking id yet
552         @param name: client name
553         @param email: client email
554         @param cur_pref: currency preference id
555         """
556         stat=sql.SQL("INSERT INTO clients (client_name, client_email, currency_id) VALUES ({name},
        {email}, {cur_pref})").\
557             format(name=sql.Literal(name),\
558                    email=sql.Literal(email),\
559                    cur_pref=sql.Literal(cur_pref_id))
560         self.cur.execute(stat)
561         self.cur.execute("SELECT currval(pg_get_serial_sequence('clients', 'client_id'));")
562         return self.cur.fetchone()[0]
```

References core.queries.database.inserts.add_client(), core.queries.database.exists.cur, queries.database.↩ exists.cur, queries.database.gets.cur, core.queries.database.gets.cur, queries.database.inserts.cur, core.queries.↩ database.inserts.cur, queries.database.updates.cur, queries.database.database.cur, and client.client.format.

Here is the call graph for this function:



7.13.3.4 add_client() [2/2]

```
def core.queries.database.inserts.add_client (
              self,
              name,
              email,
              cur_pref_id )


add new client to the network (CALLED AT THE SERVER SIDE),

note that some clients might not have banking id yet
@param name: client name
@param email: client email
@param cur_pref: currency preference id
```

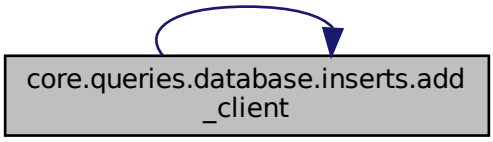
Definition at line 548 of file database.py.

```
548     def add_client(self, name, email, cur_pref_id):
549         """ add new client to the network (CALLED AT THE SERVER SIDE),
550 
551         note that some clients might not have banking id yet
552         @param name: client name
553         @param email: client email
554         @param cur_pref: currency preference id
555         """
556         stat=sql.SQL("INSERT INTO clients (client_name, client_email, currency_id) VALUES ({name},
        {email}, {cur_pref})").\
557             format(name=sql.Literal(name),\
558                    email=sql.Literal(email),\
559                    cur_pref=sql.Literal(cur_pref_id))
560         self.cur.execute(stat)
561         self.cur.execute("SELECT currval(pg_get_serial_sequence('clients', 'client_id'));")
562         return self.cur.fetchone()[0]
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩ updates.cur, queries.database.database.cur, and client.client.format.

Referenced by core.queries.database.inserts.add_client().

Here is the caller graph for this function:



7.13.3.5 add_currency() [1/2]

```
def core.queries.database.inserts.add_currency (
                self,
                cur_name,
                cur_rate )


support new currency

@param cur_name: currency_name
@param cur_rate: currency rate based by euro
```

Definition at line 517 of file database.py.

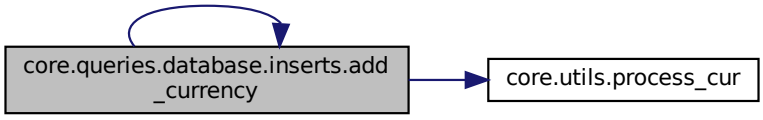
```
517     def add_currency(self, cur_name, cur_rate):
518         """support new currency
519 
520         @param cur_name: currency_name
521         @param cur_rate: currency rate based by euro
522         """
523         stat=sql.SQL("INSERT INTO currency (currency_name, currency_value) VALUES ({cur_name},
        {cur_val});").\
524             format(cur_name=sql.Literal(process_cur(cur_name)), \
525                    cur_val=sql.Literal(cur_rate))
```

```
526         print('currency added name: {}, and rate: {}'.format(cur_name, cur_rate))
527         self.cur.execute(stat)
```

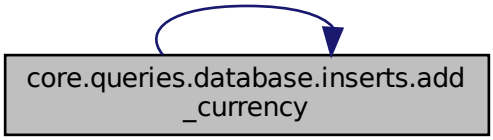
References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩ updates.cur, queries.database.database.cur, client.client.format, and core.utils.process_cur().

Referenced by core.queries.database.inserts.add_currency().

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.3.6 add_currency() [2/2]

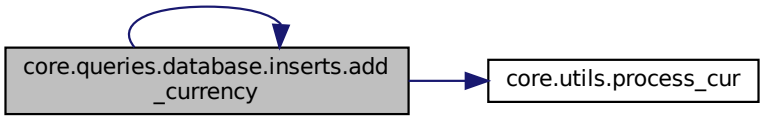
```
def core.queries.database.inserts.add_currency (
              self,
              cur_name,
              cur_rate )


support new currency

@param cur_name: currency_name
@param cur_rate: currency rate based by euro
```

Definition at line 517 of file database.py.

```
517     def add_currency(self, cur_name, cur_rate):
518         """support new currency
519 
520         @param cur_name: currency_name
521         @param cur_rate: currency rate based by euro
522         """
523         stat=sql.SQL("INSERT INTO currency (currency_name, currency_value) VALUES ({cur_name},
        {cur_val});").\
524             format(cur_name=sql.Literal(process_cur(cur_name)), \
525                    cur_val=sql.Literal(cur_rate))
526         print('currency added name: {}, and rate: {}'.format(cur_name, cur_rate))
527         self.cur.execute(stat)
```

References core.queries.database.inserts.add_currency(), queries.database.exists.cur, core.queries.database.↩ exists.cur, queries.database.gets.cur, core.queries.database.gets.cur, queries.database.inserts.cur, core.queries.↩ database.inserts.cur, queries.database.updates.cur, queries.database.database.cur, client.client.format, and core.utils.process_cur().

Here is the call graph for this function:



7.13.3.7 insert_contact() [1/2]

```
def core.queries.database.inserts.insert_contact (
                self,
                email,
                name,
                credid )


insert new contact (CALLED AT THE CLIENT SIDE)


@param email: contact's email
@param name: contact's name
@param credid: contact's credid
```

Definition at line 563 of file database.py.

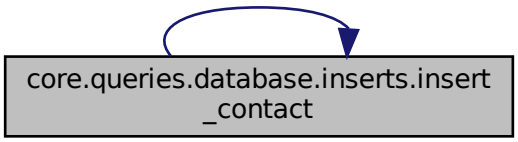
```
563     def insert_contact(self, email, name, credid):
564         """ insert new contact (CALLED AT THE CLIENT SIDE)
565 
566 
567         @param email: contact's email
568         @param name: contact's name
569         @param credid: contact's credid
570         """
571         stat=sql.SQL("INSERT INTO contacts (contact_id, contact_name, contact_email) VALUES ({credid},
        {email}, {name})").\
572             format(credid=sql.Literal(credid), \
573                    email=sql.Literal(email), \
574                    name=sql.Literal(name))
575         self.db_log.debug(stat)
```

```
576         self.cur.execute(stat)
577
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩ updates.cur, queries.database.database.cur, queries.database.gets.db_log, core.queries.database.gets.db_↩ log, queries.database.inserts.db_log, core.queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.inserts.insert_contact().

Here is the caller graph for this function:



7.13.3.8 insert_contact() [2/2]

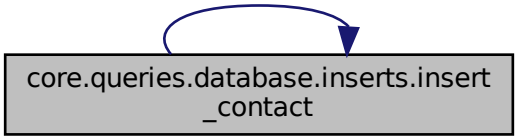
```
def core.queries.database.inserts.insert_contact (
              self,
              email,
              name,
              credid )


insert new contact (CALLED AT THE CLIENT SIDE)


@param email: contact's email
@param name: contact's name
@param credid: contact's credid
```

Definition at line 563 of file database.py.

```
563     def insert_contact(self, email, name, credid):
564         """ insert new contact (CALLED AT THE CLIENT SIDE)
565
566
567         @param email: contact's email
568         @param name: contact's name
569         @param credid: contact's credid
570         """
571         stat=sql.SQL("INSERT INTO contacts (contact_id, contact_name, contact_email) VALUES ({credid},
      {email}, {name})").\
572             format(credid=sql.Literal(credid), \
573                    email=sql.Literal(email), \
574                    name=sql.Literal(name))
575         self.db_log.debug(stat)
576         self.cur.execute(stat)
577
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩ updates.cur, queries.database.database.cur, queries.database.gets.db_log, core.queries.database.gets.db_↩ log, queries.database.inserts.db_log, core.queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, client.client.format, and core.queries.database.inserts.insert_contact().

Here is the call graph for this function:



7.13.3.9 insert_trx() [1/2]

```
def core.queries.database.inserts.insert_trx (
              self,
              des,
              src,
              cost,
              cur_id,
              name )


insert transaction from 'src' to 'des' for good with 'gid'

@param des: the transaction destination
@param src: the transaction source
@param cost: the transaction amount
@param cur_id: the currency id
@param name: the transaction name
```

Definition at line 591 of file database.py.

```
591     def insert_trx(self, des, src, cost, cur_id, name):
592         """ insert transaction from 'src' to 'des' for good with 'gid'
593 
594         @param des: the transaction destination
595         @param src: the transaction source
596         @param cost: the transaction amount
597         @param cur_id: the currency id
598         @param name: the transaction name
599         """
600         stat=sql.SQL("INSERT INTO ledger (trx_dest, trx_src, trx_cost, trx_cur_id, trx_name) VALUES
       ({des}, {src}, {cost}, {cur_id}, {name});").\
601             format(des=sql.Literal(des), \
602                    src=sql.Literal(src), \
603                    cost=sql.Literal(cost), \
604                    cur_id=sql.Literal(cur_id), \
605                    name=sql.Literal(name))
606         self.db_log.debug(stat)
607         self.cur.execute(stat)
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩ updates.cur, queries.database.database.cur, queries.database.gets.db_log, core.queries.database.gets.db_↩ log, queries.database.inserts.db_log, core.queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, client.client.format, and core.queries.database.inserts.insert_trx().

Here is the call graph for this function:

7.13.3.10 insert_trx() [2/2]

```
def core.queries.database.inserts.insert_trx (
              self,
              des,
              src,
              cost,
              cur_id,
              name )


insert transaction from 'src' to 'des' for good with 'gid'

@param des: the transaction destination
@param src: the transaction source
@param cost: the transaction amount
@param cur_id: the currency id
@param name: the transaction name
```

Definition at line 591 of file database.py.

```
591     def insert_trx(self, des, src, cost, cur_id, name):
592         """ insert transaction from 'src' to 'des' for good with 'gid'
593
594         @param des: the transaction destination
595         @param src: the transaction source
596         @param cost: the transaction amount
597         @param cur_id: the currency id
598         @param name: the transaction name
599         """
600         stat=sql.SQL("INSERT INTO ledger (trx_dest, trx_src, trx_cost, trx_cur_id, trx_name) VALUES
       ({des}, {src}, {cost}, {cur_id}, {name});").\
601             format(des=sql.Literal(des), \
602                    src=sql.Literal(src), \
603                    cost=sql.Literal(cost), \
604                    cur_id=sql.Literal(cur_id), \
605                    name=sql.Literal(name))
606         self.db_log.debug(stat)
607         self.cur.execute(stat)
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩ updates.cur, queries.database.database.cur, queries.database.gets.db_log, core.queries.database.gets.db_↩ log, queries.database.inserts.db_log, core.queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.inserts.insert_trx().

Here is the caller graph for this function:

7.13.3.11 register() [1/2]

```
def core.queries.database.inserts.register (
              self,
              cid,
              passcode,
              cred_id )


add client credentials returned from the server

@param cid: client id
@param passcode: client password
@param cred_id: credential id
```

Definition at line 578 of file database.py.

```
578     def register(self, cid, passcode, cred_id):
579         """add client credentials returned from the server
580 
581         @param cid: client id
582         @param passcode: client password
583         @param cred_id: credential id
584         """
585         stat=sql.SQL("INSERT INTO credentials (id, passcode, cred_id) VALUES ({cid}, {passcode},
      {credid});").\
586             format(cid=sql.Literal(cid),\
587                    passcode=sql.Literal(passcode), \
588                    credid=sql.Literal(cred_id))
589         self.db_log.debug(stat)
590         self.cur.execute(stat)
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩ updates.cur, queries.database.database.cur, queries.database.gets.db_log, core.queries.database.gets.db_↩ log, queries.database.inserts.db_log, core.queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, client.client.format, and core.queries.database.inserts.register().

Here is the call graph for this function:

7.13.3.12 register() [2/2]

```
def core.queries.database.inserts.register (
              self,
              cid,
              passcode,
              cred_id )



add client credentials returned from the server

@param cid: client id
@param passcode: client password
@param cred_id: credential id
```

Definition at line 578 of file database.py.

```
578     def register(self, cid, passcode, cred_id):
579         """add client credentials returned from the server
580 
581         @param cid: client id
582         @param passcode: client password
583         @param cred_id: credential id
584         """
585         stat=sql.SQL("INSERT INTO credentials (id, passcode, cred_id) VALUES ({cid}, {passcode},
      {credid});").\
586             format(cid=sql.Literal(cid),\
587                    passcode=sql.Literal(passcode), \
588                    credid=sql.Literal(cred_id))
589         self.db_log.debug(stat)
590         self.cur.execute(stat)
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩ updates.cur, queries.database.database.cur, queries.database.gets.db_log, core.queries.database.gets.db_↩ log, queries.database.inserts.db_log, core.queries.database.inserts.db_log, queries.database.updates.db_log, server.server.debug, and client.client.format.

Referenced by core.queries.database.inserts.register().

Here is the caller graph for this function:

7.13.4 Field Documentation

7.13.4.1 conn

`core.queries.database.inserts.conn`

Definition at line 514 of file database.py.

Referenced by core.queries.database.inserts.__init__(), core.queries.database.updates.__init__(), core.queries.database.database.__init__(), core.queries.database.database.close(), core.queries.database.database.commit(), core.queries.database.database.init(), and core.queries.database.database.rollback().

7.13.4.2 cur

`core.queries.database.inserts.cur`

Definition at line 515 of file database.py.

Referenced by core.queries.database.inserts.__init__(), core.queries.database.updates.__init__(), core.queries.database.database.__init__(), core.queries.database.inserts.add_bank_account(), core.queries.database.inserts.add_client(), core.queries.database.inserts.add_currency(), core.queries.database.database.committed_read(), core.queries.database.updates.currency_preference(), core.queries.database.database.init(), core.queries.database.inserts.insert_contact(), core.queries.database.inserts.insert_trx(), core.queries.database.database.lock_advisory(), core.queries.database.inserts.register(), core.queries.database.database.repeatable_read(), core.queries.database.database.unlock_advisory(), and core.queries.database.updates.update_account().

7.13.4.3 db_log

`core.queries.database.inserts.db_log`

Definition at line 516 of file database.py.

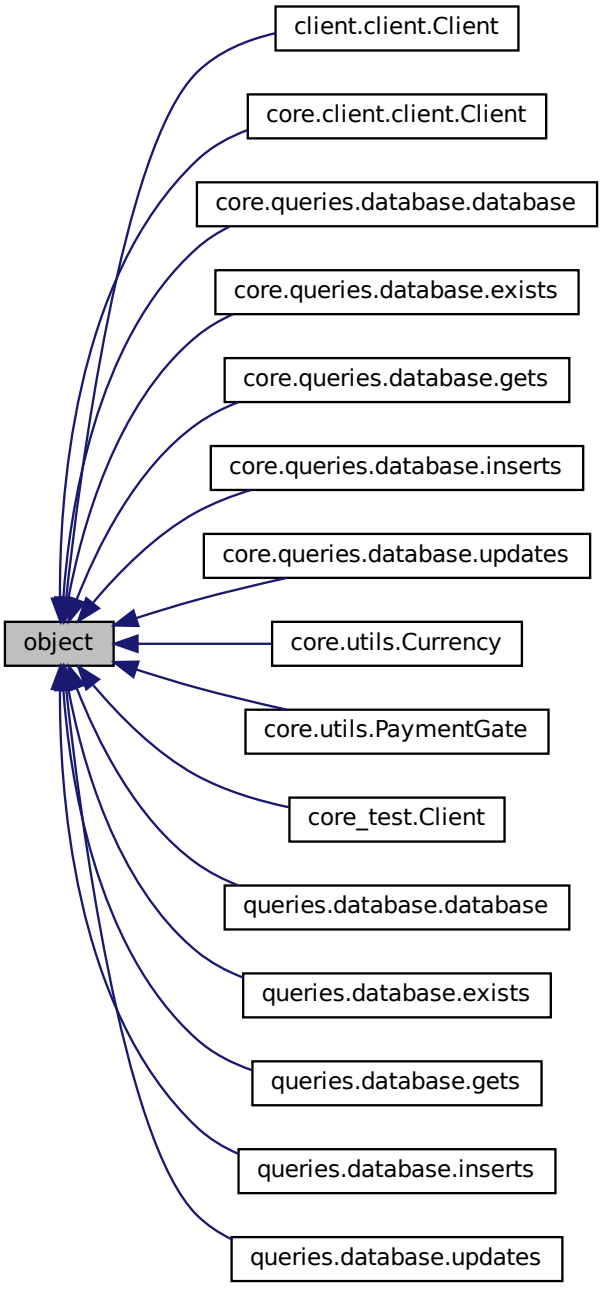
Referenced by core.queries.database.inserts.__init__(), core.queries.database.updates.__init__(), core.queries.database.inserts.add_bank_account(), core.queries.database.inserts.insert_contact(), core.queries.database.inserts.insert_trx(), and core.queries.database.inserts.register().

The documentation for this class was generated from the following file:

- core/queries/database.py

7.14 object Class Reference

Inheritance diagram for object:

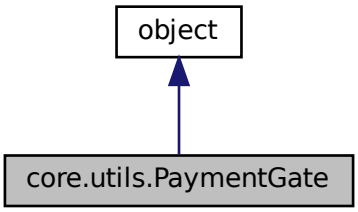


The documentation for this class was generated from the following file:

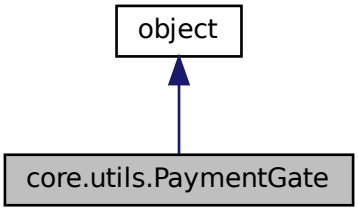
- core/utils.py

7.15 core.utils.PaymentGate Class Reference

Inheritance diagram for core.utils.PaymentGate:



Collaboration diagram for core.utils.PaymentGate:



Public Member Functions

- def __init__ (self, bank_name, branch_number, account_number, name_reference)
- def authenticated (self)
- def get_balance (self)
- def __init__ (self, bank_name, branch_number, account_number, name_reference)
- def authenticated (self)
- def get_balance (self)

Data Fields

- bank_name
- branch_number
- account_number
- name_reference
- base
- balance

Private Member Functions

- def __get_balance (self)
- def __get_balance (self)

7.15.1 Detailed Description

Definition at line 115 of file utils.py.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 __init__() [1/2]

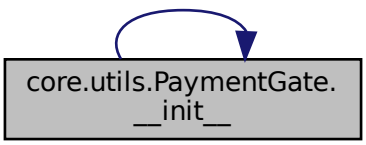
```
def core.utils.PaymentGate.__init__ (
              self,
              bank_name,
              branch_number,
              account_number,
              name_reference )
```

Definition at line 116 of file utils.py.

```
116     def __init__(self, bank_name, branch_number, account_number, name_reference):
117         self.bank_name=bank_name
118         self.branch_number=branch_number
119         self.account_number=account_number
120         self.name_reference=name_reference
121
```

Referenced by core.utils.PaymentGate.__init__().

Here is the caller graph for this function:



7.15.2.2 __init__() [2/2]

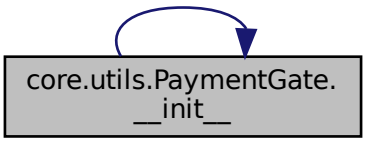
```
def core.utils.PaymentGate.__init__ (
              self,
              bank_name,
              branch_number,
              account_number,
              name_reference )
```

Definition at line 116 of file utils.py.

```
116     def __init__(self, bank_name, branch_number, account_number, name_reference):
117         self.bank_name=bank_name
118         self.branch_number=branch_number
119         self.account_number=account_number
120         self.name_reference=name_reference
121
```

References core.utils.PaymentGate.__init__(), core_test.Client.account_number, client.client.Client.account_number, core.client.client.Client.account_number, core.utils.PaymentGate.account_number, core_test.Client.bank_name, client.client.Client.bank_name, core.client.client.Client.bank_name, core.utils.PaymentGate.bank_name, core_test.Client.branch_number, client.client.Client.branch_number, core.client.client.Client.branch_number, core.utils.PaymentGate.branch_number, core_test.Client.name_reference, core.client.client.Client.name_reference, client.client.Client.name_reference, and core.utils.PaymentGate.name_reference.

Here is the call graph for this function:



7.15.3 Member Function Documentation

7.15.3.1 __get_balance() [1/2]

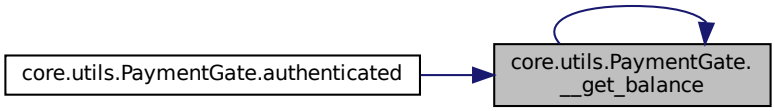
```
def core.utils.PaymentGate.__get_balance (
              self )  [private]
```

Definition at line 122 of file utils.py.

```
122     def __get_balance(self):
123         return random.random()*MAX_BALANCE
124
```

Referenced by core.utils.PaymentGate.__get_balance(), and core.utils.PaymentGate.authenticated().

Here is the caller graph for this function:



7.15.3.2 __get_balance() [2/2]

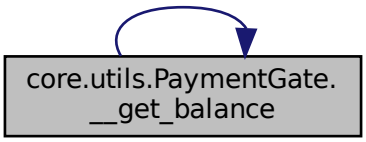
```
def core.utils.PaymentGate.__get_balance (
              self )  [private]
```

Definition at line 122 of file utils.py.

```
122     def __get_balance(self):
123         return random.random()*MAX_BALANCE
124
```

References core.utils.PaymentGate.__get_balance().

Here is the call graph for this function:



7.15.3.3 authenticated() [1/2]

```
def core.utils.PaymentGate.authenticated (
              self )
```

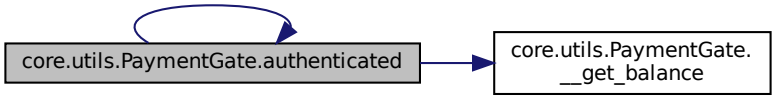
contact the corresponding banking infrastructure server for authentication, and retrieve the balance

Definition at line 125 of file utils.py.

```
125     def authenticated(self):
126         ''' contact the corresponding banking infrastructure server for authentication, and retrieve the
        balance
127         '''
128         self.base=EUR
129         self.balance=self.__get_balance()
130         return True
```

References core.utils.PaymentGate.__get_balance(), core.utils.PaymentGate.authenticated(), core.utils.Payment↩Gate.balance, core_test.Client.balance, client.client.Client.balance, core.client.client.Client.balance, core_test.↩RestfulTest.balance, core.utils.Currency.base, and core.utils.PaymentGate.base.

Here is the call graph for this function:



7.15.3.4 authenticated() [2/2]

```
def core.utils.PaymentGate.authenticated (
              self )
```

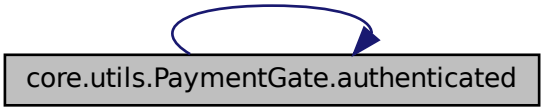
```
contact the corresponding banking infrastructure server for authentication, and retrieve the balance
```

Definition at line 125 of file utils.py.

```
125     def authenticated(self):
126         ''' contact the corresponding banking infrastructure server for authentication, and retrieve the
        balance
127         '''
128         self.base=EUR
129         self.balance=self.__get_balance()
130         return True
```

Referenced by core.utils.PaymentGate.authenticated().

Here is the caller graph for this function:



7.15.3.5 get_balance() [1/2]

```
def core.utils.PaymentGate.get_balance (
              self )
```

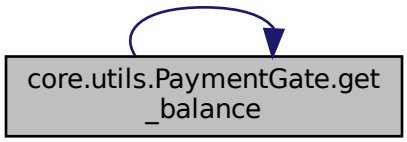
Definition at line 131 of file utils.py.

```
131     def get_balance(self):
132         return {'balance':self.balance, 'base':self.base}
133
```

References core.utils.PaymentGate.balance, core_test.Client.balance, core.client.client.Client.balance, client.↩client.Client.balance, core_test.RestfulTest.balance, core.utils.Currency.base, and core.utils.PaymentGate.base.

Referenced by core.utils.PaymentGate.get_balance().

Here is the caller graph for this function:



7.15.3.6 get_balance() [2/2]

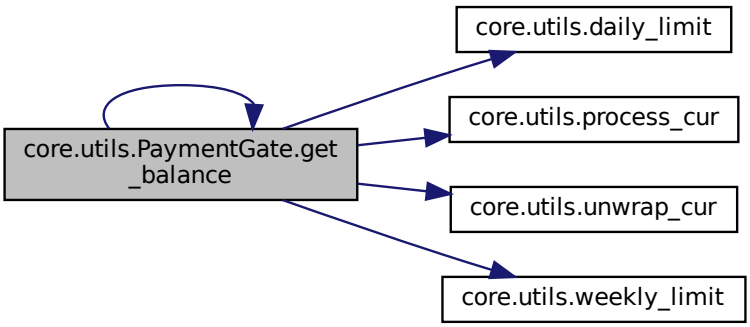
```
def core.utils.PaymentGate.get_balance (
              self )
```

Definition at line 131 of file utils.py.

```
131     def get_balance(self):
132         return {'balance':self.balance, 'base':self.base}
133
```

References core.utils.PaymentGate.balance, core_test.Client.balance, client.client.Client.balance, core.client.↩client.Client.balance, core_test.RestfulTest.balance, core.utils.Currency.base, core.utils.PaymentGate.base, core.utils.daily_limit(), core.utils.format, core.utils.PaymentGate.get_balance(), core.utils.process_cur(), core.↩utils.unwrap_cur(), and core.utils.weekly_limit().

Here is the call graph for this function:



7.15.4 Field Documentation

7.15.4.1 account_number

`core.utils.PaymentGate.account_number`

Definition at line 119 of file utils.py.

Referenced by core.utils.PaymentGate.__init__().

7.15.4.2 balance

`core.utils.PaymentGate.balance`

Definition at line 129 of file utils.py.

Referenced by core.utils.PaymentGate.authenticated(), and core.utils.PaymentGate.get_balance().

7.15.4.3 bank_name

`core.utils.PaymentGate.bank_name`

Definition at line 117 of file utils.py.

Referenced by core.utils.PaymentGate.__init__().

7.15.4.4 base

`core.utils.PaymentGate.base`

Definition at line 128 of file utils.py.

Referenced by core.utils.PaymentGate.authenticated(), and core.utils.PaymentGate.get_balance().

7.15.4.5 branch_number

`core.utils.PaymentGate.branch_number`

Definition at line 118 of file utils.py.

Referenced by core.utils.PaymentGate.__init__().

7.15.4.6 name_reference

`core.utils.PaymentGate.name_reference`

Definition at line 120 of file utils.py.

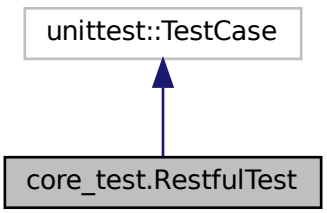
Referenced by core.utils.PaymentGate.__init__().

The documentation for this class was generated from the following file:

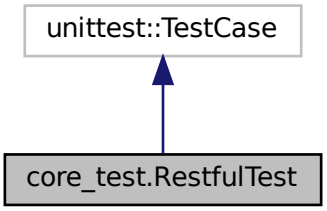
- core/utils.py

7.16 core_test.RestfulTest Class Reference

Inheritance diagram for core_test.RestfulTest:



Collaboration diagram for core_test.RestfulTest:



Public Member Functions

- def setUp (self)
- def test_register (self)
- def test_get_balance (self)
- def test_add_contact (self)
- def test_udapte_ledger (self)
- def test_make_transaction (self)
- def add_contact (self, email, name)
- def autotract (self)

Data Fields

- app
- uname
- pas
- balance

Private Member Functions

- def __auth (self)
- def __get_dict (self, ret)
- def __rand_alphanum (self, L=9)
- def __register_bank_account (self, bank_name=None, branch_number=None, account_number=None, name_reference=None)
- def __transact (self, credid, amount, currency='USD')
- def __add_trax (self, trans)
- def __update_balance (self, balance)
- def __ledger_timestamp (self)
- def __update_ledger (self)

7.16.1 Detailed Description

Definition at line 170 of file core_test.py.

7.16.2 Member Function Documentation

7.16.2.1 __add_trax()

```
def core_test.RestfulTest.__add_trax (
              self,
              trans )  [private]
```

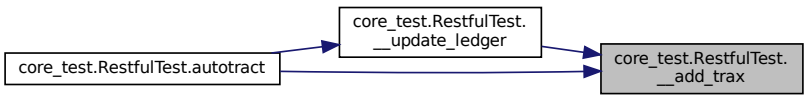
Definition at line 394 of file core_test.py.

```
394     def __add_trax(self, trans):
395         trans=trans['transactions'] #TODO (res)
396         self.db.init()
397         try:
398             print("trans: ", trans)
399             #self.db.inserts.insert_trx(trans["trx_src"], trans['trx_dest'], trans['trx_gid'])
400             self.db.inserts.insert_trx(trans["trx_src"], trans['trx_dest'], trans['trx_cost'])
401             self.db.commit()
402         except psycopg2.DatabaseError as error:
403             self.db.rollback()
404         finally:
405             self.db.close()
```

References core_test.Client.db, client.client.Client.db, and core.client.client.Client.db.

Referenced by core_test.RestfulTest.__update_ledger(), and core_test.RestfulTest.autotract().

Here is the caller graph for this function:



7.16.2.2 __auth()

```
def core_test.RestfulTest.__auth (
              self )  [private]
```

Definition at line 197 of file core_test.py.

```
197     def __auth(self):
198         self.assertFalse(self.uname==None)
199         self.assertFalse(self.pas==None)
200         valid_cred = \
201             base64.b64encode(bytes("{}:{}".format(self.uname, \
202                                                   self.pas), \
203                                    'utf-8')).decode("utf-8")
204         return  valid_cred
205
```

References core_test.format, core.client.client.Client.pas, client.client.Client.pas, core_test.RestfulTest.pas, core.↩client.client.Client.uname, client.client.Client.uname, and core_test.RestfulTest.uname.

7.16.2.3 __get_dict()

```
def core_test.RestfulTest.__get_dict (
              self,
              ret )  [private]
```

Definition at line 321 of file core_test.py.

```
321     def __get_dict(self, ret):
322         res=ret.decode('utf8')#.replace('"', "'")
323         return json.loads(res)
```

7.16.2.4 __ledger_timestamp()

```
def core_test.RestfulTest.__ledger_timestamp (
              self )  [private]
```

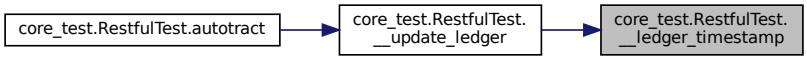
Definition at line 412 of file core_test.py.

```
412     def __ledger_timestamp(self):
413         dt=None
414         self.db.init()
415         try:
416             dt=self.db.gets.get_last_timestamp()
417             self.db.commmit()
418         except psycopg2.DatabaseError as error:
419             logger.critical("failed to retrieve ledger time stamp")
420             self.db.rollback()
421         finally:
422             self.db.close()
423         print("timestamp: ", dt)
424         return dt
```

References core_test.Client.db, client.client.Client.db, and core.client.client.Client.db.

Referenced by core_test.RestfulTest.__update_ledger().

Here is the caller graph for this function:



7.16.2.5 __rand_alphanum()

```
def core_test.RestfulTest.__rand_alphanum (
              self,
              L = 9 )  [private]
```

Definition at line 324 of file core_test.py.

```
324     def __rand_alphanum(self, L=9):
325         passcode=''.join(rand.choice(string.ascii_uppercase+\
326                                      string.ascii_lowercase+\
327                                      string.digits)\
328                          for _ in range(L))
329
```

7.16.2.6 __register_bank_account()

```
def core_test.RestfulTest.__register_bank_account (
                self,
                bank_name = None,
                branch_number = None,
                account_number = None,
                name_reference = None )  [private]
```

check if this client has credentials, if not register

Definition at line 330 of file core_test.py.

```
330     def __register_bank_account(self, bank_name=None, branch_number=None, account_number=None,
        name_reference=None):
331         """ check if this client has credentials, if not register
332 
333         """
334         #check if user have credentials
335         BRANCH_NUM_MAX=100
336         ACCOUNT_NUMBER_MAX=10000000000
337         self.db.init()
338         try:
339             #register user
340             bank_name = self.faker.name() if bank_name==None else bank_name
341             branch_number = random.random()*BRANCH_NUM_MAX if branch_number==None else branch_number
342             account_number= random.random()*ACCOUNT_NUMBER_MAX if account_number==None else
        account_number
343             name_reference= self.faker.name() if name_reference==None else anem_reference
344             payload={'bank_name':bank_name,
345                      'branch_number':branch_number,
346                      'account_number':account_number,
347                      'name_reference':name_reference}
348             res=requests.post(ADD_BANK_ACCOUNT_URL, \
349                               data=json.dumps(payload))
350             response = json.loads(res.text)
351             scode=res.status_code
352             #TODO support multiple accounts
353             #create local client table for banking,
354             #to insert balance, or each account.
355             self.balance=response['balance']
356             self.db.commit()
357 
358         except psycopg2.DatabaseError as error:
359             logger.critical("bank account registration failed!, error: "+ str(error))
360             print("bank account register failed!, error: "+str(error))
361             self.db.rollback()
362         finally:
363             self.db.close()
```

References core_test.Client.db, core.client.client.Client.db, client.client.Client.db, core_test.Client.faker, client.↩client.Client.faker, core.client.client.Client.faker, and setup.name.

7.16.2.7 __transact()

```
def core_test.RestfulTest.__transact (
                self,
                credid,
                amount,
                currency = 'USD' )  [private]
```

Definition at line 386 of file core_test.py.

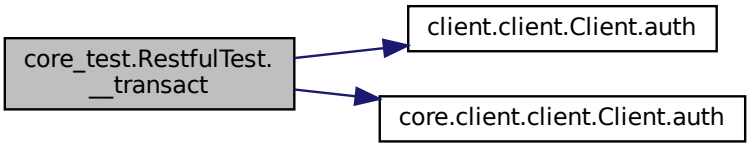
```
386     def __transact(self, credid, amount, currency='USD'):
387         pur_item = {'credid': credid,
388                     'amount':amount,
389                     'currency': 'USD'}
390         ret=requests.post(PURCHASE_URL, data=json.dumps(pur_item), auth=self.auth())
391         response=ret.text#ret.text.decode('utf8')#.replace('"', "'")
```

```
392         trx=json.loads(response)
393         return trx
```

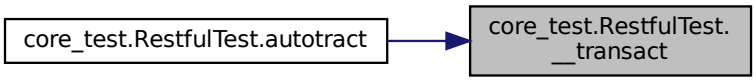
References client.client.Client.auth(), and core.client.client.Client.auth().

Referenced by core_test.RestfulTest.autotract().

Here is the call graph for this function:



Here is the caller graph for this function:



7.16.2.8 __update_balance()

```
def core_test.RestfulTest.__update_balance (
              self,
              balance )  [private]
```

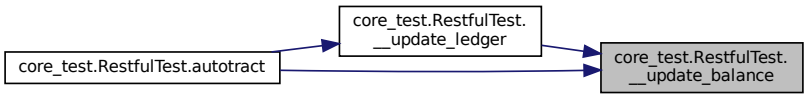
Definition at line 406 of file core_test.py.

```
406     def __update_balance(self, balance):
407         #TODO fix
408         if type(balance)==dict:
409             self.balance=balance['balance']
410             return
411         self.balance=balance
```

References core_test.Client.balance, client.client.Client.balance, core.client.client.Client.balance, core_test.↩ RestfulTest.balance, and client.client.type.

Referenced by core_test.RestfulTest.__update_ledger(), and core_test.RestfulTest.autotract().

Here is the caller graph for this function:



7.16.2.9 __update_ledger()

```
def core_test.RestfulTest.__update_ledger (
              self )  [private]
```

update ledger with sold goods, and update balance

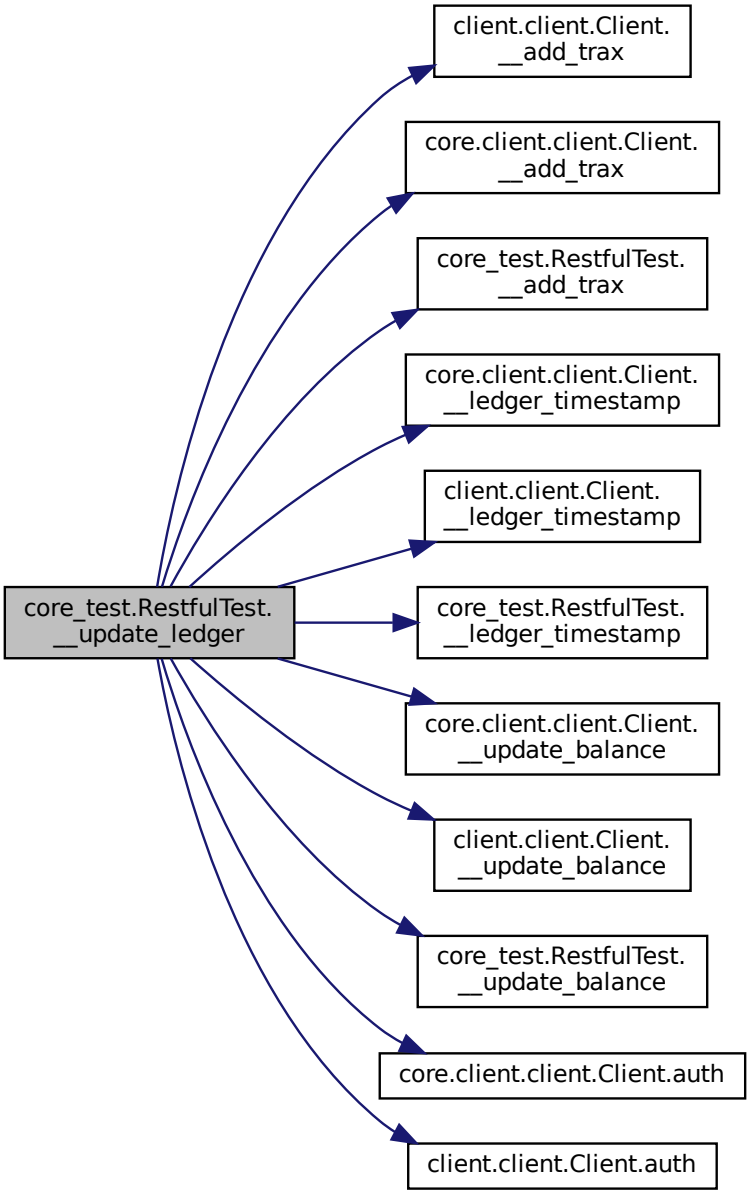
Definition at line 425 of file core_test.py.

```
425     def __update_ledger(self):
426         """update ledger with sold goods, and update balance
427         """
428         ret=requests.post(LEDGER_URL, data=json.dumps({'trx_dt': self.__ledger_timestamp()}),
      auth=self.auth())
429         #TODO always process the status code!
430         res = json.loads(ret.text) #self.__get_dict(ret.text)
431         print("new balance: ", res['balance'])
432         self.__update_balance(res['balance'])
433         self.db.init();
434         try:
435             new_trxs=res['transactions']
436             print("trxs: ", new_trxs)
437             transactions=pd.read_json(new_trxs)
438             print("trxs pandas: ", transactions)
439             if len(transactions)==0:
440                 raise Exception("empty transaction!")
441             for i in range(len(transactions)-1):
442                 self.__add_trax(transactions.iloc[i])
443         except psycopg2.DatabaseError as error:
444             self.db.rollback()
445         except:
446             self.db.rollback()
447         finally:
448             self.db.close()
```

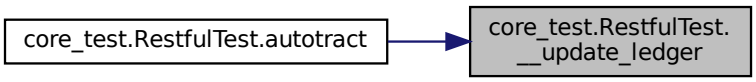
References client.client.Client.__add_trax(), core.client.client.Client.__add_trax(), core_test.RestfulTest.__add_↩trax(), core.client.client.Client.__ledger_timestamp(), client.client.Client.__ledger_timestamp(), core_test.Restful↩Test.__ledger_timestamp(), client.client.Client.__update_balance(), core.client.client.Client.__update_balance(), core_test.RestfulTest.__update_balance(), client.client.Client.auth(), core.client.client.Client.auth(), core_test.↩Client.db, core.client.client.Client.db, and client.client.Client.db.

Referenced by core_test.RestfulTest.autotract().

Here is the call graph for this function:



Here is the caller graph for this function:



7.16.2.10 add_contact()

```
def core_test.RestfulTest.add_contact (
                self,
                email,
                name )
```

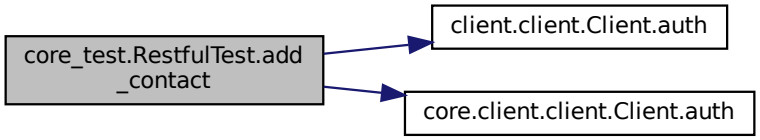
```
Fetch the client with given email/name

@param email: contact email
@param name: contact name
```

Definition at line 364 of file core_test.py.

```
364     def add_contact(self, email, name):
365         """Fetch the client with given email/name
366 
367         @param email: contact email
368         @param name: contact name
369         """
370         #get client credential id
371         cred={'email': email}
372         ret=requests.post(CONTACTS_URL, \
373                           data=json.dumps(cred), \
374                           auth=self.auth())
375         respose=ret.text#.decode('utf8')#.replace('"', "'")
376         res = json.laods(response)
377         credid = res['credid']
378         self.db.init()
379         try:
380             self.db.inserts.insert_contact(email, name, credid)
381             self.db.commit()
382         except psycopg2.DatabaseError as error:
383             self.db.rollback()
384         finally:
385             self.db.close()
```

References client.client.Client.auth(), core.client.client.Client.auth(), core_test.Client.db, client.client.Client.db, and core.client.client.Client.db.

Here is the call graph for this function:



7.16.2.11 autotract()

```
def core_test.RestfulTest.autotract (
              self )
```

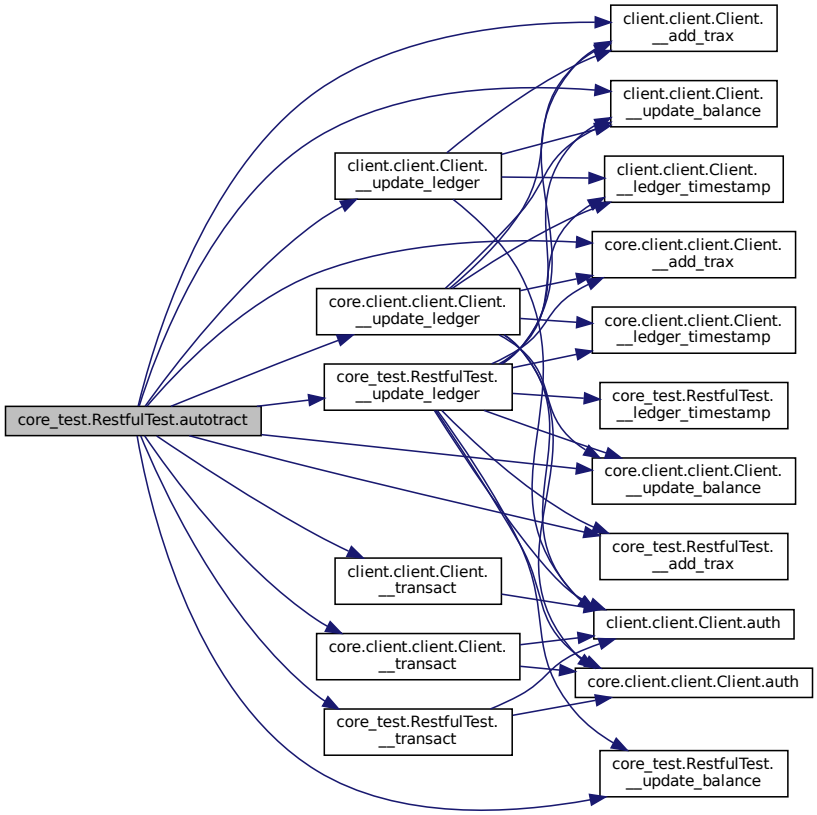
```
Stochastic fake auto-tracting

trade with randomly with maintained contacts with 0.5 probability for each, and 0.1 probability for all contac
```

Definition at line 449 of file core_test.py.

```
449     def autotract(self):
450         """Stochastic fake auto-tracting
451 
452         trade with randomly with maintained contacts with 0.5 probability for each, and 0.1 probability
      for all contacts goods, update balance, and goods for each contact
453         """
454         #update balance, and add new transactions
455         MAX_TRACT_BALANCE=100000
456         self.__update_ledger()
457         contacts_df=db.gets.get_all_contacts()
458         for i in range(len(contacts_df-1)):
459             contact_credid=contacts_df.iloc[i]['contact_id']
460             contact_name=contacts_df.iloc[i]['contact_name']
461             contact_email=contacts_df.iloc[i]['contact_email']
462             amount=random.random()*MAX_TRACT_BALANCE
463             logger.info("making transaction with {}[{}] by amount: {}".\
464                         format(contact_name, contact_email, amount))
465             #if random.random()> STOCHASTIC_TRADE_THRESHOLD and good.cost <= self.balance:
466             #TODO (fix) doesn't work!
467             #trx=self.__purchase(good.gid)
468             trx=self.__transact(contact_credid, amount)
469             self.__add_trax(trx)
470             self.__update_balance(trx)
471 
```

References client.client.Client.__add_trax(), core.client.client.Client.__add_trax(), core_test.RestfulTest.__add↩
_trax(), client.client.Client.__transact(), core.client.client.Client.__transact(), core_test.RestfulTest.__transact(), client.client.Client.__update_balance(), core.client.client.Client.__update_balance(), core_test.RestfulTest.__↩
update_balance(), client.client.Client.__update_ledger(), core.client.client.Client.__update_ledger(), core_test.↩
RestfulTest.__update_ledger(), and core_test.format.

Here is the call graph for this function:



7.16.2.12 setUp()

```
def core_test.RestfulTest.setUp (
              self )
```

Definition at line 171 of file core_test.py.

```
171     def setUp(self):
172         self.app=app.test_client()
173         self.uname=None
174         self.pas=None
175         logger.info("client initialized")
```

7.16.2.13 test_add_contact()

```
def core_test.RestfulTest.test_add_contact (
              self )
```

Definition at line 213 of file core_test.py.

```
213     def test_add_contact(self):
```

```
214         src = Client(self.app)
215         dest = Client(self.app)
216         credid=src.add_contact(dest.email, dest.name)
217         self.assertEqual(credid, dest.credid)
```

References core_test.Client.app, and core_test.RestfulTest.app.

7.16.2.14 test_get_balance()

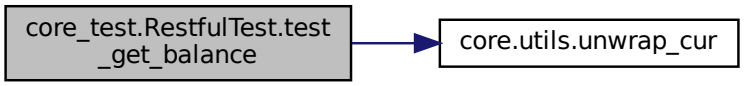
```
def core_test.RestfulTest.test_get_balance (
                self )
```

Definition at line 206 of file core_test.py.

```
206     def test_get_balance(self):
207         src = Client(self.app)
208         balance=src.get_balance()
209         print('|------------>balance: ', balance)
210         self.assertEqual(balance[0], src.balance)
211         self.assertEqual(unwrap_cur(balance[1]), unwrap_cur(src.currency_pref))
212
```

References core_test.Client.app, core_test.RestfulTest.app, and core.utils.unwrap_cur().

Here is the call graph for this function:



7.16.2.15 test_make_transaction()

```
def core_test.RestfulTest.test_make_transaction (
                self )
```

Definition at line 220 of file core_test.py.

```
220     def test_make_transaction(self):
221         src = Client(self.app)
222         dest = Client(self.app)
223         amount=get_amount()
224         old_balance=src.balance
225         new_balance, trxs = src.make_transaction(dest.email, dest.name, amount)
226         print("transaction made: ", trxs)
227         self.assertEqual(new_balance, old_balance-(amount+FEE))
228
```

7.16.2.16 test_register()

```
def core_test.RestfulTest.test_register (
              self )
```

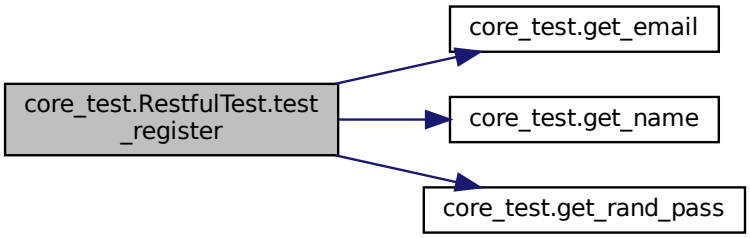
check if this client has credentials, if not register

Definition at line 176 of file core_test.py.

```
176     def test_register(self):
177         """ check if this client has credentials, if not register
178 
179         """
180         name = get_name()
181         email = get_email()
182         passcode = get_rand_pass()
183         payload={'name':name,\
184                  'email': email,\
185                  'passcode': passcode}
186         res=self.app.post(REGISTER, data=json.dumps(payload))
187         response = res.json #json.loads(res.json)
188         credid=response['cred_id']
189         #cid set to 0, since it never matters in the client side
190         #self.db.inserts.register(0, passcode, credid)
191         logger.debug("user registered with credentials username: {}, email: {}, passcode:
        {}".format(name, email, passcode))
192         self.assertTrue(type(credid)==int)
193         self.assertEqual(res.status_code, 201)
194         self.uname=email
195         self.pas=passcode
```

References core_test.Client.app, core_test.RestfulTest.app, core_test.format, core_test.get_email(), core_test.get_name(), core_test.get_rand_pass(), client.client.Client.pas, core.client.client.Client.pas, core_test.RestfulTest.pas, client.client.type, client.client.Client.uname, core.client.client.Client.uname, and core_test.RestfulTest.uname.

Here is the call graph for this function:



7.16.2.17 test_udapte_ledger()

```
def core_test.RestfulTest.test_udapte_ledger (
              self )
```

Definition at line 218 of file core_test.py.

```
218     def test_udapte_ledger(self):
219         pass
```

7.16.3 Field Documentation

7.16.3.1 app

`core_test.RestfulTest.app`

Definition at line 172 of file core_test.py.

Referenced by core_test.RestfulTest.test_add_contact(), core_test.RestfulTest.test_get_balance(), and core_test.RestfulTest.test_register().

7.16.3.2 balance

`core_test.RestfulTest.balance`

Definition at line 355 of file core_test.py.

Referenced by core_test.RestfulTest.__update_balance(), core.utils.PaymentGate.authenticated(), and core.utils.PaymentGate.get_balance().

7.16.3.3 pas

`core_test.RestfulTest.pas`

Definition at line 174 of file core_test.py.

Referenced by core_test.RestfulTest.__auth(), and core_test.RestfulTest.test_register().

7.16.3.4 uname

`core_test.RestfulTest.uname`

Definition at line 173 of file core_test.py.

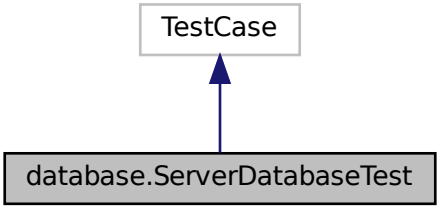
Referenced by core_test.RestfulTest.__auth(), and core_test.RestfulTest.test_register().

The documentation for this class was generated from the following file:

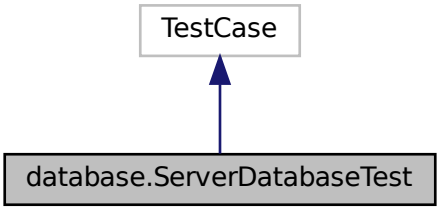
- core/tests/core_test.py

7.17 database.ServerDatabaseTest Class Reference

Inheritance diagram for database.ServerDatabaseTest:



Collaboration diagram for database.ServerDatabaseTest:



Public Member Functions

- def test_currency (self)
- def test_banking_byemail (self)
- def test_banking_byname (self)
- def test_bank_account_exist_by_cid (self)
- def test_client_exists (self)
- def test_credential_exists (self)
- def test_add_bank_account (self)
- def test_bid_cid_conversion (self)
- def test_balance (self)
- def test_update_balance (self)
- def test_credid2cid (self)
- def test_password (self)
- def test_transaction (self)

7.17.1 Detailed Description

Definition at line 53 of file database.py.

7.17.2 Member Function Documentation

7.17.2.1 test_add_bank_account()

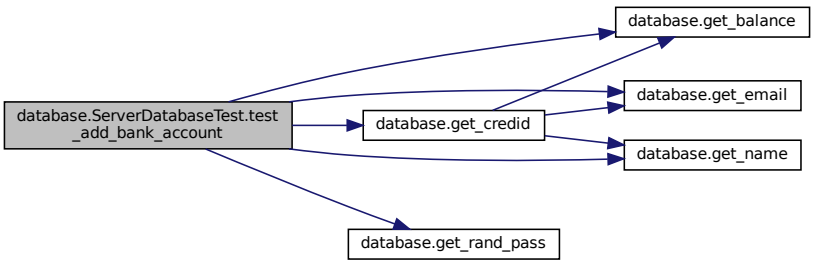
```
def database.ServerDatabaseTest.test_add_bank_account (
            self )
```

Definition at line 139 of file database.py.

```
139     def test_add_bank_account(self):
140         exchange=Currency(EUR)
141         rate=exchange.rate
142         if not db.exists.currency(EUR):
143             db.inserts.add_currency(EUR, rate)
144         curr_id=db.gets.get_currency_id(EUR)
145         passcode=get_rand_pass()
146         email=get_email()
147         credid=get_credid()
148         banalce=get_balance()
149         email=get_email()
150         name=get_name()
151         db.inserts.add_client(name, email, curr_id)
152         cid=db.gets.get_client_id_byemail(email)
153         db.inserts.register(cid, passcode, credid)
154         #add_bank_addount
155         bid=db.inserts.add_bank_account(cid, balance, bank_name, branch_number, account_number,
    name_reference, curr_id)
156         self.assertTrue(db.exists.bank_account_bycid(cid))
```

References database.get_balance(), database.get_credid(), database.get_email(), database.get_name(), and database.get_rand_pass().

Here is the call graph for this function:



7.17.2.2 test_balance()

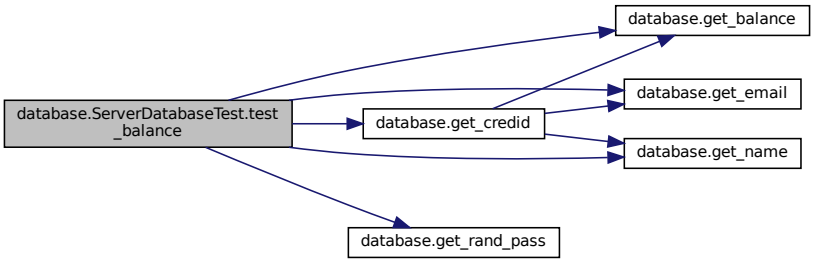
```
def database.ServerDatabaseTest.test_balance (
              self )
```

Definition at line 183 of file database.py.

```
183     def test_balance(self):
184         exchange=Currency(EUR)
185         rate=exchange.rate
186         if not db.exists.currency(EUR):
187             db.inserts.add_currency(EUR, rate)
188         curr_id=db.gets.get_currency_id(EUR)
189         passcode=get_rand_pass()
190         email=get_email()
191         credid=get_credid()
192         banalce=get_balance()
193         email=get_email()
194         name=get_name()
195         db.inserts.add_client(name, email, curr_id)
196         cid=db.gets.get_client_id_byemail(email)
197         db.inserts.register(cid, passcode, credid)
198         #add_bank_addount
199         bid=db.inserts.add_bank_account(cid, balance, bank_name, branch_number, account_number,
    name_reference, curr_id)
200         bank_exists=db.exists.bank_account_bycid(cid)
201         self.assertTrue(bank_exists)
202         print('cid: ', cid)
203         balance_cur=db.gets.get_balance_by_cid(cid)['balance']
204         self.assertEqual(balance_cur, balance)
```

References database.get_balance(), database.get_credid(), database.get_email(), database.get_name(), and database.get_rand_pass().

Here is the call graph for this function:



7.17.2.3 test_bank_account_exist_by_cid()

```
def database.ServerDatabaseTest.test_bank_account_exist_by_cid (
              self )
```

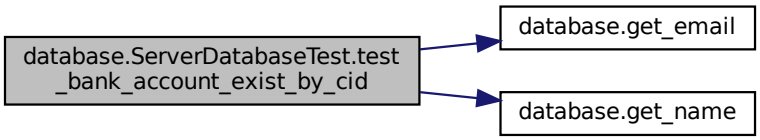
Definition at line 96 of file database.py.

```
96     def test_bank_account_exist_by_cid(self):
97         exchange=Currency(EUR)
98         rate=exchange.rate
99         if not db.exists.currency(EUR):
100             db.inserts.add_currency(EUR, rate)
101         curr_id=db.gets.get_currency_id(EUR)
102         name=get_name()
103         email=get_email()
104         db.inserts.add_client(name, email, curr_id)
```

```
105         cid=db.gets.get_client_id_byemail(email)
106         self.assertFalse(db.exists.bank_account_bycid(cid))
107
```

References database.get_email(), and database.get_name().

Here is the call graph for this function:



7.17.2.4 test_banking_byemail()

```
def database.ServerDatabaseTest.test_banking_byemail (
              self )
```

Definition at line 65 of file database.py.

```
65     def test_banking_byemail(self):
66         #db.init()
67         #db.repeatable_read()
68         #db.lock_advisory(lock)
69         exchange=Currency(EUR)
70         rate=exchange.rate
71         if not db.exists.currency(EUR):
72             db.inserts.add_currency(EUR, rate)
73         curr_id=db.gets.get_currency_id(EUR)
74         db.inserts.add_client(name, email, curr_id)
75         self.assertTrue(db.exists.account_byemail(email))
76         cid=db.gets.get_client_id_byemail(email)
77         self.assertTrue(db.exists.client_exists(cid))
78         #db.rollback(lock)
79
```

7.17.2.5 test_banking_byname()

```
def database.ServerDatabaseTest.test_banking_byname (
              self )
```

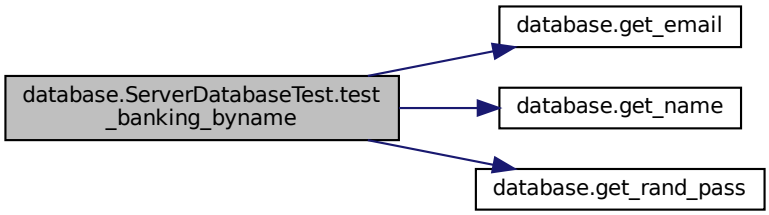
Definition at line 80 of file database.py.

```
80     def test_banking_byname(self):
81         exchange=Currency(EUR)
82         rate=exchange.rate
83         if not db.exists.currency(EUR):
84             db.inserts.add_currency(EUR, rate)
85         curr_id=db.gets.get_currency_id(EUR)
86         passcode=get_rand_pass()
87         email=get_email()
88         name=get_name()
89         db.inserts.add_client(name, email, curr_id)
90         cid=db.gets.get_client_id_byemail(email)
91         db.inserts.register(cid, passcode, credid)
```

```
92         self.assertTrue(db.exists.account_byname(name, passcode))
93         cid=db.gets.get_client_id_byname(name, passcode)
94         self.assertTrue(db.exists.client_exists(cid))
95
```

References database.get_email(), database.get_name(), and database.get_rand_pass().

Here is the call graph for this function:



7.17.2.6 test_bid_cid_conversion()

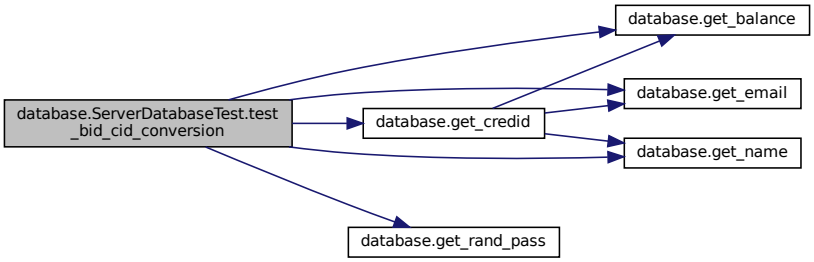
```
def database.ServerDatabaseTest.test_bid_cid_conversion (
              self )


bid_cid conversion testing

convert, and cross-reference from client id, to bank id
```

Definition at line 157 of file database.py.

```
157     def test_bid_cid_conversion(self):
158         """ bid_cid conversion testing
159
160         convert, and cross-reference from client id, to bank id
161         """
162         exchange=Currency(EUR)
163         rate=exchange.rate
164         if not db.exists.currency(EUR):
165             db.inserts.add_currency(EUR, rate)
166         curr_id=db.gets.get_currency_id(EUR)
167         passcode=get_rand_pass()
168         email=get_email()
169         credid=get_credid()
170         banalce=get_balance()
171         email=get_email()
172         name=get_name()
173         db.inserts.add_client(name, email, curr_id)
174         cid=db.gets.get_client_id_byemail(email)
175         db.inserts.register(cid, passcode, credid)
176         #add_bank_addount
177         bid=db.inserts.add_bank_account(cid, balance, bank_name, branch_number, account_number,
      name_reference, curr_id)
178         cid_eq=db.gets.get_client_id(bid)
179         bid_eq=db.gets.get_banking_id(cid_eq)
180         self.assertEqual(cid_eq, cid)
181         self.assertEqual(bid_eq, bid)
182
```

References database.get_balance(), database.get_credid(), database.get_email(), database.get_name(), and database.get_rand_pass().

Here is the call graph for this function:



7.17.2.7 test_client_exists()

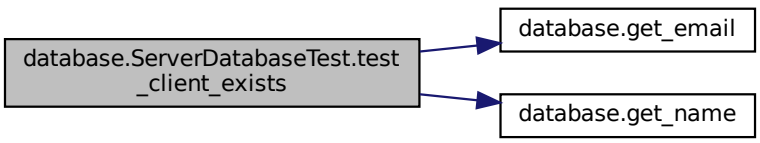
```
def database.ServerDatabaseTest.test_client_exists (
              self )
```

Definition at line 108 of file database.py.

```
108     def test_client_exists(self):
109         exchange=Currency(EUR)
110         rate=exchange.rate
111         if not db.exists.currency(EUR):
112             db.inserts.add_currency(EUR, rate)
113         curr_id=db.gets.get_currency_id(EUR)
114         name=get_name()
115         email=get_email()
116         db.inserts.add_client(name, email, curr_id)
117         cid=db.gets.get_client_id_byemail(email)
118         self.assertTrue(db.exists.client_exists(cid))
```

References database.get_email(), and database.get_name().

Here is the call graph for this function:



7.17.2.8 test_credential_exists()

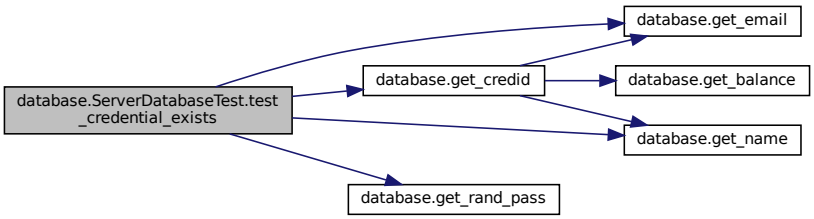
```
def database.ServerDatabaseTest.test_credential_exists (
                self )
```

Definition at line 119 of file database.py.

```
119     def test_credential_exists(self):
120         exchange=Currency(EUR)
121         rate=exchange.rate
122         if not db.exists.currency(EUR):
123             db.inserts.add_currency(EUR, rate)
124         curr_id=db.gets.get_currency_id(EUR)
125         name=get_name()
126         email=get_email()
127         passcode=get_rand_pass()
128         credid=get_credid()
129         db.inserts.add_client(name, email, curr_id)
130         db.commit()
131         cid=db.gets.get_client_id_byemail(email)
132         db.commit()
133         self.assertFalse(db.exists.credential_exists(0))
134         db.inserts.register(cid, passcode, credid)
135         cid=db.gets.get_client_id_byname(name, passcode)
136         db.commit()
137         self.assertTrue(db.exists.credential_exists(cid))
138
```

References database.get_credid(), database.get_email(), database.get_name(), and database.get_rand_pass().

Here is the call graph for this function:



7.17.2.9 test_credid2cid()

```
def database.ServerDatabaseTest.test_credid2cid (
                self )
```

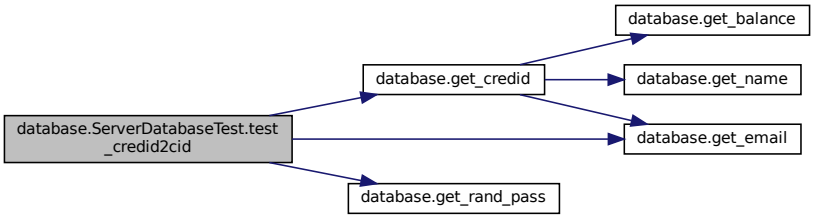
Definition at line 229 of file database.py.

```
229     def test_credid2cid(self):
230         exchange=Currency(EUR)
231         rate=exchange.rate
232         if not db.exists.currency(EUR):
233             db.inserts.add_currency(EUR, rate)
234         curr_id=db.gets.get_currency_id(EUR)
235         passcode=get_rand_pass()
236         email=get_email()
237         credid=get_credid()
238         # add new client
239         db.inserts.add_client(name, email, curr_id)
240         cid=db.gets.get_client_id_byemail(email)
241         # register client's credentials
242         db.inserts.register(cid, passcode, credid)
243         # credid2cid conversion
244         cid_eq=db.gets.credid2cid(credid)
```

```
245         self.assertEqual(cid, cid_eq)
246         credid_eq=db.gets.cid2credid(cid)
247         self.assertEqual(credid, credid_eq)
248
```

References database.get_credid(), database.get_email(), and database.get_rand_pass().

Here is the call graph for this function:



7.17.2.10 test_currency()

```
def database.ServerDatabaseTest.test_currency (
              self )
```

Definition at line 54 of file database.py.

```
54     def test_currency(self):
55         #db.init()
56         #db.repeatable_read()
57         #db.lock_advisory(lock)
58         exchange=Currency(EUR)
59         rate=exchange.rate
60         if not db.exists.currency(EUR):
61             db.inserts.add_currency(EUR, rate)
62         self.assertEqual(rate, 1)
63         #db.rollback(lock)
64
```

7.17.2.11 test_password()

```
def database.ServerDatabaseTest.test_password (
              self )
```

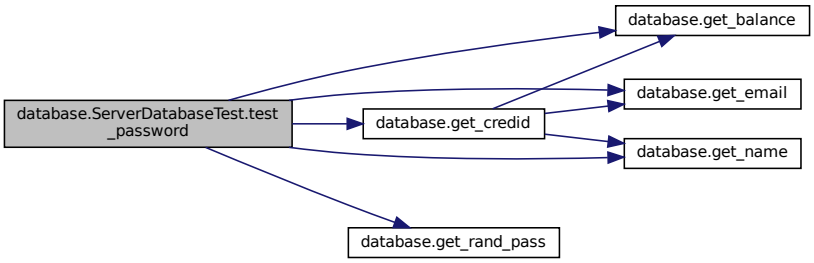
Definition at line 249 of file database.py.

```
249     def test_password(self):
250         exchange=Currency(EUR)
251         rate=exchange.rate
252         if not db.exists.currency(EUR):
253             db.inserts.add_currency(EUR, rate)
254         curr_id=db.gets.get_currency_id(EUR)
255         passcode=get_rand_pass()
256         email=get_email()
257         credid=get_credid()
258         banalce=get_balance()
259         email=get_email()
260         name=get_name()
261         db.inserts.add_client(name, email, curr_id)
262         cid=db.gets.get_client_id_byemail(email)
```

```
263         db.inserts.register(cid, passcode, credid)
264         #add_bank_addount
265         bid=db.inserts.add_bank_account(cid, balance, bank_name, branch_number, account_number,
      name_reference, curr_id)
266         passcode_eq=db.gets.get_password(credid)
267         self.assertEqual(passcode, passcode_eq)
268
```

References database.get_balance(), database.get_credid(), database.get_email(), database.get_name(), and database.get_rand_pass().

Here is the call graph for this function:



7.17.2.12 test_transaction()

```
def database.ServerDatabaseTest.test_transaction (
              self )
```

```
create two clients, client_1, client_2

procedure:
- client_1 sends 10k to client_2
- client_1 sends 20k to client_2
- client_2 sends 5k to client_1
- transaction sum 35k sent/received
```

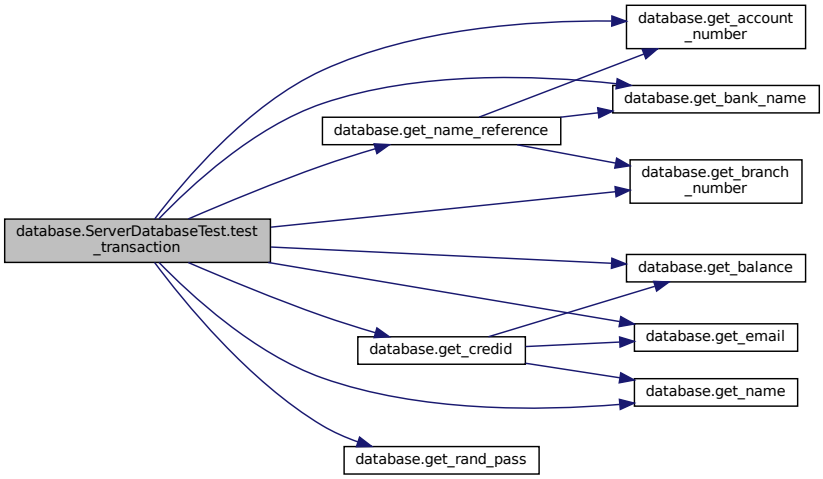
Definition at line 269 of file database.py.

```
269     def test_transaction(self):
270         """ create two clients, client_1, client_2
271
272         procedure:
273         - client_1 sends 10k to client_2
274         - client_1 sends 20k to client_2
275         - client_2 sends 5k to client_1
276         - transaction sum 35k sent/received
277         """
278         exchange=Currency(EUR)
279         rate=exchange.rate
280         if not db.exists.currency(EUR):
281             db.inserts.add_currency(EUR, rate)
282         curr_id=db.gets.get_currency_id(EUR)
283         c1_name=get_name()
284         c1_email=get_email()
285         c1_passcode=get_rand_pass()
286         c1_credid=get_credid()
287         c1_bank_name=get_bank_name()
288         c1_branch_number=get_branch_number()
289         c1_account_number=get_account_number()
290         c1_name_reference=get_name_reference()
```

```
291         c1_balance=get_balance()
292         db.inserts.add_client(c1_name, c1_email, curr_id)
293         db.commit()
294         c1_cid=db.gets.get_client_id_byemail(c1_email)
295         db.inserts.register(c1_cid, c1_passcode, c1_credid)
296         db.commit()
297         db.inserts.add_bank_account(c1_cid, c1_balance, c1_bank_name, c1_branch_number,
    c1_account_number, c1_name_reference, curr_id)
298         db.commit()
299         #
300         c2_name=get_name()
301         c2_email=get_email()
302         c2_passcode=get_rand_pass()
303         c2_credid=get_credid()
304         c2_bank_name=get_bank_name()
305         c2_branch_number=get_branch_number()
306         c2_account_number=get_account_number()
307         c2_name_reference=get_name_reference()
308         c2_balance=get_balance()
309         db.inserts.add_client(c2_name, c2_email, curr_id)
310         db.commit()
311         c2_cid=db.gets.get_client_id_byemail(c2_email)
312         db.inserts.register(c2_cid, c2_passcode, c2_credid)
313         db.commit()
314         db.inserts.add_bank_account(c2_cid, c2_balance, c2_bank_name, c2_branch_number,
    c2_account_number, c2_name_reference, curr_id)
315         db.commit()
316         #
317
320         costs=[10000, 20000, 5000]
321         trx_st_0=datetime.datetime.now().strftime(TIMESTAMP_FORMAT)
322         db.inserts.insert_trx(c2_credid, c1_credid, costs[0], curr_id, 'transaction1')
323         db.commit()
324         trx_st_1=datetime.datetime.now().strftime(TIMESTAMP_FORMAT)
325         db.inserts.insert_trx(c2_credid, c1_credid, costs[1], curr_id, 'transaction2')
326         db.commit()
327         trx_st_2=datetime.datetime.now().strftime(TIMESTAMP_FORMAT)
328         db.inserts.insert_trx(c1_credid, c2_credid, costs[2], curr_id, 'transaction3')
329         db.commit()
330         trx_st_3=datetime.datetime.now().strftime(TIMESTAMP_FORMAT)
331
335         c1_trx_sum_0=db.gets.get_transactions_sum(c1_credid, trx_st_0)
336         db.commit()
337         self.assertEqual(sum(costs), c1_trx_sum_0)
338         #epoch 2
339         c1_trx_sum_1=db.gets.get_transactions_sum(c1_credid, trx_st_1)
340         db.commit()
341         self.assertEqual(sum(costs[1:]), c1_trx_sum_1)
342         #epoch 3
343         c1_trx_sum_2=db.gets.get_transactions_sum(c1_credid, trx_st_2)
344         db.commit()
345         self.assertEqual(sum(costs[2:]), c1_trx_sum_2)
346
```

References database.get_account_number(), database.get_balance(), database.get_bank_name(), database.↩ get_branch_number(), database.get_credid(), database.get_email(), database.get_name(), database.get_name↩ _reference(), and database.get_rand_pass().

Here is the call graph for this function:



7.17.2.13 test_update_balance()

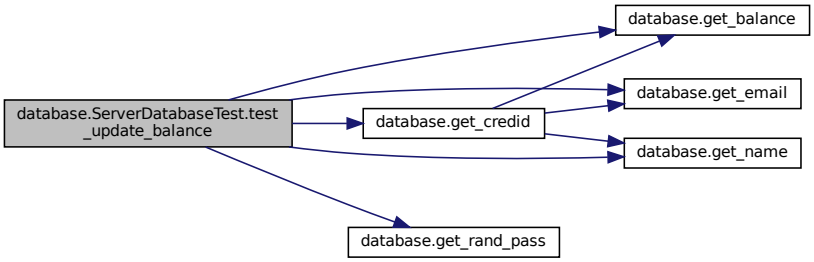
```
def database.ServerDatabaseTest.test_update_balance (
              self )
```

Definition at line 205 of file database.py.

```
205     def test_update_balance(self):
206         exchange=Currency(EUR)
207         rate=exchange.rate
208         if not db.exists.currency(EUR):
209             db.inserts.add_currency(EUR, rate)
210         curr_id=db.gets.get_currency_id(EUR)
211         passcode=get_rand_pass()
212         email=get_email()
213         credid=get_credid()
214         balance=get_balance()
215         email=get_email()
216         name=get_name()
217         db.inserts.add_client(name, email, curr_id)
218         db.commit()
219         cid=db.gets.get_client_id_byemail(email)
220         db.inserts.register(cid, passcode, credid)
221         db.commit()
222         #add_bank_addount
223         bid=db.inserts.add_bank_account(cid, balance, bank_name, branch_number, account_number,
      name_reference, curr_id)
224         db.commit()
225         db.updates.update_account(cid, 0)
226         balance_cur=db.gets.get_balance_by_cid(cid)['balance']
227         self.assertEqual(balance_cur, 0)
```

References database.get_balance(), database.get_credid(), database.get_email(), database.get_name(), and database.get_rand_pass().

Here is the call graph for this function:

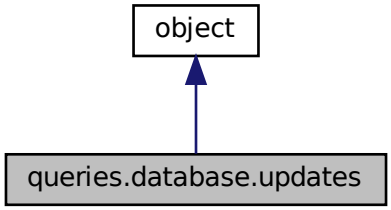


The documentation for this class was generated from the following file:

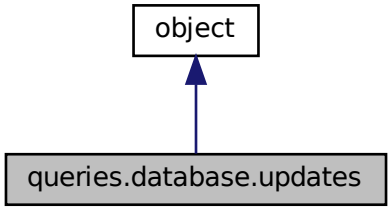
- core/tests/database.py

7.18 queries.database.updates Class Reference

Inheritance diagram for queries.database.updates:



Collaboration diagram for queries.database.updates:



Public Member Functions

- def __init__ (self, conn, cur, logger)
- def update_account (self, cid, balance)

Data Fields

- conn
- cur
- db_log

7.18.1 Detailed Description

Definition at line 584 of file database.py.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 __init__()

```
def queries.database.updates.__init__ (
              self,
              conn,
              cur,
              logger )
```

Definition at line 585 of file database.py.

```
585     def __init__(self, conn, cur, logger):
586         self.conn=conn
587         self.cur=cur
588         self.db_log=logger
```

7.18.3 Member Function Documentation

7.18.3.1 update_account()

```
def queries.database.updates.update_account (
              self,
              cid,
              balance )
```

```
update the banking account with the calculated new balance (CALLED FROM SERVER SIDE)

@param cid: client id
@param balance: the account balance
```

Definition at line 589 of file database.py.

```
589     def update_account(self, cid, balance):
590         """update the banking account with the calculated new balance (CALLED FROM SERVER SIDE)
591 
592         @param cid: client id
593         @param balance: the account balance
594         """
595         stat = sql.SQL("UPDATE banking SET (balance, balance_dt) = ({balance}, {dt}) WHERE
        client_id={cid}").\
596             format(balance=sql.Literal(balance), \
597                    dt=sql.Literal(dt.datetime.now().strftime(TIMESTAMP_FORMAT)), \
598                    cid=sql.Literal(cid))
599         self.cur.execute(stat)
```

References queries.database.exists.cur, queries.database.gets.cur, queries.database.inserts.cur, queries.↩database.updates.cur, and client.client.format.

7.18.4 Field Documentation

7.18.4.1 conn

```
queries.database.updates.conn
```

Definition at line 586 of file database.py.

Referenced by core.queries.database.exists.__init__(), core.queries.database.gets.__init__(), core.queries.↩database.inserts.__init__(), core.queries.database.updates.__init__(), core.queries.database.database.__init__(), queries.database.database.close(), core.queries.database.database.close(), queries.database.database.commit(), core.queries.database.database.commit(), core.queries.database.gets.get_all_clients(), core.queries.database.↩gets.get_all_contacts(), core.queries.database.gets.get_all_credentials(), core.queries.database.gets.get_↩credential(), core.queries.database.gets.get_sells(), core.queries.database.gets.get_transactions(), core.queries.↩database.gets.get_transactions_sum(), queries.database.database.init(), core.queries.database.database.init(), queries.database.database.rollback(), and core.queries.database.database.rollback().

7.18.4.2 cur

queries.database.updates.cur

Definition at line 587 of file database.py.

Referenced by core.queries.database.exists.__init__(), core.queries.database.gets.__init__(), core.queries.database.inserts.__init__(), core.queries.database.updates.__init__(), core.queries.database.database.__init__(), core.queries.database.exists.account_byemail(), core.queries.database.exists.account_byname(), core.queries.database.inserts.add_bank_account(), core.queries.database.inserts.add_client(), core.queries.database.inserts.add_currency(), core.queries.database.exists.bank_account_bycid(), core.queries.database.gets.cid2credid(), core.queries.database.exists.client_exists(), queries.database.database.committed_read(), core.queries.database.database.committed_read(), core.queries.database.exists.contact_exists(), core.queries.database.exists.credential_exists(), core.queries.database.gets.credid2cid(), core.queries.database.exists.currency(), core.queries.database.updates.currency_preference(), core.queries.database.gets.get(), core.queries.database.gets.get_balance_by_cid(), core.queries.database.gets.get_balance_by_credid(), core.queries.database.gets.get_banking_id(), core.queries.database.gets.get_client_id(), core.queries.database.gets.get_client_id_byemail(), core.queries.database.gets.get_client_id_byname(), core.queries.database.gets.get_currency_id(), core.queries.database.gets.get_currency_name(), core.queries.database.gets.get_last_timestamp(), core.queries.database.gets.get_password(), core.queries.database.gets.get_preference_currency_bycid(), queries.database.database.init(), core.queries.database.database.init(), core.queries.database.inserts.insert_contact(), core.queries.database.inserts.insert_trx(), queries.database.database.lock_advisory(), core.queries.database.database.lock_advisory(), core.queries.database.inserts.register(), queries.database.database.repeatable_read(), core.queries.database.database.repeatable_read(), core.queries.database.gets.to_euro(), queries.database.database.unlock_advisory(), core.queries.database.database.unlock_advisory(), queries.database.updates.update_account(), and core.queries.database.updates.update_account().

7.18.4.3 db_log

queries.database.updates.db_log

Definition at line 588 of file database.py.

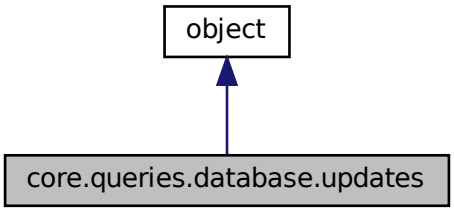
Referenced by core.queries.database.gets.__init__(), core.queries.database.inserts.__init__(), core.queries.database.updates.__init__(), core.queries.database.inserts.add_bank_account(), core.queries.database.gets.cid2credid(), core.queries.database.gets.credid2cid(), core.queries.database.gets.get(), core.queries.database.gets.get_all_clients(), core.queries.database.gets.get_all_contacts(), core.queries.database.gets.get_all_credentials(), core.queries.database.gets.get_balance_by_cid(), core.queries.database.gets.get_balance_by_credid(), core.queries.database.gets.get_banking_id(), core.queries.database.gets.get_client_id(), core.queries.database.gets.get_client_id_byemail(), core.queries.database.gets.get_client_id_byname(), core.queries.database.gets.get_credential(), core.queries.database.gets.get_currency_id(), core.queries.database.gets.get_currency_name(), core.queries.database.gets.get_last_timestamp(), core.queries.database.gets.get_password(), core.queries.database.gets.get_preference_currency_bycid(), core.queries.database.gets.get_sells(), core.queries.database.gets.get_transactions(), core.queries.database.gets.get_transactions_sum(), core.queries.database.inserts.insert_contact(), core.queries.database.inserts.insert_trx(), core.queries.database.inserts.register(), and core.queries.database.gets.to_euro().

The documentation for this class was generated from the following file:

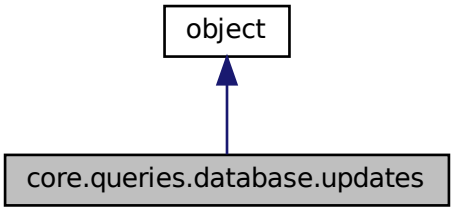
- core/build/lib/queries/database.py

7.19 core.queries.database.updates Class Reference

Inheritance diagram for core.queries.database.updates:



Collaboration diagram for core.queries.database.updates:



Public Member Functions

- def __init__ (self, conn, cur, logger)
- def update_account (self, cid, balance)
- def currency_preference (self, cid, base)
- def __init__ (self, conn, cur, logger)
- def update_account (self, cid, balance)
- def currency_preference (self, cid, base)

Data Fields

- conn
- cur
- db_log

7.19.1 Detailed Description

Definition at line 656 of file database.py.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 __init__() [1/2]

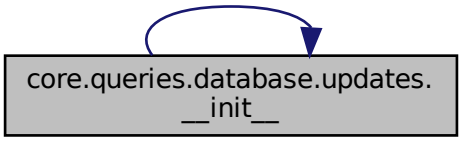
```
def core.queries.database.updates.__init__ (
               self,
               conn,
               cur,
               logger )
```

Definition at line 657 of file database.py.

```
657     def __init__(self, conn, cur, logger):
658         self.conn=conn
659         self.cur=cur
660         self.db_log=logger
```

Referenced by core.queries.database.updates.__init__().

Here is the caller graph for this function:



7.19.2.2 __init__() [2/2]

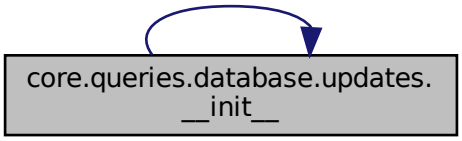
```
def core.queries.database.updates.__init__ (
               self,
               conn,
               cur,
               logger )
```

Definition at line 657 of file database.py.

```
657     def __init__(self, conn, cur, logger):
658         self.conn=conn
659         self.cur=cur
660         self.db_log=logger
```

References core.queries.database.updates.__init__(), queries.database.exists.conn, core.queries.database.↩
exists.conn, queries.database.gets.conn, core.queries.database.gets.conn, queries.database.inserts.conn, core.↩
queries.database.inserts.conn, queries.database.updates.conn, queries.database.database.conn, core.queries.↩
database.updates.conn, core.queries.database.exists.cur, queries.database.exists.cur, queries.database.gets.↩
cur, core.queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.↩
database.updates.cur, queries.database.database.cur, core.queries.database.updates.cur, queries.database.↩
gets.db_log, core.queries.database.gets.db_log, queries.database.inserts.db_log, core.queries.database.inserts.↩
db_log, queries.database.updates.db_log, and core.queries.database.updates.db_log.

Here is the call graph for this function:



7.19.3 Member Function Documentation

7.19.3.1 currency_preference() [1/2]

```
def core.queries.database.updates.currency_preference (
              self,
              cid,
              base )


update currency preference (base ) for the client with client id (cid)

@param cid: client id
@param base: base currency of preference
```

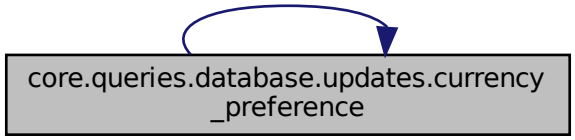
Definition at line 672 of file database.py.

```
672     def currency_preference(self, cid, base):
673         """ update currency preference (base ) for the client with client id (cid)
674 
675         @param cid: client id
676         @param base: base currency of preference
677         """
678         stat = sql.SQL("update clients SET c.currency_id = cur.id FROM currency AS cur  JOIN clients AS
       c WHERE c.client_id={cid} AND cur.currency_name={cur_name}").\
679             format(cur_name=sql.Literal(cur_name), \
680                    cid=sql.Literal(cid))
681         self.cur.execute(stat)
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩
queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩
updates.cur, queries.database.database.cur, core.queries.database.updates.cur, and client.client.format.

Referenced by core.queries.database.updates.currency_preference().

Here is the caller graph for this function:



7.19.3.2 currency_preference() [2/2]

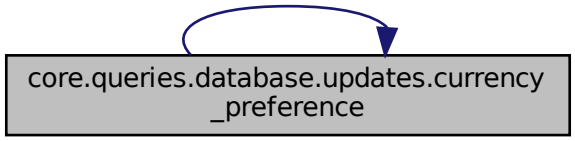
```
def core.queries.database.updates.currency_preference (
              self,
              cid,
              base )

update currency preference (base ) for the client with client id (cid)

@param cid: client id
@param base: base currency of preference
```

Definition at line 672 of file database.py.

```
672     def currency_preference(self, cid, base):
673         """ update currency preference (base ) for the client with client id (cid)
674 
675         @param cid: client id
676         @param base: base currency of preference
677         """
678         stat = sql.SQL("update clients SET c.currency_id = cur.id FROM currency AS cur  JOIN clients AS
       c WHERE c.client_id={cid} AND cur.currency_name={cur_name}").\
679             format(cur_name=sql.Literal(cur_name), \
680                    cid=sql.Literal(cid))
681         self.cur.execute(stat)
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩ updates.cur, queries.database.database.cur, core.queries.database.updates.cur, core.queries.database.updates.↩ currency_preference(), and client.client.format.

Here is the call graph for this function:



7.19.3.3 update_account() [1/2]

```
def core.queries.database.updates.update_account (
              self,
              cid,
              balance )
```

update the banking account with the calculated new balance (CALLED FROM SERVER SIDE)

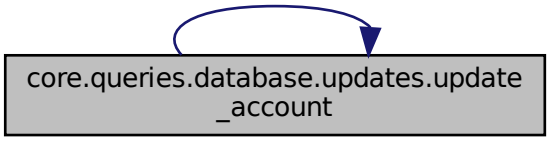
@param cid: client id
@param balance: the account balance

Definition at line 661 of file database.py.

```
661     def update_account(self, cid, balance):
662         """update the banking account with the calculated new balance (CALLED FROM SERVER SIDE)
663 
664         @param cid: client id
665         @param balance: the account balance
666         """
667         stat = sql.SQL("UPDATE banking SET (balance, balance_dt) = ({balance}, {dt}) WHERE
       client_id={cid}").\
668             format(balance=sql.Literal(balance), \
669                    dt=sql.Literal(dt.datetime.now().strftime(TIMESTAMP_FORMAT)), \
670                    cid=sql.Literal(cid))
671         self.cur.execute(stat)
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩updates.cur, queries.database.database.cur, core.queries.database.updates.cur, client.client.format, and core.↩queries.database.updates.update_account().

Here is the call graph for this function:



7.19.3.4 update_account() [2/2]

```
def core.queries.database.updates.update_account (
              self,
              cid,
              balance )
```

update the banking account with the calculated new balance (CALLED FROM SERVER SIDE)

@param cid: client id
@param balance: the account balance

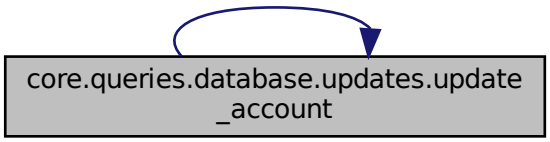
Definition at line 661 of file database.py.

```
661     def update_account(self, cid, balance):
662         """update the banking account with the calculated new balance (CALLED FROM SERVER SIDE)
663 
664         @param cid: client id
665         @param balance: the account balance
666         """
667         stat = sql.SQL("UPDATE banking SET (balance, balance_dt) = ({balance}, {dt}) WHERE
        client_id={cid}").\
668             format(balance=sql.Literal(balance), \
669                    dt=sql.Literal(dt.datetime.now().strftime(TIMESTAMP_FORMAT)), \
670                    cid=sql.Literal(cid))
671         self.cur.execute(stat)
```

References queries.database.exists.cur, core.queries.database.exists.cur, queries.database.gets.cur, core.↩ queries.database.gets.cur, queries.database.inserts.cur, core.queries.database.inserts.cur, queries.database.↩ updates.cur, queries.database.database.cur, core.queries.database.updates.cur, and client.client.format.

Referenced by core.queries.database.updates.update_account().

Here is the caller graph for this function:



7.19.4 Field Documentation

7.19.4.1 conn

`core.queries.database.updates.conn`

Definition at line 658 of file database.py.

Referenced by core.queries.database.updates.__init__(), core.queries.database.database.__init__(), core.↩ queries.database.database.close(), core.queries.database.database.commit(), core.queries.database.database.↩ init(), and core.queries.database.database.rollback().

7.19.4.2 cur

`core.queries.database.updates.cur`

Definition at line 659 of file database.py.

Referenced by core.queries.database.updates.__init__(), core.queries.database.database.__init__(), core.↩ queries.database.database.committed_read(), core.queries.database.updates.currency_preference(), core.↩ queries.database.database.init(), core.queries.database.database.lock_advisory(), core.queries.database.↩ database.repeatable_read(), core.queries.database.database.unlock_advisory(), and core.queries.database.↩ updates.update_account().

7.19.4.3 db_log

`core.queries.database.updates.db_log`

Definition at line 660 of file database.py.

Referenced by core.queries.database.updates.__init__().

The documentation for this class was generated from the following file:

- core/queries/database.py

Chapter 8

File Documentation

8.1 core/__init__.py File Reference

Namespaces

- core

8.2 core/build/lib/client/__init__.py File Reference

Namespaces

- client

8.3 core/build/lib/queries/__init__.py File Reference

Namespaces

- queries

8.4 core/build/lib/queries/exists/__init__.py File Reference

Namespaces

- queries.exists

8.5 core/build/lib/queries/gets/__init__.py File Reference

Namespaces

- queries.gets

8.6 core/build/lib/queries/inserts/__init__.py File Reference

Namespaces

- queries.inserts

8.7 core/build/lib/queries/updates/__init__.py File Reference

Namespaces

- queries.updates

8.8 core/build/lib/server/__init__.py File Reference

Namespaces

- server

8.9 core/client/__init__.py File Reference

Namespaces

- core.client

8.10 core/queries/__init__.py File Reference

Namespaces

- core.queries

8.11 core/queries/exists/__init__.py File Reference

Namespaces

- core.queries.exists

8.12 core/queries/gets/__init__.py File Reference

Namespaces

- core.queries.gets

8.13 core/queries/inserts/__init__.py File Reference

Namespaces

- core.queries.inserts

8.14 core/queries/updates/__init__.py File Reference

Namespaces

- core.queries.updates

8.15 core/server/__init__.py File Reference

Namespaces

- core.server

8.16 build/lib/core/__init__.py File Reference

Namespaces

- core

8.17 build/lib/core/client/__init__.py File Reference

Namespaces

- core.client

8.18 build/lib/core/queries/__init__.py File Reference

Namespaces

- core.queries

8.19 build/lib/core/queries/exists/__init__.py File Reference

Namespaces

- core.queries.exists

8.20 build/lib/core/queries/gets/__init__.py File Reference

Namespaces

- core.queries.gets

8.21 build/lib/core/queries/inserts/__init__.py File Reference

Namespaces

- core.queries.inserts

8.22 build/lib/core/queries/updates/__init__.py File Reference

Namespaces

- core.queries.updates

8.23 build/lib/core/server/__init__.py File Reference

Namespaces

- core.server

8.24 core/build/lib/client/client.py File Reference

Data Structures

- class client.client.Client

Namespaces

- client.client

Functions

- def client.client.get_bank_name ()
- def client.client.get_branch_number ()
- def client.client.get_account_number ()
- def client.client.get_name_reference ()
- def client.client.get_name ()
- def client.client.get_email ()
- def client.client.get_balance ()
- def client.client.get_credid ()
- def client.client.get_rand_pass (L=9)
- def client.client.rand_alphanum (L=9)

Variables

- string client.client.db_configs = "dbname='demo_client' user='tahweela_client' password='tahweela'"
- client.client.filename
- client.client.format
- client.client.filemode
- client.client.logger = logging.getLogger()
- client.client.seed = int.from_bytes(os.urandom(2), 'big')
- client.client.faker = Faker(seed)
- client.client.parser = ArgumentParser()
- client.client.type
- client.client.args = parser.parse_args()
- client.client.cred_id = args.add_contact
- client.client.trader1 = Client()
- client.client.trader2 = Client()

8.25 core/client/client.py File Reference

Data Structures

- class core.client.client.Client

Namespaces

- core.client.client

Functions

- def core.client.client.get_bank_name ()
- def core.client.client.get_branch_number ()
- def core.client.client.get_account_number ()
- def core.client.client.get_name_reference ()
- def core.client.client.get_name ()
- def core.client.client.get_email ()
- def core.client.client.get_balance ()
- def core.client.client.get_credid ()
- def core.client.client.get_rand_pass (L=9)
- def core.client.client.rand_alphanum (L=9)

Variables

- string core.client.client.db_configs = "dbname='demo_client' user='tahweela_client' password='tahweela'"
- core.client.client.filename
- core.client.client.format
- core.client.client.filemode
- core.client.client.logger = logging.getLogger()
- core.client.client.seed = int.from_bytes(os.urandom(2), 'big')
- core.client.client.faker = Faker(seed)
- core.client.client.parser = ArgumentParser()
- core.client.client.type
- core.client.client.args = parser.parse_args()
- core.client.client.cred_id = args.add_contact
- core.client.client.trader1 = Client()
- core.client.client.trader2 = Client()

8.26 build/lib/core/client/client.py File Reference

Data Structures

- class core.client.client.Client

Namespaces

- core.client.client

Functions

- def core.client.client.get_bank_name ()
- def core.client.client.get_branch_number ()
- def core.client.client.get_account_number ()
- def core.client.client.get_name_reference ()
- def core.client.client.get_name ()
- def core.client.client.get_email ()
- def core.client.client.get_balance ()
- def core.client.client.get_credid ()
- def core.client.client.get_rand_pass (L=9)
- def core.client.client.rand_alphanum (L=9)

8.27 core/build/lib/queries/database.py File Reference

Data Structures

- class queries.database.exists
- class queries.database.gets
- class queries.database.inserts
- class queries.database.updates
- class queries.database.database

Namespaces

- queries.database

8.28 core/queries/database.py File Reference

Data Structures

- class core.queries.database.exists
- class core.queries.database.gets
- class core.queries.database.inserts
- class core.queries.database.updates
- class core.queries.database.database

Namespaces

- core.queries.database

8.29 core/tests/database.py File Reference

Data Structures

- class database.ServerDatabaseTest

Namespaces

- database

Functions

- def database.get_bank_name ()
- def database.get_branch_number ()
- def database.get_account_number ()
- def database.get_name_reference ()
- def database.get_name ()
- def database.get_email ()
- def database.get_balance ()
- def database.get_credid ()
- def database.get_rand_pass (L=9)

Variables

- database.seed = int.from_bytes(os.urandom(2), "big")
- database.faker = Faker(seed)
- string database.db_configs = "dbname='demo' user='tahweela' password='tahweela'"
- database.db = database(db_configs)

8.30 build/lib/core/queries/database.py File Reference

Data Structures

- class core.queries.database.exists
- class core.queries.database.gets
- class core.queries.database.inserts
- class core.queries.database.updates
- class core.queries.database.database

Namespaces

- core.queries.database

8.31 core/build/lib/queries/exists.py File Reference

Namespaces

- queries.exists

8.32 core/queries/exists.py File Reference

Namespaces

- core.queries.exists

8.33 build/lib/core/queries/exists.py File Reference

Namespaces

- core.queries.exists

8.34 core/build/lib/queries/exists/banking.py File Reference

Namespaces

- queries.exists.banking

Functions

- def queries.exists.banking.exists (cid)

8.35 core/build/lib/queries/gets/banking.py File Reference

Namespaces

- queries.gets.banking

Functions

- def queries.gets.banking.get_client_id (bid)
- def queries.gets.banking.get_banking_id (cid)
- def queries.gets.banking.get_balance_by_cid (cid)
- def queries.gets.banking.get_balance_by_credid (cred_id)

8.36 core/build/lib/queries/inserts/banking.py File Reference

Namespaces

- queries.inserts.banking

Functions

- def queries.inserts.banking.insert_banking (cid, balance)

8.37 core/build/lib/queries/updates/banking.py File Reference

Namespaces

- queries.updates.banking

Functions

- def queries.updates.banking.update_account (cid, balance)

8.38 core/queries/exists/banking.py File Reference

Namespaces

- core.queries.exists.banking

Functions

- def core.queries.exists.banking.exists (cid)

8.39 core/queries/gets/banking.py File Reference

Namespaces

- core.queries.gets.banking

Functions

- def core.queries.gets.banking.get_client_id (bid)
- def core.queries.gets.banking.get_banking_id (cid)
- def core.queries.gets.banking.get_balance_by_cid (cid)
- def core.queries.gets.banking.get_balance_by_credid (cred_id)

8.40 core/queries/inserts/banking.py File Reference

Namespaces

- core.queries.inserts.banking

Functions

- def core.queries.inserts.banking.insert_banking (cid, balance)

8.41 core/queries/updates/banking.py File Reference

Namespaces

- core.queries.updates.banking

Functions

- def core.queries.updates.banking.update_account (cid, balance)

8.42 build/lib/core/queries/exists/banking.py File Reference

Namespaces

- core.queries.exists.banking

Functions

- def core.queries.exists.banking.exists (cid)

8.43 build/lib/core/queries/gets/banking.py File Reference

Namespaces

- core.queries.gets.banking

Functions

- def core.queries.gets.banking.get_client_id (bid)
- def core.queries.gets.banking.get_banking_id (cid)
- def core.queries.gets.banking.get_balance_by_cid (cid)
- def core.queries.gets.banking.get_balance_by_credid (cred_id)

8.44 build/lib/core/queries/inserts/banking.py File Reference

Namespaces

- core.queries.inserts.banking

Functions

- def core.queries.inserts.banking.insert_banking (cid, balance)

8.45 build/lib/core/queries/updates/banking.py File Reference

Namespaces

- core.queries.updates.banking

Functions

- def core.queries.updates.banking.update_account (cid, balance)

8.46 core/build/lib/queries/exists/clients.py File Reference

Namespaces

- queries.exists.clients

Functions

- def queries.exists.clients.exists (cid)

8.47 core/build/lib/queries/gets/clients.py File Reference

Namespaces

- queries.gets.clients

Functions

- def queries.gets.clients.get_all ()
- def queries.gets.clients.get (cid)
- def queries.gets.clients.get_name (cid)

8.48 core/build/lib/queries/inserts/clients.py File Reference

Namespaces

- queries.inserts.clients

Functions

- def queries.inserts.clients.insert_client (name)

8.49 core/queries/exists/clients.py File Reference

Namespaces

- core.queries.exists.clients

Functions

- def core.queries.exists.clients.exists (cid)

8.50 core/queries/gets/clients.py File Reference

Namespaces

- core.queries.gets.clients

Functions

- def core.queries.gets.clients.get_all ()
- def core.queries.gets.clients.get (cid)
- def core.queries.gets.clients.get_name (cid)

8.51 core/queries/inserts/clients.py File Reference

Namespaces

- core.queries.inserts.clients

Functions

- def core.queries.inserts.clients.insert_client (name)

8.52 build/lib/core/queries/exists/clients.py File Reference

Namespaces

- core.queries.exists.clients

Functions

- def core.queries.exists.clients.exists (cid)

8.53 build/lib/core/queries/gets/clients.py File Reference

Namespaces

- core.queries.gets.clients

Functions

- def core.queries.gets.clients.get_all ()
- def core.queries.gets.clients.get (cid)
- def core.queries.gets.clients.get_name (cid)

8.54 build/lib/core/queries/inserts/clients.py File Reference

Namespaces

- core.queries.inserts.clients

Functions

- def core.queries.inserts.clients.insert_client (name)

8.55 core/build/lib/queries/exists/contacts.py File Reference

Namespaces

- queries.exists.contacts

Functions

- def queries.exists.contacts.exists (cid)

8.56 core/build/lib/queries/gets/contacts.py File Reference

Namespaces

- queries.gets.contacts

Functions

- def queries.gets.contacts.get_all ()
- def queries.gets.contacts.get_banking_id (cid)

8.57 core/build/lib/queries/inserts/contacts.py File Reference

Namespaces

- queries.inserts.contacts

Functions

- def queries.inserts.contacts.insert_contact (cid, cname, bid)

8.58 core/queries/exists/contacts.py File Reference

Namespaces

- core.queries.exists.contacts

Functions

- def core.queries.exists.contacts.exists (cid)

8.59 core/queries/gets/contacts.py File Reference

Namespaces

- core.queries.gets.contacts

Functions

- def core.queries.gets.contacts.get_all ()
- def core.queries.gets.contacts.get_banking_id (cid)

8.60 core/queries/inserts/contacts.py File Reference

Namespaces

- core.queries.inserts.contacts

Functions

- def core.queries.inserts.contacts.insert_contact (cid, cname, bid)

8.61 build/lib/core/queries/exists/contacts.py File Reference

Namespaces

- core.queries.exists.contacts

Functions

- def core.queries.exists.contacts.exists (cid)

8.62 build/lib/core/queries/gets/contacts.py File Reference

Namespaces

- core.queries.gets.contacts

Functions

- def core.queries.gets.contacts.get_all ()
- def core.queries.gets.contacts.get_banking_id (cid)

8.63 build/lib/core/queries/inserts/contacts.py File Reference

Namespaces

- core.queries.inserts.contacts

Functions

- def core.queries.inserts.contacts.insert_contact (cid, cname, bid)

8.64 core/build/lib/queries/exists/credentials.py File Reference

Namespaces

- queries.exists.credentials

Functions

- def queries.exists.credentials.exists (cid)

8.65 core/build/lib/queries/gets/credentials.py File Reference

Namespaces

- queries.gets.credentials

Functions

- def queries.gets.credentials.get_all ()
- def queries.gets.credentials.get_credential (cid)
- def queries.gets.credentials.get_id (cred_id)
- def queries.gets.credentials.get_password (cred_id)
- def queries.gets.credentials.get_credid_with_gid (gid)

8.66 core/build/lib/queries/inserts/credentials.py File Reference

Namespaces

- queries.inserts.credentials

Functions

- def queries.inserts.credentials.new_cred (passcode, cred_id)
- def queries.inserts.credentials.register (cid)

8.67 core/queries/exists/credentials.py File Reference

Namespaces

- core.queries.exists.credentials

Functions

- def core.queries.exists.credentials.exists (cid)

8.68 core/queries/gets/credentials.py File Reference

Namespaces

- core.queries.gets.credentials

Functions

- def core.queries.gets.credentials.get_all ()
- def core.queries.gets.credentials.get_credential (cid)
- def core.queries.gets.credentials.get_id (cred_id)
- def core.queries.gets.credentials.get_password (cred_id)
- def core.queries.gets.credentials.get_credid_with_gid (gid)

8.69 core/queries/inserts/credentials.py File Reference

Namespaces

- core.queries.inserts.credentials

Functions

- def core.queries.inserts.credentials.new_cred (passcode, cred_id)
- def core.queries.inserts.credentials.register (cid)

8.70 build/lib/core/queries/exists/credentials.py File Reference

Namespaces

- core.queries.exists.credentials

Functions

- def core.queries.exists.credentials.exists (cid)

8.71 build/lib/core/queries/gets/credentials.py File Reference

Namespaces

- core.queries.gets.credentials

Functions

- def core.queries.gets.credentials.get_all ()
- def core.queries.gets.credentials.get_credential (cid)
- def core.queries.gets.credentials.get_id (cred_id)
- def core.queries.gets.credentials.get_password (cred_id)
- def core.queries.gets.credentials.get_credid_with_gid (gid)

8.72 build/lib/core/queries/inserts/credentials.py File Reference

Namespaces

- core.queries.inserts.credentials

Functions

- def core.queries.inserts.credentials.new_cred (passcode, cred_id)
- def core.queries.inserts.credentials.register (cid)

8.73 core/build/lib/queries/exists/goods.py File Reference

Namespaces

- queries.exists.goods

Functions

- def queries.exists.goods.exists (gid)

8.74 core/build/lib/queries/gets/goods.py File Reference

Namespaces

- queries.gets.goods

Functions

- def queries.gets.goods.get_all ()
- def queries.gets.goods.get_good (gid)
- def queries.gets.goods.get_commodity (gname, quality=0)
- def queries.gets.goods.get_new_price (gid)

8.75 core/build/lib/queries/inserts/goods.py File Reference

Namespaces

- queries.inserts.goods

Functions

- def queries.inserts.goods.add_good (gname, gquality, gcost, gcid=1)

8.76 core/queries/exists/goods.py File Reference

Namespaces

- core.queries.exists.goods

Functions

- def core.queries.exists.goods.exists (gid)

8.77 core/queries/gets/goods.py File Reference

Namespaces

- core.queries.gets.goods

Functions

- def core.queries.gets.goods.get_all ()
- def core.queries.gets.goods.get_good (gid)
- def core.queries.gets.goods.get_commodity (gname, quality=0)
- def core.queries.gets.goods.get_new_price (gid)

8.78 core/queries/inserts/goods.py File Reference

Namespaces

- core.queries.inserts.goods

Functions

- def core.queries.inserts.goods.add_good (gname, gquality, gcost, gcid=1)

8.79 build/lib/core/queries/exists/goods.py File Reference

Namespaces

- core.queries.exists.goods

Functions

- def core.queries.exists.goods.exists (gid)

8.80 build/lib/core/queries/gets/goods.py File Reference

Namespaces

- core.queries.gets.goods

Functions

- def core.queries.gets.goods.get_all ()
- def core.queries.gets.goods.get_good (gid)
- def core.queries.gets.goods.get_commodity (gname, quality=0)
- def core.queries.gets.goods.get_new_price (gid)

8.81 build/lib/core/queries/inserts/goods.py File Reference

Namespaces

- core.queries.inserts.goods

Functions

- def core.queries.inserts.goods.add_good (gname, gquality, gcost, gcid=1)

8.82 core/build/lib/queries/exists/owners.py File Reference

Namespaces

- queries.exists.owners

Functions

- def queries.exists.owners.exists ()

8.83 core/build/lib/queries/inserts/owners.py File Reference

Namespaces

- queries.inserts.owners

Functions

- def queries.inserts.owners.add_owner (oid, gid)

8.84 core/build/lib/queries/updates/owners.py File Reference

Namespaces

- queries.updates.owners

Functions

- def queries.updates.owners.update_owner (oid, gid)

8.85 core/queries/exists/owners.py File Reference

Namespaces

- core.queries.exists.owners

Functions

- def core.queries.exists.owners.exists ()

8.86 core/queries/inserts/owners.py File Reference

Namespaces

- core.queries.inserts.owners

Functions

- def core.queries.inserts.owners.add_owner (oid, gid)

8.87 core/queries/updates/owners.py File Reference

Namespaces

- core.queries.updates.owners

Functions

- def core.queries.updates.owners.update_owner (oid, gid)

8.88 build/lib/core/queries/exists/owners.py File Reference

Namespaces

- core.queries.exists.owners

Functions

- def core.queries.exists.owners.exists ()

8.89 build/lib/core/queries/inserts/owners.py File Reference

Namespaces

- core.queries.inserts.owners

Functions

- def core.queries.inserts.owners.add_owner (oid, gid)

8.90 build/lib/core/queries/updates/owners.py File Reference

Namespaces

- core.queries.updates.owners

Functions

- def core.queries.updates.owners.update_owner (oid, gid)

8.91 core/build/lib/queries/gets/currency.py File Reference

Namespaces

- queries.gets.currency

Functions

- def queries.gets.currency.to_dollar (cid)

8.92 core/queries/gets/currency.py File Reference

Namespaces

- core.queries.gets.currency

Functions

- def core.queries.gets.currency.to_dollar (cid)

8.93 build/lib/core/queries/gets/currency.py File Reference

Namespaces

- core.queries.gets.currency

Functions

- def core.queries.gets.currency.to_dollar (cid)

8.94 core/build/lib/queries/gets/ledger.py File Reference

Namespaces

- queries.gets.ledger

Functions

- def queries.gets.ledger.get_transactions (st_dt, end_dt=dt.datetime.now())
- def queries.gets.ledger.get_sells (dest, st_dt, end_dt=None)
- def queries.gets.ledger.get_last_timestamp ()

8.95 core/build/lib/queries/inserts/ledger.py File Reference

Namespaces

- queries.inserts.ledger

Functions

- def queries.inserts.ledger.insert_trx (des, src, gid)

8.96 core/queries/gets/ledger.py File Reference

Namespaces

- core.queries.gets.ledger

Functions

- def core.queries.gets.ledger.get_transactions (st_dt, end_dt=dt.datetime.now())
- def core.queries.gets.ledger.get_sells (dest, st_dt, end_dt=None)
- def core.queries.gets.ledger.get_last_timestamp ()

8.97 core/queries/inserts/ledger.py File Reference

Namespaces

- core.queries.inserts.ledger

Functions

- def core.queries.inserts.ledger.insert_trx (des, src, gid)

8.98 build/lib/core/queries/gets/ledger.py File Reference

Namespaces

- core.queries.gets.ledger

Functions

- def core.queries.gets.ledger.get_transactions (st_dt, end_dt=dt.datetime.now())
- def core.queries.gets.ledger.get_sells (dest, st_dt, end_dt=None)
- def core.queries.gets.ledger.get_last_timestamp ()

8.99 build/lib/core/queries/inserts/ledger.py File Reference

Namespaces

- core.queries.inserts.ledger

Functions

- def core.queries.inserts.ledger.insert_trx (des, src, gid)

8.100 core/build/lib/queries/gets/owner.py File Reference

Namespaces

- queries.gets.owner

Functions

- def queries.gets.owner.get_all ()
- def queries.gets.owner.get_good_owner (gid)
- def queries.gets.owner.get_owner_goods (oid)

8.101 core/queries/gets/owner.py File Reference

Namespaces

- core.queries.gets.owner

Functions

- def core.queries.gets.owner.get_all ()
- def core.queries.gets.owner.get_good_owner (gid)
- def core.queries.gets.owner.get_owner_goods (oid)

8.102 build/lib/core/queries/gets/owner.py File Reference

Namespaces

- core.queries.gets.owner

Functions

- def core.queries.gets.owner.get_all ()
- def core.queries.gets.owner.get_good_owner (gid)
- def core.queries.gets.owner.get_owner_goods (oid)

8.103 core/build/lib/server/server.py File Reference

Namespaces

- server.server

Functions

- def server.server.get_credid ()
- def server.server.is_email (email)
- def server.server.authenticate (user, passcode)
- def server.server.unauthorized ()
- def server.server.register_client ()
- def server.server.add_bank_account ()
- def server.server.add_contact ()
- def server.server.get_balance ()
- def server.server.update_ledger ()
- def server.server.make_transaction ()

Variables

- server.server.seed = int.from_bytes(os.urandom(2), 'big')
- string server.server.db_configs = "dbname='demo' user='tahweela' password='tahweela'"
- server.server.db = database(db_configs)
- server.server.filename
- server.server.format
- server.server.filemode
- server.server.level
- server.server.logger = logging.getLogger()
- server.server.app = Flask('tahweela')
- server.server.auth = HTTPBasicAuth()
- server.server.client_passcode = None
- server.server.client_cred_id = None
- server.server.debug

8.104 core/server/server.py File Reference

Namespaces

- core.server.server

Functions

- def core.server.server.get_credid ()
- def core.server.server.is_email (email)
- def core.server.server.authenticate (user, passcode)
- def core.server.server.unauthorized ()
- def core.server.server.register_client ()
- def core.server.server.add_bank_account ()
- def core.server.server.add_contact ()
- def core.server.server.get_balance ()
- def core.server.server.update_balance_preference ()
- def core.server.server.update_ledger ()
- def core.server.server.make_transaction ()

Variables

- core.server.server.seed = int.from_bytes(os.urandom(2), 'big')
- string core.server.server.db_configs = "dbname='demo' user='tahweela' password='tahweela'"
- core.server.server.db = database(db_configs)
- core.server.server.filename
- core.server.server.format
- core.server.server.filemode
- core.server.server.level
- core.server.server.logger = logging.getLogger()
- core.server.server.app = Flask('tahweela')
- core.server.server.auth = HTTPBasicAuth()
- core.server.server.client_passcode = None
- core.server.server.client_cred_id = None
- core.server.server.debug

8.105 build/lib/core/server/server.py File Reference

Namespaces

- core.server.server

Functions

- def core.server.server.get_credid ()
- def core.server.server.is_email (email)
- def core.server.server.authenticate (user, passcode)
- def core.server.server.unauthorized ()
- def core.server.server.register_client ()
- def core.server.server.add_bank_account ()
- def core.server.server.add_contact ()
- def core.server.server.get_balance ()
- def core.server.server.update_balance_preference ()
- def core.server.server.update_ledger ()
- def core.server.server.make_transaction ()

8.106 core/connection_cursor.py File Reference

Namespaces

- core.connection_cursor

8.107 build/lib/core/connection_cursor.py File Reference

Namespaces

- core.connection_cursor

8.108 core/tahweela.egg-info/dependency_links.txt File Reference

8.109 tahweela.egg-info/dependency_links.txt File Reference

8.110 core/tahweela.egg-info/SOURCES.txt File Reference

8.111 tahweela.egg-info/SOURCES.txt File Reference

8.112 core/tahweela.egg-info/top_level.txt File Reference

8.113 tahweela.egg-info/top_level.txt File Reference

8.114 core/tests/core_test.py File Reference

Data Structures

- class core_test.Client
- class core_test.RestfulTest

Namespaces

- core_test

Functions

- def core_test.get_bank_name ()
- def core_test.get_branch_number ()
- def core_test.get_account_number ()
- def core_test.get_name_reference ()
- def core_test.get_name ()
- def core_test.get_email ()
- def core_test.get_balance ()
- def core_test.get_credid ()
- def core_test.get_rand_pass (L=9)
- def core_test.rand_alphanum (L=9)
- def core_test.get_amount ()

Variables

- core_test.seed = int.from_bytes(os.urandom(3), 'big')
- core_test.faker = Faker(seed)
- string core_test.db_configs = "dbname='demo' user='tahweela' password='tahweela'"
- core_test.filename
- core_test.format
- core_test.filemode
- core_test.logger = logging.getLogger()

8.115 core/utils.py File Reference

Data Structures

- class core.utils.Currency
- class core.utils.PaymentGate

Namespaces

- core.utils

Functions

- def core.utils.exchangerate_rate (base, pref)
- def core.utils.fixer_rate (base, pref)
- def core.utils.exchange (base, pref)
- def core.utils.daily_limit (pref=EUR)
- def core.utils.weekly_limit (pref=EUR)
- def core.utils.process_cur (cur)
- def core.utils.unwrap_cur (cur)

Variables

- core.utils.seed = int.from_bytes(os.urandom(2), 'big')
- string core.utils.TIMESTAMP_FORMAT = "%Y-%m-%d %H:%M:%S.%f"
- float core.utils.FEE = 0.01
- float core.utils.QUALITY_REDUCTION = 0.1
- int core.utils.MAX_COST = 1000000
- int core.utils.MAX_GOODS = 100
- string core.utils.REGISTER = "/api/v0.1/register"
- string core.utils.LEDGER = "/api/v0.1/ledger"
- string core.utils.CONTACTS = "/api/v0.1/contacts"
- string core.utils.PURCHASE = "/api/v0.1/purchase"
- string core.utils.GOODS = "/api/v0.1/goods"
- string core.utils.GOODS_URL = "http://localhost:5000/api/v0.1/goods"
- string core.utils.BALANCE_URL = "http://localhost:5000/api/v0.1/balance"
- string core.utils.BALANCE = "/api/v0.1/balance"
- string core.utils.CURRENCY_URL = "http://localhost:5000/api/v0.1/currency"
- string core.utils.CURRENCY = "/api/v0.1/currency"
- string core.utils.CONTACTS_URL = "http://localhost:5000/api/v0.1/contacts"
- string core.utils.REGISTER_URL = "http://localhost:5000/api/v0.1/register"
- string core.utils.LEDGER_URL = "http://localhost:5000/api/v0.1/ledger"
- string core.utils.PURCHASE_URL = "http://localhost:5000/api/v0.1/purchase"
- string core.utils.ADD_BANK_ACCOUNT_URL = "http://localhost:5000/api/v0.1/addbank"
- string core.utils.ADD_BANK_ACCOUNT = "/api/v0.1/addbank"
- string core.utils.TRANSACTION_URL = "http://localhost:5000/api/v0.1/transaction"
- string core.utils.TRANSACTION = "/api/v0.1/transaction"
- int core.utils.MAX_CRED_ID = 9223372036854775807
- int core.utils.MAX_BALANCE = MAX_COST∗10
- float core.utils.STOCHASTIC_TRADE_THRESHOLD = 0.9
- int core.utils.DAILY_LIMIT_EGP = 10000

- int core.utils.WEEKLY_LIMIT_EGP = 50000
- string core.utils.EUR = 'EUR'
- string core.utils.EGP = 'EGP'
- string core.utils.USD = 'USD'
- string core.utils.db_configs = "dbname='demo' user='tahweela' password='tahweela'"
- core.utils.filename
- core.utils.format
- core.utils.filemode
- core.utils.log = logging.getLogger()

8.116 build/lib/core/utils.py File Reference

Data Structures

- class core.utils.Currency
- class core.utils.PaymentGate

Namespaces

- core.utils

Functions

- def core.utils.exchangerate_rate (base, pref)
- def core.utils.fixer_rate (base, pref)
- def core.utils.exchange (base, pref)
- def core.utils.daily_limit (pref=EUR)
- def core.utils.weekly_limit (pref=EUR)
- def core.utils.process_cur (cur)
- def core.utils.unwrap_cur (cur)

8.117 README.md File Reference

8.118 requirements.txt File Reference

8.119 setup.py File Reference

Data Structures

- class setup.CleanCommand

Namespaces

- setup

Functions

- def setup.read (fname)

Variables

- string setup.description
- setup.name
- setup.version
- setup.author
- setup.author_email
- setup.license
- setup.packages
- setup.long_description
- setup.cmdclass