smartOBD

0.7.5

Generated by Doxygen 1.8.17

Chapter 1

Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Namespace Documentation

3.1 main Namespace Reference

Functions

- def main ()

    main function

3.1.1 Function Documentation

3.1.1.1 main()

```
def main.main ( )
```

main function

initialization and interface for smartOBD Simple command line interface, with choices for asynchronous data and a full query

3.2 smartOBD Namespace Reference

Namespaces

- asynco
- test_commands

Variables

- string cur = ''

  database cursor for use in both async and full query
- string dbtable = ''

  database table name for use in both async and full query
- string dbconn = ''

  database connection for the psycopg2 package

3.2.1 Variable Documentation

3.2.1.1 cur

```
string smartOBD.cur = ''
```

database cursor for use in both async and full query

3.2.1.2 dbconn

```
string smartOBD.dbconn = ''
```

database connection for the psycopg2 package

3.2.1.3 dbtable

```
string smartOBD.dbtable = ''
```

database table name for use in both async and full query

is updated through the @userGet function

3.3 smartOBD.asynco Namespace Reference

Functions

- def userGet ()

  User Get.
- def writeToDB ()

  Write to Database.
- def new_speed (s)

  new_speed
- def new_rpm (r)

  new_rpm
- def new_temp (t)

  new_temp
- def new_fuel (f)

  new_temp
- def getAsync (dur)

  getAsync

Variables

- list data = [datetime.datetime.now()]

  storage of data to be updated to the database

3.3.1 Function Documentation

3.3.1.1 getAsync()

```
def smartOBD.asynco.getAsync (
              dur )
```

getAsync

sets connection for async, starts connection and waits for key entry to stop connection

3.3.1.2 new_fuel()

```
def smartOBD.asynco.new_fuel (
              f )
```

new_temp

callback for fuel level writing to @data

3.3.1.3 new_rpm()

```
def smartOBD.asynco.new_rpm (
              r )
```

new_rpm

callback for rpm writing to @data

3.3.1.4 new_speed()

```
def smartOBD.asynco.new_speed (
              s )
```

new_speed

callback for speed writing to @data

3.3.1.5 new_temp()

```
def smartOBD.asynco.new_temp (
              t )
```

new_temp

callback for coolant temperature writing to @data

3.3.1.6 userGet()

```
def smartOBD.asynco.userGet ( )
```

User Get.

fetches car table and sets dbtable to carX_temp inputs: username sorts through database to find final car table

3.3.1.7 writeToDB()

```
def smartOBD.asynco.writeToDB ( )
```

Write to Database.

erases data from database and writes updated values to database

3.3.2 Variable Documentation

3.3.2.1 data

```
list smartOBD.asynco.data = [datetime.datetime.now()]
```

storage of data to be updated to the database

3.4 smartOBD.test_commands Namespace Reference

Functions

- def userGet (dbconn, cur)

  User Get.
- def fullQuery ()

  fullQuery

3.4.1 Function Documentation

3.4.1.1 fullQuery()

```
def smartOBD.test_commands.fullQuery ( )
```

fullQuery

parses through all OBDCommands as a dictionary, and queries the car with all commands,
appends results to a data array,
checks database for all columns and appends new ones,
finally, writes to database

```
# dictionary generation
for key, i in test_dict.items():
    # print(key, test_dict[key])
    command.append((key, test_dict[key]))
#basic loop for running commands from dictionary
for i in range(0, len(temp2)):
res = str((car.query(temp2[i])).value)
description = str(temp2[i])
if(res != 'None'):
    columns.append(description.rsplit(': ', 1)[1])
    results.append(str(res).rsplit(' ', 1)[0])
# after running all queries, final column generation and insertion
# * length checking for all arrays
if(len(columns) != len(results)):
    print("Results error")
# *final loop for database access
else:
    print("Parsing success")
    print(len(columns),"=",len(results))
    # * checking all columns for existence
    for i in range(1, len(columns)):
        data = columns[i]
        data = data.replace("'", " ")
        data = data.replace("\"", " ")
        cur.execute("select exists(select 1 from information_schema.columns where table_name='%s' and
       column_name='%s');",
                    (AsIs(dbtable), AsIs(data)))
        test = cur.fetchone()[0]
        if(not test):
            data.replace("'", " ")
            data.replace("\"", " ")
            cur.execute("alter table %s add column \"%s\" VARCHAR(2000)",
                        (AsIs(dbtable), AsIs(data)))
            print("TABLE ALTERED",data)
    # * final insertion
    dbconn.commit()
    q1 = sql.SQL("insert into {0} values ({1})").format(sql.Identifier(dbtable),
                                                         sql.SQL(', ').join(sql.Placeholder() *
       len(results)))
    # print(results)
    cur.execute(q1, results)
    dbconn.commit()
    print("Successful Read")
```

3.4.1.2 userGet()

```
def smartOBD.test_commands.userGet (
                dbconn,
                cur )
```

User Get.

fetches car table and sets dbtable to carX
inputs: username
sorts through database to find final car table

3.5 test_commands Namespace Reference

3.5.1 Detailed Description

Parsing through all OBDCommands as a dictionary, and then querying the car with all of them.
Takes results, and writes them to database

Chapter 4

File Documentation

4.1 dynamic_commands/main.py File Reference

Namespaces

- main

Functions

- def main.main ()

  main function

4.2 dynamic_commands/smartOBD/__init__.py File Reference

Namespaces

- smartOBD

Variables

- string smartOBD.cur = ''

  database cursor for use in both async and full query
- string smartOBD.dbtable = ''

  database table name for use in both async and full query
- string smartOBD.dbconn = ''

  database connection for the psycopg2 package

4.3 dynamic_commands/smartOBD/asynco.py File Reference

Namespaces

- smartOBD.asynco

Functions

- def smartOBD.asynco.userGet ()

    User Get.
- def smartOBD.asynco.writeToDB ()

    Write to Database.
- def smartOBD.asynco.new_speed (s)

    new_speed
- def smartOBD.asynco.new_rpm (r)

    new_rpm
- def smartOBD.asynco.new_temp (t)

    new_temp
- def smartOBD.asynco.new_fuel (f)

    new_temp
- def smartOBD.asynco.getAsync (dur)

    getAsync

Variables

- list smartOBD.asynco.data = [datetime.datetime.now()]

    storage of data to be updated to the database

4.4 dynamic_commands/smartOBD/test_commands.py File Reference

Namespaces

- smartOBD.test_commands
- test_commands

Functions

- def smartOBD.test_commands.userGet (dbconn, cur)

    User Get.
- def smartOBD.test_commands.fullQuery ()

    fullQuery

Index