FOTA

Generated by Doxygen 1.8.13

Contents

Chapter 1

pythonOTAServer

OTA server TCP/IP custom protocol

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

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces 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

Functions

- def verifyFinishedUpdate (data)

Variables

- string ackClient = b"@"
- string approvedUpdate = b"@4#"
- string HOST = '209.97.145.137'
- int PORT = 4000
- list binaryCode = [ ]
- s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
- data = s.recv(1024)
- def finishedUpdate = verifyFinishedUpdate(data)
- int TIMEOUT = 5
- string host = '127.0.0.1'
- int port = 10000
- sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
- buf = sock.recv(1024)

6.1.1 Function Documentation

6.1.1.1 verifyFinishedUpdate()

```
def client.verifyFinishedUpdate (
              data )
```

```
21 def verifyFinishedUpdate(data):
22      match = re.search(r'^@([0-9A-F]{1,2})##$', data)
23      if match:
24          return 0
25      else:
26          return 1
27
```

6.1.2 Variable Documentation

6.1.2.1 ackClient

```
string client.ackClient = b"@"
```

6.1.2.2 approvedUpdate

```
string client.approvedUpdate = b"@4#"
```

6.1.2.3 binaryCode

```
list client.binaryCode = []
```

6.1.2.4 buf

```
client.buf = sock.recv(1024)
```

6.1.2.5 data

```
string client.data = s.recv(1024)
```

6.1.2.6 finishedUpdate

```
def client.finishedUpdate = verifyFinishedUpdate(data)
```

6.1.2.7 host

```
string client.host = '127.0.0.1'
```

6.1.2.8 HOST

```
string client.HOST = '209.97.145.137'
```

6.1.2.9 port

```
int client.port = 10000
```

6.1.2.10 PORT

```
int client.PORT = 4000
```

6.1.2.11 s

```
client.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

6.1.2.12 sock

```
client.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

6.1.2.13 TIMEOUT

```
int client.TIMEOUT = 5
```

6.2 customlogger Namespace Reference

Classes

- class CustomLogger

6.3 NoClient Namespace Reference

Functions

- def verifyFinishedUpdate (data)

Variables

- string ackClient = b"@"
- string approvedUpdate = b"@4#"
- string HOST = '209.97.145.137'
- int PORT = 4000
- list binaryCode = [ ]
- s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
- data = s.recv(1024)
- def finishedUpdate = verifyFinishedUpdate(data)

6.3.1 Function Documentation

6.3.1.1 verifyFinishedUpdate()

```
def NoClient.verifyFinishedUpdate (
              data )
```

```
21 def verifyFinishedUpdate(data):
22      match = re.search(r'^@([0-9A-F]{1,2})##$', data)
23      if match:
24           return 0
25      else:
26           return 1
27
```

6.3.2 Variable Documentation

6.3.2.1 ackClient

```
string NoClient.ackClient = b"@"
```

6.3.2.2 approvedUpdate

```
string NoClient.approvedUpdate = b"@4#"
```

6.3.2.3 binaryCode

```
list NoClient.binaryCode = [ ]
```

6.3.2.4 data

```
string NoClient.data = s.recv(1024)
```

6.3.2.5 finishedUpdate

```
def NoClient.finishedUpdate = verifyFinishedUpdate(data)
```

6.3.2.6 HOST

```
string NoClient.HOST = '209.97.145.137'
```

6.3.2.7 PORT

```
int NoClient.PORT = 4000
```

6.3.2.8 s

```
NoClient.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

6.4 Server Namespace Reference

Namespaces

- py

  Create a server and enable REST API for firmware updates.

6.5 server Namespace Reference

Functions

- def readArgs ()
- def configureLogger (logFile)
- def createServer (connectionsList, clientsList)
- def acceptConnections (connectionsList, clientsList, sock, logger)
- def verifyStartedConnection (connections, connection)
- def getMCUID (connection, address, logger)
- def verifyClientId (connection, mcuid, logger)
- def lookForClientInDatabase (mcuid, logger)
- def closeConnection (connection, logger)
- def prepareUpdate (clientInfo, clientsList, logger)
- def bufferData (filename, binaryFileLines, logger)
- def sendUpdate (connection, address, mcuid, clientToUpdate, sock, logger)
- def validateCodeChunk (codechunk)
- def connectSocket (socket)

Variables

- int DEBUG_ON = 1

  Enable 1 Disable 0 Debug.
- string LOG_PATH_DEFAULT = "/tmp"

  Defaults files to log.
- string LOG_FILENAME_DEFAULT = "/otaserver.log"
- LOG_LEVEL = logging.INFO
- int SUCCESSFUL = 0
- int CLIENT_UNVERIFIED = 1
- int MISSED_FILENAME = 2
- int UNFORMATTED_MCUID = 3
- int ALREADY_STARTED_CONNECTION = 4
- int NOT_STARTED_CONNECTION = 4
- int UNABLE_BUFFERING_CODE = 5
- int UNPROPER_FILE_FORMAT = 6
- int TIMEOUT_CONNECTION = 7
- int INVALID_CODE_CHUNK = 8
- int BUFFERING_CODE_INCOMPLETE = 9
- int READY_TO_UPDATE = 10
- int TIMEOUT = 30
- string HOST_DEFAULT = ''
- int PORT_DEFAULT = 4000
- string PATH_BINARY_FILE_DEFAULT = "/root/fota/otaserver/bin"
- string PATH_DATABASE_DEFAULT = "/root/fota/otaserver/database/devices2update.txt"
- string allowedUpdate = b'@4#'
- string bannedUpdate = b'@0#'
- string ackClient = b'@'
- int bufferSizeFile = 300
- int maxBytes = 1024
- string pathBinaryFiles = ""
- string databaseName = ""
- string host = ""
- string port = ""
- string logPath = ""
- list inputs = [ ]
- def args = readArgs()
- string logFile = logPath + LOG_FILENAME_DEFAULT
- def logger = configureLogger(logFile)
- list connectionsList = [ ]
- list clientsList = [ ]
- int MAX_OPEN_CONNECTIONS = 5
- int connectedClients = 0
- list openConnections = [ ]
- serversocket = socket.socket (socket.AF_INET, socket.SOCK_STREAM)
- target
- connectSocket
- sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
- list outputs = [ ]
- list excepts = [ ]
- readable
- writable
- exceptional
- clientsock
- clientaddr
- data = readableSocket.recv(1024)

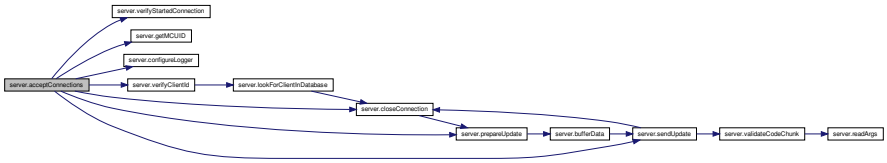
6.5.1 Function Documentation

6.5.1.1 acceptConnections()

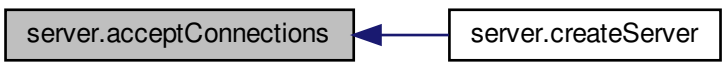
```
def server.acceptConnections (
              connectionsList,
              clientsList,
              sock,
              logger )
```

```
121 def acceptConnections(connectionsList, clientsList, sock, logger):
122     while True:
123          #Accept the connection from the address
124          sock.setblocking(1)
125          connection, address = sock.accept()
126          logger.info("Accepted connection from {}".format(connection))
127          if (verifyStartedConnection(connectionsList, connection) ==
     NOT_STARTED_CONNECTION):
128               connectionsList.append(connection)
129               #Verify if id client belongs to database
130               mcuid = getMCUID(connection, address, logger)
131               date = datetime.datetime.now().strftime("%d-%m-%y_%H-%M")
132               logFile = logPath + "/" + str(mcuid) + "-" + date +".txt"
133
134               print("logfile {}".format(logFile))
135               logger = configureLogger(logFile)
136               statusIdClient, clientInfo = verifyClientId(connection, mcuid, logger)
137               if (statusIdClient == SUCCESSFUL):
138                    connection.send(allowedUpdate)
139                    logger.info("Allowing update to {}".format(mcuid))
140                    logger.info("Starting update to {}".format(mcuid))
141                    prepareUpdate(clientInfo, clientsList, logger)
142                    sendUpdate(connection, address, mcuid, clientsList, sock, logger)
143               else:
144                    logger.warning("Banned connection from {}".format(address))
145                    logger.error("Closing connection")
146                    connection.send(bannedUpdate)
147                    closeConnection(connection, logger)
148
149
```

Here is the call graph for this function:



Here is the caller graph for this function:

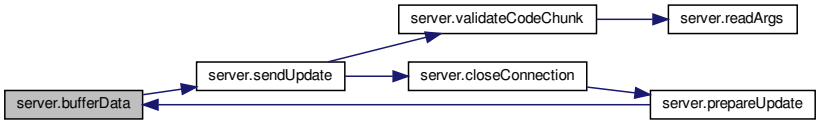


6.5.1.2 bufferData()

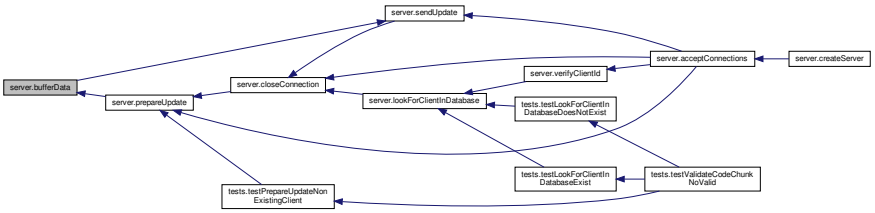
```
def server.bufferData (
              filename,
              binaryFileLines,
              logger )
```

```
224 def bufferData(filename, binaryFileLines, logger):
225      countLines = 0
226      path = pathBinaryFiles+"/"+ filename
227      if (os.path.exists(path)== False):
228           logger.error("Required file does not exist {}".format(path))
229           return UNABLE_BUFFERING_CODE
230      with open(path, 'r') as file:
231           logger.info("Opening file {}".format(path))
232           line = file.readline()
233           countLines += 1
234           while (line and countLines < bufferSizeFile):
235                line = file.readline()
236                chunk = re.findall(r'^S1+[0-9A-F]+\w|^S2+[0-9A-F]+\w|^S3+[0-9A-F]+\w', line)
237                if (len(chunk)>0):
238                     binaryFileLines.append(chunk[0])
239                countLines += 1
240           if(file.readline() == ''):
241                file.close()
242                return SUCCESSFUL
243           file.close()
244           return BUFFERING_CODE_INCOMPLETE
245      return UNABLE_BUFFERING_CODE
246
```

Here is the call graph for this function:



Here is the caller graph for this function:

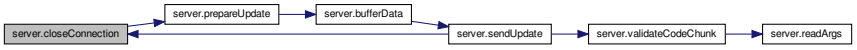


6.5.1.3 closeConnection()

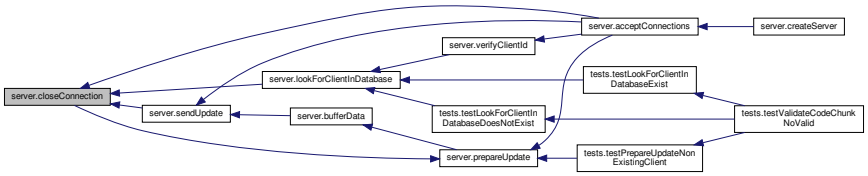
```
def server.closeConnection (
              connection,
              logger )
```

```
199 def closeConnection(connection, logger):
200      logger.info("Close of connection has been requested")
201      connection.close()
202      return SUCCESSFUL
203
```

Here is the call graph for this function:



Here is the caller graph for this function:

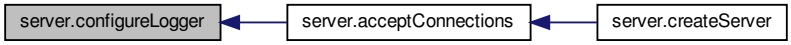


6.5.1.4 configureLogger()

```
def server.configureLogger (
              logFile )
```

```
83 def configureLogger(logFile):
84      # Logger name
85      logger = logging.getLogger(logFile)
86      # Set the log level to LOG_LEVEL
87      logger.setLevel(LOG_LEVEL)
88      # Make a handler that wirtes to a file, making a new file at midnight and keeping
89      # 3 backups
90      handler = logging.handlers.TimedRotatingFileHandler(logFile, when="midnight", backupCount =3)
91      #Format each log message like this
92      formatter = logging.Formatter('%(asctime)s %(levelname)-8s %(message)s')
93      #Attach the formatter to the handler
94      handler.setFormatter(formatter)
95      #Attach the handler to the logger
96      logger.addHandler(handler)
97      #Replacement of stdout with loggin to file at INFO level
98      sys.stdout = CustomLogger(logger, logging.INFO)
99      #Replacement of stdout with loggin to file at ERROR level
100      sys.stderr = CustomLogger(logger, logging.ERROR)
101      return logger
102
```

Here is the caller graph for this function:



6.5.1.5 connectSocket()

```
def server.connectSocket (
              socket )
```

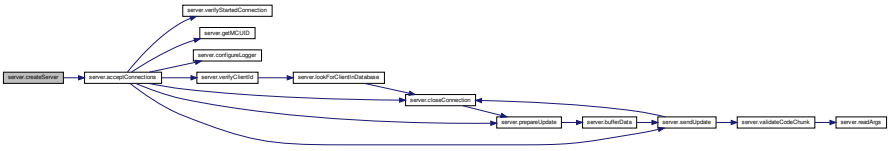
```
20 def connectSocket(socket):
21      data = ""
22      client, address = socket.accept()
23      client.settimeout(TIMEOUT)
24      data =  client.recv()
25      openConnections.append(client)
26      if data != "@":
27           logger.debug("{u} connected".format(u=address))
28           client.send("OK")
29      openConnections.remove()
30      client.close()
31
32
33
```

6.5.1.6 createServer()

```
def server.createServer (
              connectionsList,
              clientsList )
```

```
103 def createServer(connectionsList, clientsList):
104
107      sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
108      hostname = socket.gethostname()
109      logger.info("Hostname {}".format(hostname))
110      #Cleaning previous connections
111      sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
112      #For binding address and port
113      # 0.0.0.0 because that makes our server available over any IP address
114      sock.bind((host,port))
115      #Listen argument: Maximum in queue pendings
116      sock.listen(5)
117      inputs.append(sock)
118      logger.info("Server ready to listen")
119      acceptConnections(connectionsList, clientsList, sock, logger)
120
```

Here is the call graph for this function:

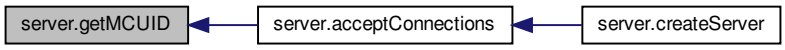


6.5.1.7 getMCUID()

```
def server.getMCUID (
              connection,
              address,
              logger )
```

```
157 def getMCUID(connection, address, logger):
158      receivedData = connection.recv(maxBytes)
159      logger.info("Server has received data from {}".format(address))
160      mcuid = re.findall("^\@(\d{25})[0-9]*#$", receivedData.decode("utf-8"))
161
162      if(len(mcuid) == 1):
163           return mcuid[0]
164      else:
165           return UNFORMATTED_MCUID
166
```

Here is the caller graph for this function:



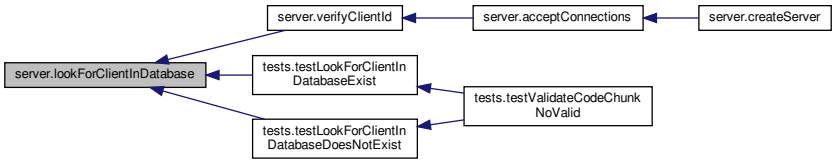
6.5.1.8 lookForClientInDatabase()

```
def server.lookForClientInDatabase (
              mcuid,
              logger )
```

```
179 def lookForClientInDatabase(mcuid, logger):
180
181      clientInfo = []
182      logger.info("Opening file {}, to verify MCUID {}".format(databaseName,mcuid))
183      try:
184           with open(databaseName, 'r') as file:              for line in file:
185                     clientInfo = line.split(',')
186                     if (len(clientInfo) == 3):
187                          if (clientInfo[0] == mcuid):
188                              return SUCCESSFUL, clientInfo
189                          else:
190                               continue
191                     else:
192                          return UNPROPER_FILE_FORMAT, clientInfo
193           file.close()
194      except:
195           logger.error("Error opening file")
196      return CLIENT_UNVERIFIED, clientInfo
197
198
```

Here is the call graph for this function:

Here is the caller graph for this function:

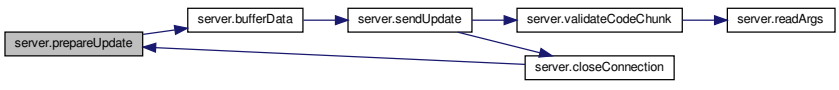


6.5.1.9 prepareUpdate()

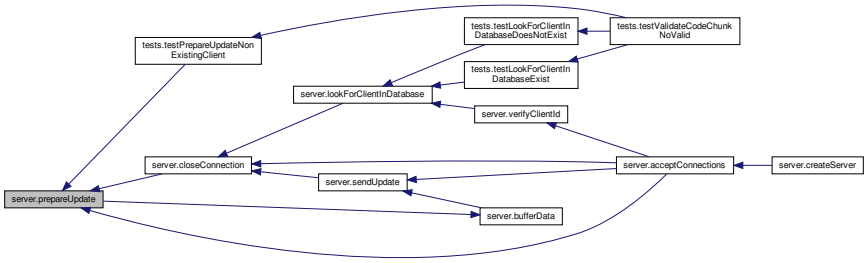
```
def server.prepareUpdate (
              clientInfo,
              clientsList,
              logger )
```

```
204 def prepareUpdate(clientInfo, clientsList, logger):
205      logger.info("Preparing update for {}".format(clientInfo))
206      #Buffer data from file to be sent to specific client
207      binaryFileLines = []
208      bufferingStatus = bufferData(clientInfo[1], binaryFileLines, logger)
209      logger.info("Status prepare update {}".format(bufferingStatus))
210      if (bufferingStatus == SUCCESSFUL):
211           FWVersion = re.findall(r"([0-9A-F]+)", clientInfo[2])
212           binaryFileLines.append("@{}##".format(FWVersion[0]))
213           logger.info("Buffered code {}".format(binaryFileLines))
214           logger.info("Code buffer done successfully")
215           clientsList.append({"mcuid":clientInfo[0], "filename":clientInfo[1], "codelines": binaryFileLines
     , "status": READY_TO_UPDATE})
216           return SUCCESSFUL
217      if (bufferingStatus == BUFFERING_CODE_INCOMPLETE):
218
219           logger.info("Incomplete buffering code")
220           return BUFFERING_CODE_INCOMPLETE
221      else:
222           return UNABLE_BUFFERING_CODE
223
```

Here is the call graph for this function:



Here is the caller graph for this function:

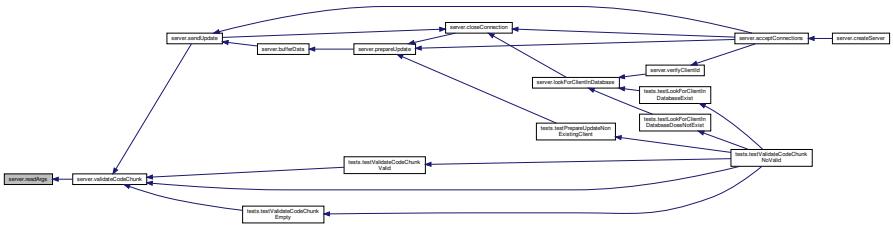


6.5.1.10 readArgs()

```
def server.readArgs ( )
```

```
73 def readArgs():
74       parser = argparse.ArgumentParser(description="OTA server in python")
75       parser.add_argument("-db", "--dbname", help="Database file name (path included)", default=
     PATH_DATABASE_DEFAULT)
76       parser.add_argument("-pbf", "--pathbinaryfiles", help="Binary files path", default=
     PATH_BINARY_FILE_DEFAULT)
77       parser.add_argument("-ho", "--host", help="Host IP", default=HOST_DEFAULT)
78       parser.add_argument("-p", "--port", help="Port to establish communication", default=PORT_DEFAULT)
79       parser.add_argument("-lp", "--logpath", help="Path to save logging", default=LOG_PATH_DEFAULT)
80       args = parser.parse_args()
81       return args
82
```

Here is the caller graph for this function:

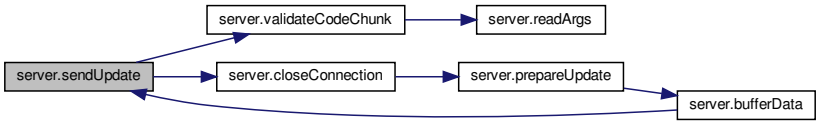


6.5.1.11 sendUpdate()

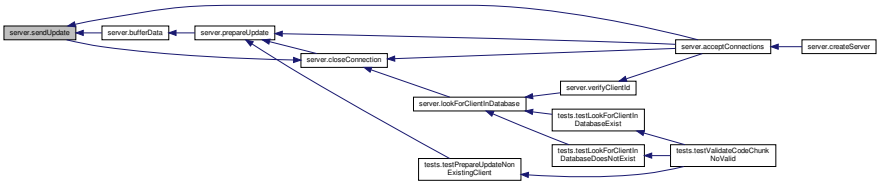
```
def server.sendUpdate (
              connection,
              address,
              mcuid,
              clientToUpdate,
              sock,
              logger )
```

```
247 def sendUpdate(connection, address, mcuid, clientToUpdate, sock, logger):
248      sock.settimeout(TIMEOUT)
249      for codechunk in clientToUpdate[0]['codelines']:
250           if (validateCodeChunk(codechunk) == SUCCESSFUL):
251                logger.info("Data from server {}".format(codechunk))
252                try:
253                     connection.send(codechunk.encode())
254                     receivedData = connection.recv(1024)
255                     logger.info("Data from client: {}".format(receivedData))
256                     if (receivedData == ackClient):
257                          continue
258                     else:
259                          closeConnection()
260                          return TIMEOUT_CONNECTION
261                except socket.timeout as e:
262                          logger.error("Timeout exceed {}".format(e))
263                except socket.error as e:
264                     logger.error("Error receiving data: {}".format(e))
265
266           else:
267                continue
268      logger.info("Update finished")
269      closeConnection(connection, logger)
270      sock.setblocking(0)
271      return SUCCESSFUL
272
```

Here is the call graph for this function:



Here is the caller graph for this function:

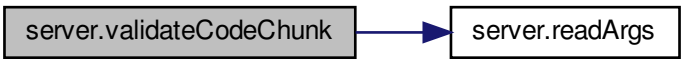


6.5.1.12 validateCodeChunk()

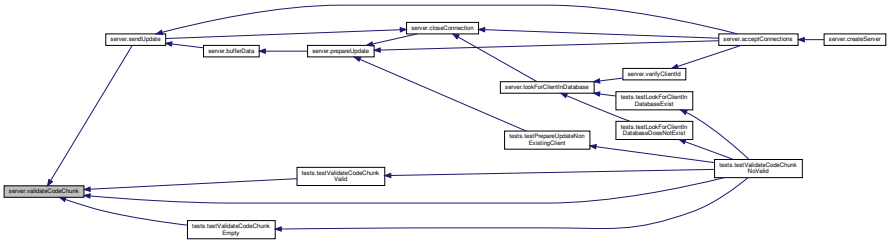
```
def server.validateCodeChunk (
              codechunk )
```

```
273 def  validateCodeChunk(codechunk):
274      match1 = re.search(r'^S1+[0-9A-F]+\w|^S2+[0-9A-F]+\w|^S3+[0-9A-F]+\w', codechunk)
275      match2 = re.search(r'^@([0-9A-F]{1,2})##$', codechunk)
276      if match1 or match2:
277           return SUCCESSFUL
278      else:
279           return INVALID_CODE_CHUNK
280
```

Here is the call graph for this function:



Here is the caller graph for this function:



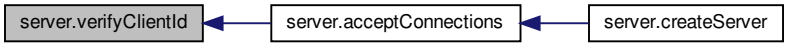
6.5.1.13 verifyClientId()

```
def server.verifyClientId (
              connection,
              mcuid,
              logger )
```

```
167 def verifyClientId(connection, mcuid, logger):
168
169      clientInfo = []
170      statusClient, clientInfo = lookForClientInDatabase(mcuid, logger)
171      if(statusClient== SUCCESSFUL):
172           logger.info("Client Info {}".format(clientInfo))
173           logger.info("Client {} exists in database".format(mcuid))
174           return SUCCESSFUL, clientInfo
175      else:
176           logger.error("Client {} does not exist in database".format(mcuid))
177           return CLIENT_UNVERIFIED, clientInfo
178
```

Here is the call graph for this function:

Here is the caller graph for this function:

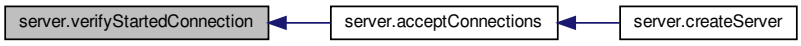


6.5.1.14 verifyStartedConnection()

```
def server.verifyStartedConnection (
              connections,
              connection )
```

```
150 def verifyStartedConnection(connections, connection):
151      for conn in connections:
152           if(conn == connection):
153                return ALREADY_STARTED_CONNECTION
154      connections.append(connection)
155      return NOT_STARTED_CONNECTION
156
```

Here is the caller graph for this function:



6.5.2 Variable Documentation

6.5.2.1 ackClient

```
string server.ackClient = b'@'
```

6.5.2.2 allowedUpdate

```
string server.allowedUpdate = b'@4#'
```

6.5.2.3 ALREADY_STARTED_CONNECTION

```
int server.ALREADY_STARTED_CONNECTION = 4
```

6.5.2.4 args

```
server.args = readArgs()
```

6.5.2.5 bannedUpdate

```
string server.bannedUpdate = b'@0#'
```

6.5.2.6 BUFFERING_CODE_INCOMPLETE

```
int server.BUFFERING_CODE_INCOMPLETE = 9
```

6.5.2.7 bufferSizeFile

```
int server.bufferSizeFile = 300
```

6.5.2.8 CLIENT_UNVERIFIED

```
int server.CLIENT_UNVERIFIED = 1
```

6.5.2.9 clientaddr

```
server.clientaddr
```

6.5.2.10 clientsList

```
list server.clientsList = []
```

6.5.2.11 clientsock

```
server.clientsock
```

6.5.2.12 connectedClients

```
int server.connectedClients = 0
```

6.5.2.13 connectionsList

```
list server.connectionsList = []
```

6.5.2.14 connectSocket

```
server.connectSocket
```

6.5.2.15 data

```
server.data = readableSocket.recv(1024)
```

6.5.2.16 databaseName

```
def server.databaseName = ""
```

6.5.2.17 DEBUG_ON

```
int server.DEBUG_ON = 1
```

Enable 1 Disable 0 Debug.

6.5.2.18 exceptional

```
server.exceptional
```

6.5.2.19 excepts

```
list server.excepts = []
```

6.5.2.20 host

```
string server.host = ""
```

6.5.2.21 HOST_DEFAULT

```
string server.HOST_DEFAULT = ''
```

6.5.2.22 inputs

```
list server.inputs = []
```

6.5.2.23 INVALID_CODE_CHUNK

```
int server.INVALID_CODE_CHUNK = 8
```

6.5.2.24 LOG_FILENAME_DEFAULT

```
string server.LOG_FILENAME_DEFAULT = "/otaserver.log"
```

6.5.2.25 LOG_LEVEL

```
server.LOG_LEVEL = logging.INFO
```

6.5.2.26 LOG_PATH_DEFAULT

```
string server.LOG_PATH_DEFAULT = "/tmp"
```

Defaults files to log.

6.5.2.27 logFile

```
string server.logFile = logPath + LOG_FILENAME_DEFAULT
```

6.5.2.28 logger

```
server.logger = configureLogger(logFile)
```

6.5.2.29 logPath

```
def server.logPath = ""
```

6.5.2.30 MAX_OPEN_CONNECTIONS

```
int server.MAX_OPEN_CONNECTIONS = 5
```

6.5.2.31 maxBytes

```
int server.maxBytes = 1024
```

6.5.2.32 MISSED_FILENAME

```
int server.MISSED_FILENAME = 2
```

6.5.2.33 NOT_STARTED_CONNECTION

```
int server.NOT_STARTED_CONNECTION = 4
```

6.5.2.34 openConnections

```
list server.openConnections = []
```

6.5.2.35 outputs

```
list server.outputs = []
```

6.5.2.36 PATH_BINARY_FILE_DEFAULT

```
string server.PATH_BINARY_FILE_DEFAULT = "/root/fota/otaserver/bin"
```

6.5.2.37 PATH_DATABASE_DEFAULT

```
string server.PATH_DATABASE_DEFAULT = "/root/fota/otaserver/database/devices2update.txt"
```

6.5.2.38 pathBinaryFiles

```
def server.pathBinaryFiles = ""
```

6.5.2.39 port

```
int server.port = ""
```

6.5.2.40 PORT_DEFAULT

```
int server.PORT_DEFAULT = 4000
```

6.5.2.41 readable

```
server.readable
```

6.5.2.42 READY_TO_UPDATE

```
int server.READY_TO_UPDATE = 10
```

6.5.2.43 serversocket

```
server.serversocket = socket.socket (socket.AF_INET, socket.SOCK_STREAM)
```

6.5.2.44 sock

```
server.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

6.5.2.45 SUCCESSFUL

```
int server.SUCCESSFUL = 0
```

6.5.2.46 target

```
server.target
```

6.5.2.47 TIMEOUT

```
int server.TIMEOUT = 30
```

6.5.2.48 TIMEOUT_CONNECTION

```
int server.TIMEOUT_CONNECTION = 7
```

6.5.2.49 UNABLE_BUFFERING_CODE

```
int server.UNABLE_BUFFERING_CODE = 5
```

6.5.2.50 UNFORMATTED_MCUID

```
int server.UNFORMATTED_MCUID = 3
```

6.5.2.51 UNPROPER_FILE_FORMAT

```
int server.UNPROPER_FILE_FORMAT = 6
```

6.5.2.52 writable

```
server.writable
```

6.6 Server.py Namespace Reference

Create a server and enable REST API for firmware updates.

6.6.1 Detailed Description

Create a server and enable REST API for firmware updates.

6.7 tests Namespace Reference

Functions

- def testLookForClientInDatabaseExist ()
- def testLookForClientInDatabaseDoesNotExist ()
- def testPrepareUpdateNonExistingClient ()
- def testValidateCodeChunkEmpty ()
- def testValidateCodeChunkValid ()
- def testValidateCodeChunkNoValid ()

6.7.1 Function Documentation

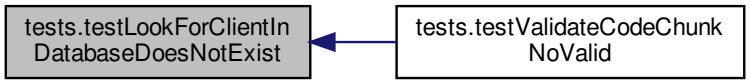
6.7.1.1 testLookForClientInDatabaseDoesNotExist()

```
def tests.testLookForClientInDatabaseDoesNotExist ( )
```

```
13 def testLookForClientInDatabaseDoesNotExist():
14      mcuid = "00000000000000000000000000"
15      clientInfo = []
16      if (server.lookForClientInDatabase(mcuid, clientInfo) !=
      server.SUCCESSFUL):
17          server.printDebugL1("Successful test")
18      else:
19          server.printDebugL1("Unsuccessfull test")
20
```

Here is the call graph for this function:

Here is the caller graph for this function:



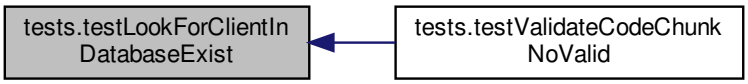
6.7.1.2 testLookForClientInDatabaseExist()

```
def tests.testLookForClientInDatabaseExist ( )
```

```
5 def testLookForClientInDatabaseExist():
6      mcuid = "00011000235931109659551 41"
7      clientInfo = []
8      if (server.lookForClientInDatabase(mcuid, clientInfo) ==
      server.SUCCESSFUL):
9          server.printDebugL1("Successful test")
10      else:
11          server.printDebugL1("Unsuccessfull test")
12
```

Here is the call graph for this function:

Here is the caller graph for this function:



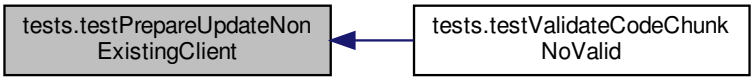
6.7.1.3 testPrepareUpdateNonExistingClient()

```
def tests.testPrepareUpdateNonExistingClient ( )
```

```
21 def testPrepareUpdateNonExistingClient():
22      clientInfo = ['00011000235931109659551 41','AN2295_TWR_K60.S19','16']
23      clientsList= []
24      if (server.prepareUpdate(clientInfo, clientsList) == server.SUCCESSFUL):
25           server.printDebugL1("{} Successful test".format(testPrepareUpdateNonExistingClient.__name__))
26      else:
27           server.printDebugL1("{} Unsuccessful test".format(testPrepareUpdateNonExistingClient.__name__))
28
```

Here is the call graph for this function:

Here is the caller graph for this function:



6.7.1.4 testValidateCodeChunkEmpty()

```
def tests.testValidateCodeChunkEmpty ( )
```

```
29 def testValidateCodeChunkEmpty():
30      codechunk = ""
31      if (server.validateCodeChunk(codechunk) == server.INVALID_CODE_CHUNK):
32           server.printDebugL1("ChunkEmpty Successful test")
33      else:
34           server.printDebugL1("ChunkEmpty Unsuccessful test")
35
```

Here is the call graph for this function:

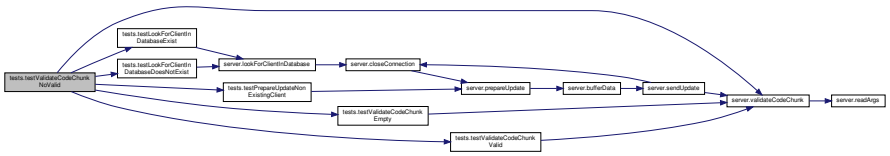
Here is the caller graph for this function:

6.7.1.5 testValidateCodeChunkNoValid()

```
def tests.testValidateCodeChunkNoValid ( )
```

```
43 def testValidateCodeChunkNoValid():
44      codechunk = "S786969"
45      if (server.validateCodeChunk(codechunk) == server.INVALID_CODE_CHUNK):
46           server.printDebugL1("ChunkNoValid Successful test")
47      else:
48           server.printDebugL1("ChunkNoValid Unsuccessful test")
49
```

Here is the call graph for this function:

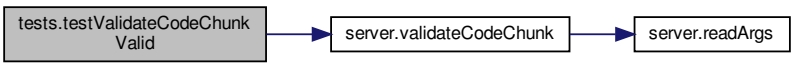


6.7.1.6 testValidateCodeChunkValid()

```
def tests.testValidateCodeChunkValid ( )
```

```
36 def testValidateCodeChunkValid():
37      codechunk = "S197AAKA9779"
38      if (server.validateCodeChunk(codechunk) == server.SUCCESSFUL):
39           server.printDebugL1("ChunkValid Successful test")
40      else:
41           server.printDebugL1("ChunkValid Unsuccessful test")
42
```

Here is the call graph for this function:



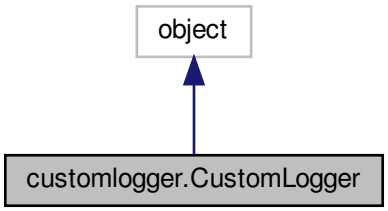
Here is the caller graph for this function:

Chapter 7

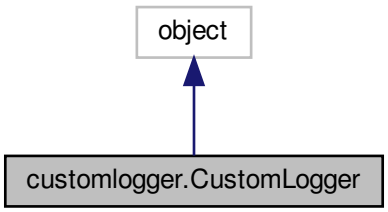
Class Documentation

7.1 customlogger.CustomLogger Class Reference

Inheritance diagram for customlogger.CustomLogger:



Collaboration diagram for customlogger.CustomLogger:



Public Member Functions

- def __init__ (self, logger, level)
- def write (self, message)

Public Attributes

- logger
- level

7.1.1 Constructor & Destructor Documentation

7.1.1.1 __init__()

```
def customlogger.CustomLogger.__init__ (
               self,
               logger,
               level )
```

```
9      def __init__(self, logger, level):
10           #Needs a logger and a logger level
11           self.logger = logger
12           self.level = level
13
```

7.1.2 Member Function Documentation

7.1.2.1 write()

```
def customlogger.CustomLogger.write (
               self,
               message )
```

```
14      def write(self, message):
15           #Only log if there is a message (not just a new line)
16           if message.rstrip() != "":
17                self.logger.log(self.level, message.rstrip())
18
```

7.1.3 Member Data Documentation

7.1.3.1 level

```
customlogger.CustomLogger.level
```

7.1.3.2 logger

```
customlogger.CustomLogger.logger
```

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

- customlogger.py

Chapter 8

File Documentation

8.1 customlogger.py File Reference

Classes

- class customlogger.CustomLogger

Namespaces

- customlogger

8.2 README.md File Reference

8.3 server.py File Reference

Namespaces

- server
- Server.py

  Create a server and enable REST API for firmware updates.

Functions

- def server.readArgs ()
- def server.configureLogger (logFile)
- def server.createServer (connectionsList, clientsList)
- def server.acceptConnections (connectionsList, clientsList, sock, logger)
- def server.verifyStartedConnection (connections, connection)
- def server.getMCUID (connection, address, logger)
- def server.verifyClientId (connection, mcuid, logger)
- def server.lookForClientInDatabase (mcuid, logger)
- def server.closeConnection (connection, logger)
- def server.prepareUpdate (clientInfo, clientsList, logger)
- def server.bufferData (filename, binaryFileLines, logger)
- def server.sendUpdate (connection, address, mcuid, clientToUpdate, sock, logger)
- def server.validateCodeChunk (codechunk)

Variables

- int server.DEBUG_ON = 1

  Enable 1 Disable 0 Debug.
- string server.LOG_PATH_DEFAULT = "/tmp"

  Defaults files to log.
- string server.LOG_FILENAME_DEFAULT = "/otaserver.log"
- server.LOG_LEVEL = logging.INFO
- int server.SUCCESSFUL = 0
- int server.CLIENT_UNVERIFIED = 1
- int server.MISSED_FILENAME = 2
- int server.UNFORMATTED_MCUID = 3
- int server.ALREADY_STARTED_CONNECTION = 4
- int server.NOT_STARTED_CONNECTION = 4
- int server.UNABLE_BUFFERING_CODE = 5
- int server.UNPROPER_FILE_FORMAT = 6
- int server.TIMEOUT_CONNECTION = 7
- int server.INVALID_CODE_CHUNK = 8
- int server.BUFFERING_CODE_INCOMPLETE = 9
- int server.READY_TO_UPDATE = 10
- int server.TIMEOUT = 30
- string server.HOST_DEFAULT = ''
- int server.PORT_DEFAULT = 4000
- string server.PATH_BINARY_FILE_DEFAULT = "/root/fota/otaserver/bin"
- string server.PATH_DATABASE_DEFAULT = "/root/fota/otaserver/database/devices2update.txt"
- string server.allowedUpdate = b'@4#'
- string server.bannedUpdate = b'@0#'
- string server.ackClient = b'@'
- int server.bufferSizeFile = 300
- int server.maxBytes = 1024
- string server.pathBinaryFiles = ""
- string server.databaseName = ""
- string server.host = ""
- string server.port = ""
- string server.logPath = ""
- list server.inputs = [ ]
- def server.args = readArgs()
- string server.logFile = logPath + LOG_FILENAME_DEFAULT
- def server.logger = configureLogger(logFile)
- list server.connectionsList = [ ]
- list server.clientsList = [ ]

8.4 tests/parallelize/multiprocessing/server.py File Reference

Namespaces

- server

Functions

- def server.connectSocket (socket)

Variables

- int server.MAX_OPEN_CONNECTIONS = 5
- int server.connectedClients = 0
- list server.openConnections = [ ]
- server.serversocket = socket.socket (socket.AF_INET, socket.SOCK_STREAM)
- server.target
- server.connectSocket

8.5 tests/parallelize/selectApproach/server.py File Reference

Namespaces

- server

Variables

- server.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
- list server.outputs = [ ]
- list server.excepts = [ ]
- server.readable
- server.writable
- server.exceptional
- server.clientsock
- server.clientaddr
- server.data = readableSocket.recv(1024)

8.6 tests/client.py File Reference

Namespaces

- client

Functions

- def client.verifyFinishedUpdate (data)

Variables

- string client.ackClient = b"@"
- string client.approvedUpdate = b"@4#"
- string client.HOST = '209.97.145.137'
- int client.PORT = 4000
- list client.binaryCode = [ ]
- client.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
- client.data = s.recv(1024)
- def client.finishedUpdate = verifyFinishedUpdate(data)

8.7 tests/parallelize/selectApproach/client.py File Reference

Namespaces

- client

Variables

- int client.TIMEOUT = 5
- string client.host = '127.0.0.1'
- int client.port = 10000
- client.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
- client.buf = sock.recv(1024)

8.8 tests/NoClient.py File Reference

Namespaces

- NoClient

Functions

- def NoClient.verifyFinishedUpdate (data)

Variables

- string NoClient.ackClient = b"@"
- string NoClient.approvedUpdate = b"@4#"
- string NoClient.HOST = '209.97.145.137'
- int NoClient.PORT = 4000
- list NoClient.binaryCode = [ ]
- NoClient.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
- NoClient.data = s.recv(1024)
- def NoClient.finishedUpdate = verifyFinishedUpdate(data)

8.9 tests/tests.py File Reference

Namespaces

- tests

Functions

- def tests.testLookForClientInDatabaseExist ()
- def tests.testLookForClientInDatabaseDoesNotExist ()
- def tests.testPrepareUpdateNonExistingClient ()
- def tests.testValidateCodeChunkEmpty ()
- def tests.testValidateCodeChunkValid ()
- def tests.testValidateCodeChunkNoValid ()

Index