目录

[File name: main.py 2](#_Toc80105520)

[File name: \_\_init\_\_.py 3](#_Toc80105521)

[File name: \_busstop.py 3](#_Toc80105522)

[File name: \_calibrator.py 5](#_Toc80105523)

[File name: \_chargingstation.py 8](#_Toc80105524)

[File name: \_edge.py 10](#_Toc80105525)

[File name: \_gui.py 14](#_Toc80105526)

[File name: \_inductionloop.py 17](#_Toc80105527)

[File name: \_junction.py 20](#_Toc80105528)

[File name: \_lane.py 22](#_Toc80105529)

[File name: \_lanearea.py 27](#_Toc80105530)

[File name: \_meandata.py 29](#_Toc80105531)

[File name: \_multientryexit.py 31](#_Toc80105532)

[File name: \_overheadwire.py 33](#_Toc80105533)

[File name: \_parkingarea.py 35](#_Toc80105534)

[File name: \_person.py 37](#_Toc80105535)

[File name: \_poi.py 44](#_Toc80105536)

[File name: \_polygon.py 48](#_Toc80105537)

[File name: rerouter.py 51](#_Toc80105538)

[File name: vehicle.py 51](#_Toc80105539)

[File name: \_trafficlight.py 68](#_Toc80105540)

[File name: \_person.py 70](#_Toc80105541)

[File name: \_edge.py 75](#_Toc80105542)

[File name: domain.py 76](#_Toc80105543)

[File name: connection.py 78](#_Toc80105544)

[File name: exceptions.py 79](#_Toc80105545)

[File name: storage.py 81](#_Toc80105546)

# File name: main.py

***Description:***

This file lists some functions that connect SUMO with TraCI interface by introducing the variables of SUMO into TraCI and establishing a connection to TraCI-Server. Moreover, some functions start a sumo server using cmd, establish a connection to it and store it under the given label or close a sumo server; also check SUMO library state and GUI states and load and use simulation.

***Functions:***

1. \_STEPS2TIME
2. setConnectHook
3. \_addTracing
4. connect
5. init
6. start
7. \_startTracing
8. isLibsumo
9. isLibtraci
10. hasGUI
11. load
12. isLoaded
13. simulationStep
14. addStepListener
15. removeStepListener
16. getVersion
17. setOrder
18. close
19. switch
20. getLabel
21. getConnection
22. setLegacyGetLeader

# File name: \_\_init\_\_.py

***Description:***

The pure python version needs to be the first variant to help IDEs finding the docstrings.

# File name: \_busstop.py

***Description:***

This file defines the “BusStopDomain” class which contains several methods for objects of busStopDomain to implement.

***Class:*** BusStopDomain(Domain)

The parameter “Domain” means that “BusStopDomain” class inherits from “Domain” class.

***Methods:***

1. \_\_init\_\_(self)(inherited from domain.Domain. \_\_init\_\_(self)

Domain.\_\_init\_\_(self, "busstop", tc.CMD\_GET\_BUSSTOP\_VARIABLE, tc.CMD\_SET\_BUSSTOP\_VARIABLE, t c.CMD\_SUBSCRIBE\_BUSSTOP\_VARIABLE, tc.RESPONSE\_SUBSCRIBE\_BUSSTOP\_VARIABLE,  
 tc.CMD\_SUBSCRIBE\_BUSSTOP\_CONTEXT, tc.RESPONSE\_SUBSCRIBE\_BUSSTOP\_CONTEXT)

*“tc” is traci.constants*

* 1. self: an instance of class BusStopDomain.
  2. “busstop”: The name of domain is “busstop”.
  3. tc.CMD\_GET\_BUSSTOP\_VARIABLE *:* A commond that gets busstop variable.

CMD\_GET\_BUSSTOP\_VARIABLE = 0xaf

* 1. tc.CMD\_SET\_BUSSTOP\_VARIABLE: A command sets busstop variable, not used yet.

CMD\_SET\_BUSSTOP\_VARIABLE = 0xcf

* 1. tc.CMD\_SUBSCRIBE\_BUSSTOP\_VARIABLE: A command subscribes busstop variable.

CMD\_SUBSCRIBE\_BUSSTOP\_VARIABLE = 0xdf

* 1. tc.RESPONSE\_SUBSCRIBE\_BUSSTOP\_VARIABLE: A response that subscribe busstop variable.

RESPONSE\_SUBSCRIBE\_BUSSTOP\_VARIABLE = 0xef

* 1. tc.CMD\_SUBSCRIBE\_BUSSTOP\_CONTEXT: A command subscribes busstop context.

CMD\_SUBSCRIBE\_BUSSTOP\_CONTEXT = 0x8f

* 1. tc.RESPONSE\_SUBSCRIBE\_BUSSTOP\_CONTEXT: The response that subscribes busstop context.

RESPONSE\_SUBSCRIBE\_BUSSTOP\_CONTEXT = 0x9f

1. getLaneID(self, stopID):

Returns the lane of this calibrator (if it applies to a single lane).

1. getStartPos(self, stopID):

The starting position of the stop along the lane measured in meter.

1. getEndPos(self, stopID):

The end position of the stop along the lane measured in m.

1. getName(self, stopID):

Returns the name of this stop.

1. getVehicleCount(self, stopID):

Get the total number of vehicles stopped at the named bus stop.

1. getVehicleIDs(self, stopID):

Get the IDs of vehicles stopped at the named bus stop.

1. getPersonCount(self, stopID):

Get the total number of waiting persons at the named bus stop.

1. getPersonIDs(self, stopID)

Get the IDs of waiting persons at the named bus stop.

# File name: \_calibrator.py

***Description:***

The calibrator is used to calibrating traffic flow of a base demand scenario by using local flow measurements. This file lists multiple methods that calibrator objects can call.

***Class:*** CalibratorDomain(Domain)

The parameter “Domain” means that “CalibratorDomain” class inherits from “Domain” class.

***Methods:***

1. \_\_init\_\_(self)

Domain.\_\_init\_\_(self, "calibrator", tc.CMD\_GET\_CALIBRATOR\_VARIABLE, tc.CMD\_SET\_CALIBRATOR\_VARIABLE,

tc.CMD\_SUBSCRIBE\_CALIBRATOR\_VARIABLE, tc.RESPONSE\_SUBSCRIBE\_CALIBRATOR\_VARIABLE,

t c.CMD\_SUBSCRIBE\_CALIBRATOR\_CONTEXT, tc.RESPONSE\_SUBSCRIBE\_CALIBRATOR\_CONTEXT)

*“tc” is traci.constants*

* 1. self: an instance of class CalibratorDomain.
  2. "calibrator": The name of domain is "calibrator".
  3. tc.CMD\_GET\_CALIBRATOR\_VARIABLE *:* A commond that gets calibrator variable.

CMD\_GET\_CALIBRATOR\_VARIABLE = 0x27

* 1. tc. CMD\_SET\_CALIBRATOR\_VARIABLE: A command sets calibrator variable.

CMD\_SET\_CALIBRATOR\_VARIABLE = 0x47

* 1. tc.CMD\_SUBSCRIBE\_CALIBRATOR \_VARIABLE: A command subscribes calibrator variable.

CMD\_SUBSCRIBE\_CALIBRATOR\_VARIABLE = 0x57

* 1. RESPONSE\_SUBSCRIBE\_CALIBRATOR\_VARIABLE: A response that subscribe calibrator variable.

RESPONSE\_SUBSCRIBE\_CALIBRATOR\_VARIABLE = 0x67

* 1. CMD\_SUBSCRIBE\_CALIBRATOR\_CONTEXT: A command subscribes calibrator context.

CMD\_SUBSCRIBE\_CALIBRATOR\_CONTEXT = 0x07

* 1. tc.RESPONSE \_ SUBSCRIBE \_ CALIBRATOR\_CONTEXT: The response that subscribes calibrator context.

RESPONSE\_SUBSCRIBE\_CALIBRATOR\_CONTEXT = 0x17

1. getEdgeID(self, calibratorID):

Returns the edge of this calibrator.

1. getLaneID(self, calibratorID):

Returns the lane of this calibrator (if it applies to a single lane).

1. getVehsPerHour(self, calibratorID):

Returns the number of vehicles per hour in the current calibration interval.

1. getSpeed(self, calibratorID):

Returns the target speed of the current calibration interval.

1. getTypeID(self, calibratorID):

Returns the type id for the current calibration interval.

1. getBegin(self, calibratorID):

Returns the begin time of the current calibration interval.

1. getEnd(self, calibratorID):

Returns the end time of the current calibration interval.

1. getRouteID(self, calibratorID):

Returns the route id for the current calibration interval.

1. getRouteProbeID(self, calibratorID):

Returns the routeProbe id for this calibrator.

1. getVTypes(self, calibratorID):

Returns a list of all types to which the calibrator applies (in a type filter is active).

1. getPassed(self, calibratorID):

Returns the number of passed vehicles in the current calibration interval.

1. getInserted(self, calibratorID):

Returns the number of passed vehicles in the current calibration interval.

1. getRemoved(self, calibratorID):

Returns the number of removed vehicles in the current calibration interval.

1. setFlow(self, calibratorID, begin, end, vehsPerHour, speed, typeID,  
    routeID, departLane="first", departSpeed="max")

Update or add a calibrator interval.

# File name: \_chargingstation.py

***Description:***

This file defines the “ChargingStationDomain” class which contains several methods for objects of ChargingStationDomain to implement.

***Class:*** ChargingStationDomain(Domain)

The parameter “Domain” means that “ChargingStationDomain” class inherits from “Domain” class.

***Methods:***

1. \_\_init\_\_(self)

Domain.\_\_init\_\_(self, "chargingstation",

tc.CMD\_GET\_CHARGINGSTATION\_VARIABLE, tc.CMD\_SET\_CHARGINGSTATION\_VARIABLE, tc.CMD\_SUBSCRIBE\_CHARGINGSTATION\_VARIABLE, tc.RESPONSE\_SUBSCRIBE\_CHARGINGSTATION\_VARIABLE, tc.CMD\_SUBSCRIBE\_CHARGINGSTATION\_CONTEXT, tc.RESPONSE\_SUBSCRIBE\_CHARGINGSTATION\_CONTEXT)

*“tc” is traci.constants*

* 1. self: an instance of class ChargingStationDomain.
  2. "chargingstation": The name of domain is "chargingstation".
  3. tc.CMD\_GET\_CHARGINGSTATION\_VARIABLE *:* A commond that gets chargingstation variable.

CMD\_GET\_CHARGINGSTATION\_VARIABLE = 0x25

* 1. tc. CMD\_SET\_ CHARGINGSTATION \_VARIABLE: A command sets chargingstation variable.

CMD\_SET\_CHARGINGSTATION\_VARIABLE = 0x45

* 1. tc.CMD\_SUBSCRIBE\_ CHARGINGSTATION \_VARIABLE: A command subscribes chargingstation variable.

CMD\_SUBSCRIBE\_CHARGINGSTATION\_VARIABLE = 0x55

* 1. RESPONSE\_SUBSCRIBE\_ CHARGINGSTATION \_VARIABLE: A response that subscribe chargingstation variable.

RESPONSE\_SUBSCRIBE\_CHARGINGSTATION\_VARIABLE = 0x65

* 1. CMD\_SUBSCRIBE\_ CHARGINGSTATION \_CONTEXT: A command subscribes chargingstation context.

CMD\_SUBSCRIBE\_CHARGINGSTATION\_CONTEXT = 0x05

* 1. tc.RESPONSE\_SUBSCRIBE\_CHARGINGSTATION \_CONTEXT: The response that subscribes chargingstation context.

RESPONSE\_SUBSCRIBE\_CHARGINGSTATION\_CONTEXT = 0x15

1. getLaneID(self, stopID):

Returns the lane of this calibrator (if it applies to a single lane).

1. getStartPos(self, stopID) :

The starting position of the stop along the lane measured in meter.

1. getEndPos(self, stopID):

The end position of the stop along the lane measured in meter.

1. getName(self, stopID):

Returns the name of this stop.

1. getVehicleCount(self, stopID):

Get the total number of vehicles stopped at the named charging station.

1. getVehicleIDs(self, stopID)

Get the IDs of vehicles stopped at the named charging station.

# File name: \_edge.py

***Description:***

This file defines the “EdgeDomain” class which contains several methods for objects of EdgeDomain to implement.

***Class:*** EdgeDomain(Domain)

The parameter “Domain” means that “EdgeDomain” class inherits from “Domain” class.

***Methods:***

1. \_\_init\_\_(self)

Domain.\_\_init\_\_(self, "edge", tc.CMD\_GET\_EDGE\_VARIABLE, tc.CMD\_SET\_EDGE\_VARIABLE,

tc.CMD\_SUBSCRIBE\_EDGE\_VARIABLE, tc.RESPONSE\_SUBSCRIBE\_EDGE\_VARIABLE,

tc.CMD\_SUBSCRIBE\_EDGE\_CONTEXT, tc.RESPONSE\_SUBSCRIBE\_EDGE\_CONTEXT, subscriptionDefault=(tc.LAST\_STEP\_VEHICLE\_NUMBER,))

*“tc” is traci.constants*

* 1. self: an instance of class EdgeDomain.
  2. " edge ": The name of domain is " edge ".
  3. tc.CMD\_GET\_EDGE \_VARIABLE *:* A commond that gets edge variable.

CMD\_GET\_EDGE\_VARIABLE = 0xaa

* 1. tc. CMD\_SET\_ EDGE \_VARIABLE: A command sets chargingstation variable.

CMD\_SET\_EDGE\_VARIABLE = 0xca

* 1. tc.CMD\_SUBSCRIBE\_ EDGE \_VARIABLE: A command subscribes edge variable.

CMD\_SUBSCRIBE\_EDGE\_VARIABLE = 0xda

* 1. tc.RESPONSE\_SUBSCRIBE\_ EDGE \_VARIABLE: A response that subscribe edge variable.

RESPONSE\_SUBSCRIBE\_EDGE\_VARIABLE = 0xea

* 1. tc.CMD\_SUBSCRIBE\_ EDGE \_CONTEXT: A command subscribes edge context.

CMD\_SUBSCRIBE\_EDGE\_CONTEXT = 0x8a

* 1. tc.RESPONSE\_SUBSCRIBE\_ EDGE\_CONTEXT: The response that subscribes edge context. RESPONSE\_SUBSCRIBE\_EDGE\_VARIABLE = 0xea
  2. subscriptionDefault=(tc.LAST\_STEP\_VEHICLE\_NUMBER,): Set subscription default is the vehicle number in last step.

LAST\_STEP\_VEHICLE\_NUMBER = 0x10

1. getAdaptedTraveltime(self, edgeID, time):

Returns the travel time value (in s) used for (re-)routing

which is valid on the edge at the given time.

1. getWaitingTime(self, edgeID):

Returns the sum of the waiting time of all vehicles currently on

that edge (see traci.vehicle.getWaitingTime).

1. getEffort(self, edgeID, time):

Returns the effort value used for (re-)routing

which is valid on the edge at the given time.

1. getCO2Emission(self, edgeID):

Returns the CO2 emission in mg for the last time step on the given edge.

1. getCOEmission(self, edgeID):

Returns the CO emission in mg for the last time step on the given edge.

1. getHCEmission(self, edgeID):

Returns the HC emission in mg for the last time step on the given edge.

1. getPMxEmission(self, edgeID):

Returns the particular matter emission in mg for the last time step on the given edge.

1. getNOxEmission(self, edgeID):

Returns the NOx emission in mg for the last time step on the given edge.

1. getFuelConsumption(self, edgeID):

Returns the fuel consumption in ml for the last time step on the given edge.

1. getNoiseEmission(self, edgeID):

Returns the noise emission in db for the last time step on the given edge.

1. getElectricityConsumption(self, edgeID):

Returns the electricity consumption in ml for the last time step.

1. getLastStepMeanSpeed(self, edgeID):

Returns the average speed in m/s for the last time step on the given edge.

1. getLastStepOccupancy(self, edgeID):

Returns the net occupancy (excluding inter-vehicle gaps) in % for the last time step on the given edge.

1. getLastStepLength(self, edgeID):

Returns the mean vehicle length in m for the last time step on the given edge.

1. getLaneNumber(self, edgeID):

Returns the number of lanes of this edge.

1. getStreetName(self, edgeID):

Returns the street name of this edge.

1. getTraveltime(self, edgeID):

Returns the estimated travel time in s for the last time step on the given edge.

1. getLastStepVehicleNumber(self, edgeID):

Returns the total number of vehicles for the last time step on the given edge.

1. getLastStepHaltingNumber(self, edgeID):

Returns the total number of halting vehicles for the last time step on the given edge.

A speed of less than 0.1 m/s is considered a halt.

1. getLastStepVehicleIDs(self, edgeID):

Returns the ids of the vehicles for the last time step on the given edge.

1. getLastStepPersonIDs(self, edgeID):

Returns the ids of the persons on the given edge during the last time step.

1. getPendingVehicles(self, edgeID):

Returns a list of all vehicle ids waiting for insertion on this edge (with depart delay).

1. adaptTraveltime(self, edgeID, time, begin=None, end=None):

Adapt the travel time value (in s) used for (re-)routing for the given edge.

When setting begin time and end time (in seconds), the changes only

apply to that time range. Otherwise they apply all the time.

Time:double

Begin:double

End:double

1. setEffort(self, edgeID, effort, begin=None, end=None):

Adapt the effort value used for (re-)routing for the given edge.

When setting begin time and end time (in seconds), the changes only

apply to that time range. Otherwise they apply all the time.

1. setAllowed(self, edgeID, allowedClasses):

Sets a list of allowed vehicle classes. Setting an empty list means all vehicles are allowed.

1. setDisallowed(self, edgeID, disallowedClasses):

Sets a list of disallowed vehicle classes.

disallowedClasses:list.:

1. setMaxSpeed(self, edgeID, speed)

Set a new maximum speed (in m/s) for all lanes of the edge.

# File name: \_gui.py

**Description:**

This file defines the “GuiDomain” class which contains several methods for objects of GuiDomain to implement.

***Class:*** GuiDomain(Domain)

The parameter “Domain” means that “GuiDomain” class inherits from “Domain” class.

***Class member:***

DEFAULT\_VIEW: the default view is string.

DEFAULT\_VIEW = **'View #0'**

***Methods:***

1. \_\_init\_\_(self)

Domain.\_\_init\_\_(self, "gui", tc.CMD\_GET\_GUI\_VARIABLE, tc.CMD\_SET\_GUI\_VARIABLE,

tc.CMD\_SUBSCRIBE\_GUI\_VARIABLE, tc.RESPONSE\_SUBSCRIBE\_GUI\_VARIABLE,

tc.CMD\_SUBSCRIBE\_GUI\_CONTEXT, tc.RESPONSE\_SUBSCRIBE\_GUI\_CONTEXT,

\_RETURN\_VALUE\_FUNC)

*“tc” is traci.constants*

* 1. self: an instance of class GuiDomain.
  2. " gui ": The name of domain is " gui ".
  3. tc.CMD\_GET\_ GUI \_VARIABLE *:* A commond that gets GUI variable.

CMD\_GET\_GUI\_VARIABLE = 0xac

* 1. tc.CMD\_SET\_ GUI \_VARIABLE: A command sets GUI variable.

CMD\_SET\_GUI\_VARIABLE = 0xcc

* 1. tc.CMD\_SUBSCRIBE\_ GUI \_VARIABLE: A command subscribes GUI variable.

CMD\_SUBSCRIBE\_GUI\_VARIABLE = 0xdc

* 1. tc.RESPONSE\_SUBSCRIBE\_ GUI \_VARIABLE: A response that subscribe GUI variable.

RESPONSE\_SUBSCRIBE\_GUI\_VARIABLE = 0xec

* 1. tc.CMD\_SUBSCRIBE\_ GUI \_CONTEXT: A command subscribes GUI context.

CMD\_SUBSCRIBE\_GUI\_CONTEXT = 0x8c

* 1. tc.RESPONSE\_SUBSCRIBE\_ GUI \_CONTEXT: The response that subscribes GUI context. RESPONSE\_SUBSCRIBE\_GUI\_CONTEXT = 0x9c

* 1. \_RETURN\_VALUE\_FUNC

1. getZoom(self, viewID=DEFAULT\_VIEW):

Returns the current zoom factor.

1. getOffset(self, viewID=DEFAULT\_VIEW):

Returns the x and y offset of the center of the current view.

1. getSchema(self, viewID=DEFAULT\_VIEW):

Returns the name of the current coloring scheme.

1. getBoundary(self, viewID=DEFAULT\_VIEW):

Returns the coordinates of the lower left and the upper right corner of the currently visible view.

1. setZoom(self, viewID, zoom):

Set the current zoom factor for the given view.

1. setOffset(self, viewID, x, y):

Set the current offset for the given view.

1. setSchema(self, viewID, schemeName):

Set the current coloring scheme for the given view.

1. setBoundary(self, viewID, xmin, ymin, xmax, ymax):

Sets the boundary of the visible network. If the window has a different aspect ratio than the given boundary, the view is expanded along one axis to meet the window aspect ratio and contain the given boundary.

1. screenshot(self, viewID, filename, width=-1, height=-1):

Save a screenshot for the given view to the given filename.

The fileformat is guessed from the extension, the available formats differ from platform to platform but should at least include ps, svg and pdf, on linux probably gif, png and jpg as well.

Width and height of the image can be given as optional parameters.

1. trackVehicle(self, viewID, vehID):

Start visually tracking the given vehicle on the given view.

1. hasView(self, viewID=DEFAULT\_VIEW):

Check whether the given view exists.

1. getTrackedVehicle(self, viewID=DEFAULT\_VIEW):

Returns the id of the currently tracked vehicle.

1. track(self, objID, viewID=DEFAULT\_VIEW):

Start visually tracking the given vehicle or person on the given view.

1. isSelected(self, objID, objType="vehicle"):

Return 1 if the object of the given type and id is select, 0 otherwise.

1. toggleSelection(self, objID, objType="vehicle")

Toggle selection status for the object of the given type and id.

# File name: \_inductionloop.py

**Description:**

This file defines the “InductionLoopDomain” class which contains several methods for objects of GuiDomain to call to gain information on the named induction loop.

***Class:*** InductionLoopDomain (Domain)

The parameter “Domain” means that “InductionLoopDomain” class inherits from “Domain” class.

***Methods:***

1. \_\_init\_\_(self)

Domain.\_\_init\_\_(self, "inductionloop", tc.CMD\_GET\_INDUCTIONLOOP\_VARIABLE, tc.CMD\_SET\_INDUCTIONLOOP\_VARIABLE, tc.CMD\_SUBSCRIBE\_INDUCTIONLOOP\_VARIABLE, tc.RESPONSE\_SUBSCRIBE\_INDUCTIONLOOP\_VARIABLE, tc.CMD\_SUBSCRIBE\_INDUCTIONLOOP\_CONTEXT, tc.RESPONSE\_SUBSCRIBE\_INDUCTIONLOOP\_CONTEXT,

\_RETURN\_VALUE\_FUNC, subscriptionDefault=(tc.LAST\_STEP\_VEHICLE\_NUMBER,))

*“tc” is traci.constants*

* 1. self: an instance of class InductionLoopDomain.
  2. " inductionloop ": The name of domain is " inductionloop ".
  3. tc.CMD\_GET\_ INDUCTIONLOOP \_VARIABLE *:* A commond that gets induction loop variable.

CMD\_GET\_INDUCTIONLOOP\_VARIABLE = 0xa0

* 1. tc.CMD\_SET\_ INDUCTIONLOOP \_VARIABLE: A command sets induction loop variable.

CMD\_SET\_INDUCTIONLOOP\_VARIABLE = 0xc0

* 1. tc.CMD\_SUBSCRIBE\_ INDUCTIONLOOP \_VARIABLE: A command subscribes induction loop variable.

CMD\_SUBSCRIBE\_INDUCTIONLOOP\_VARIABLE = 0xd0

* 1. tc.RESPONSE\_SUBSCRIBE\_ INDUCTIONLOOP \_VARIABLE: A response that subscribe induction loop variable.

RESPONSE\_SUBSCRIBE\_INDUCTIONLOOP\_VARIABLE = 0xe0

* 1. tc.CMD\_SUBSCRIBE\_ INDUCTIONLOOP \_CONTEXT: A command subscribes induction loop context.

CMD\_SUBSCRIBE\_INDUCTIONLOOP\_CONTEXT = 0x80

* 1. tc.RESPONSE\_SUBSCRIBE\_ INDUCTIONLOOP \_CONTEXT: The response that subscribes induction loop context.

RESPONSE\_SUBSCRIBE\_INDUCTIONLOOP\_CONTEXT = 0x90

* 1. \_RETURN\_VALUE\_FUNC
  2. subscriptionDefault=(tc.LAST\_STEP\_VEHICLE\_NUMBER,)

Set subscription default is the vehicle number in last step.

LAST\_STEP\_VEHICLE\_NUMBER = 0x10

1. getPosition(self, loopID):

Returns the position measured from the beginning of the lane in meters.

1. getLaneID(self, loopID):

Returns the id of the lane the loop is on.

1. getLastStepVehicleNumber(self, loopID):

Returns the number of vehicles that were on the named induction loop within the last simulation step.

1. getLastStepMeanSpeed(self, loopID):

Returns the mean speed in m/s of vehicles that were on the named induction loop within the last simulation step.

1. getLastStepVehicleIDs(self, loopID):

Returns the list of ids of vehicles that were on the named induction loop in the last simulation step.

1. getLastStepOccupancy(self, loopID):

Returns the percentage of time the detector was occupied by a vehicle.

1. getLastStepMeanLength(self, loopID):

Returns the mean length in m of vehicles which were on the detector in the last step.

1. getTimeSinceDetection(self, loopID):

Returns the time in s since last detection.

1. getVehicleData(self, loopID):

Returns a complex structure containing several information about vehicles which passed the detector.

# File name: \_junction.py

**Description:**

This file defines the “JunctionDomain” class which contains several methods for objects of JunctionDomain to call to gain information of the named junction.

***Class:*** JunctionDomain (Domain)

The parameter “Domain” means that “JunctionDomain” class inherits from “Domain” class.

***Methods:***

1. \_\_init\_\_(self)

Domain.\_\_init\_\_(self, "junction", tc.CMD\_GET\_JUNCTION\_VARIABLE, tc.CMD\_SET\_JUNCTION\_VARIABLE,

tc.CMD\_SUBSCRIBE\_JUNCTION\_VARIABLE, tc.RESPONSE\_SUBSCRIBE\_JUNCTION\_VARIABLE,

tc.CMD\_SUBSCRIBE\_JUNCTION\_CONTEXT, tc.RESPONSE\_SUBSCRIBE\_JUNCTION\_CONTEXT)

*“tc” is traci.constants*

* 1. self: an instance of class junctionDomain.
  2. " junction": The name of domain is " junction".
  3. tc.CMD\_GET\_ JUNCTION \_VARIABLE *:* A commond that gets junction variable.

CMD\_GET\_JUNCTION\_VARIABLE = 0xa9

* 1. tc.CMD\_SET\_ JUNCTION \_VARIABLE: A command sets junction variable.

CMD\_SET\_JUNCTION\_VARIABLE = 0xc9

* 1. tc.CMD\_SUBSCRIBE\_ JUNCTION \_VARIABLE: A command subscribes junction variable.

CMD\_SUBSCRIBE\_JUNCTION\_VARIABLE = 0xd9

* 1. tc.RESPONSE\_SUBSCRIBE\_ JUNCTION \_VARIABLE: A response that subscribe junction variable.

RESPONSE\_SUBSCRIBE\_JUNCTION\_VARIABLE = 0xe9

* 1. tc.CMD\_SUBSCRIBE\_ JUNCTION \_CONTEXT: A command subscribes junction context.

CMD\_SUBSCRIBE\_JUNCTION\_CONTEXT = 0x89

* 1. tc.RESPONSE\_SUBSCRIBE\_ JUNCTION \_CONTEXT: The response that subscribes junction context.

RESPONSE\_SUBSCRIBE\_JUNCTION\_CONTEXT = 0x99

1. getPosition(self, junctionID, includeZ=False):

Returns the coordinates of the center of the junction.

1. getShape(self, junctionID)：

List of 2 dimentional positions (cartesian) describing the geometry.

# File name: \_lane.py

**Description:**

This file defines the “LaneDomain” class which contains several methods for objects of LaneDomain to call to gain information of the named lane and to set some functions.

***Class:*** LaneDomain (Domain)

The parameter “Domain” means that “LaneDomain” class inherits from “Domain” class.

***Methods:***

1. \_\_init\_\_(self)

Domain.\_\_init\_\_(self, "lane", tc.CMD\_GET\_LANE\_VARIABLE, tc.CMD\_SET\_LANE\_VARIABLE,

tc.CMD\_SUBSCRIBE\_LANE\_VARIABLE, tc.RESPONSE\_SUBSCRIBE\_LANE\_VARIABLE,

tc.CMD\_SUBSCRIBE\_LANE\_CONTEXT, tc.RESPONSE\_SUBSCRIBE\_LANE\_CONTEXT,

\_RETURN\_VALUE\_FUNC, subscriptionDefault=(tc.LAST\_STEP\_VEHICLE\_NUMBER,))

*“tc” is traci.constants*

* 1. self: an instance of class LaneDomain.
  2. " lane ": The name of domain is " lane ".
  3. tc.CMD\_GET\_ LANE \_VARIABLE *:* A commond that gets lane variable.

CMD\_GET\_LANE\_VARIABLE = 0xa3

* 1. tc.CMD\_SET\_ LANE \_VARIABLE: A command sets lane variable.

CMD\_SET\_LANE\_VARIABLE = 0xc3

* 1. tc.CMD\_SUBSCRIBE\_ LANE \_VARIABLE: A command subscribes lane variable.

CMD\_SUBSCRIBE\_LANE\_VARIABLE = 0xd3

* 1. tc.RESPONSE\_SUBSCRIBE\_ LANE \_VARIABLE: A response that subscribe lane variable.

RESPONSE\_SUBSCRIBE\_LANE\_CONTEXT = 0x93

* 1. tc.CMD\_SUBSCRIBE\_ LANE \_CONTEXT: A command subscribes lane context.

CMD\_SUBSCRIBE\_LANE\_CONTEXT = 0x83

* 1. tc.RESPONSE\_SUBSCRIBE\_ LANE \_CONTEXT: The response that subscribes lane context.

RESPONSE\_SUBSCRIBE\_LANE\_CONTEXT = 0x93

* 1. \_RETURN\_VALUE\_FUNC
  2. subscriptionDefault=(tc.LAST\_STEP\_VEHICLE\_NUMBER,)

Set subscription default is the vehicle number in last step.

LAST\_STEP\_VEHICLE\_NUMBER = 0x10

1. getLength(self, laneID):

Returns the length in meter.

1. getMaxSpeed(self, laneID):

Returns the maximum allowed speed on the lane in m/s.

1. getWidth(self, laneID):

Returns the width of the lane in m.

1. getAllowed(self, laneID):

Returns a list of allowed vehicle classes. An empty list means all vehicles are allowed.

1. getDisallowed(self, laneID):

Returns a list of disallowed vehicle classes.

1. getLinkNumber(self, laneID):

Returns the number of connections to successive lanes.

1. getLinks(self, laneID, extended=True):

A list containing id of successor lane together with priority, open and foe

for each link.

if extended=True, each result tuple contains

(string approachedLane, bool hasPrio, bool isOpen, bool hasFoe,

string approachedInternal, string state, string direction, float length)

isOpen: whether a vehicle driving at the speed limit (minimum auf

incoming and outgoing lane) could safely pass the junction with

regard to approaching foes if it were to enter it in this step

(false for red traffic light).

Foe vehicles that are already on the junction are ignored!

hasPrio: whether the link is the main road at a priority junction or

currently has green light ('G')

hasFoe: whether any foe vehicles are approaching the junction or on the

junction that would interfere with passing it in the current time step.

1. getShape(self, laneID):

List of 2D positions (cartesian) describing the geometry.

1. getEdgeID(self, laneID):

Returns the id of the edge the lane belongs to.

1. getCO2Emission(self, laneID):

Returns the CO2 emission in mg for the last time step on the given lane.

1. getCOEmission(self, laneID):

Returns the CO emission in mg for the last time step on the given lane.

1. getHCEmission(self, laneID):

Returns the HC emission in mg for the last time step on the given lane.

1. getPMxEmission(self, laneID):

Returns the particular matter emission in mg for the last time step on the given lane.

1. getNOxEmission(self, laneID):

Returns the NOx emission in mg for the last time step on the given lane.

1. getFuelConsumption(self, laneID):

Returns the fuel consumption in ml for the last time step on the given lane.

1. getNoiseEmission(self, laneID):

Returns the noise emission in db for the last time step on the given lane.

1. getElectricityConsumption(self, laneID):

Returns the electricity consumption in ml for the last time step.

1. getLastStepMeanSpeed(self, laneID):

Returns the average speed in m/s for the last time step on the given lane.

1. getLastStepOccupancy(self, laneID):

Returns the occupancy in % for the last time step on the given lane.

1. getLastStepLength(self, laneID):

Returns the mean vehicle length in meter for the last time step on the given lane.

1. getWaitingTime(self, laneID):
2. getTraveltime(self, laneID):

Returns the estimated travel time in s for the last time step on the given lane.

1. getLastStepVehicleNumber(self, laneID):

Returns the total number of vehicles for the last time step on the given lane.

1. getLastStepHaltingNumber(self, laneID):

Returns the total number of halting vehicles for the last time step on the given lane. A speed of less than 0.1 m/s is considered a halt.

1. getLastStepVehicleIDs(self, laneID):

Returns the ids of the vehicles for the last time step on the given lane.

1. getFoes(self, laneID, toLaneID):

Returns the ids of incoming lanes that have right of way over the connection from laneID to toLaneID.

1. getInternalFoes(self, laneID):

Returns the ids of internal lanes that are in conflict with the given internal lane id.

1. getPendingVehicles(self, laneID):

Returns a list of all vehicle ids waiting for insertion on this lane (with depart delay).

1. setAllowed(self, laneID, allowedClasses):

Sets a list of allowed vehicle classes. Setting an empty list means all vehicles are allowed.

1. setDisallowed(self, laneID, disallowedClasses):

Sets a list of allowed vehicle classes. Setting an empty list means all vehicles are allowed.

1. setMaxSpeed(self, laneID, speed):

Sets a new maximum allowed speed on the lane in m/s.

1. setLength(self, laneID, length):

Sets the length of the lane in m.

# File name: \_lanearea.py

**Description:**

This file defines the “LaneAreaDomain” class which contains several methods for objects of LaneAreaDomain to call to gain information of the named detector and to set some functions.

***Class:*** LaneAreaDomain (Domain)

The parameter “Domain” means that “LaneAreaDomain” class inherits from “Domain” class.

***Methods:***

1. \_\_init\_\_(self, name="lanearea", deprecatedFor=None)

Domain.\_\_init\_\_(self, lanearea, tc.CMD\_GET\_LANEAREA\_VARIABLE, tc.CMD\_SET\_LANEAREA\_VARIABLE,

tc.CMD\_SUBSCRIBE\_LANEAREA\_VARIABLE, tc.RESPONSE\_SUBSCRIBE\_LANEAREA\_VARIABLE,

tc.CMD\_SUBSCRIBE\_LANEAREA\_CONTEXT, tc.RESPONSE\_SUBSCRIBE\_LANEAREA\_CONTEXT,

{}, deprecatedFor, subscriptionDefault=(tc.LAST\_STEP\_VEHICLE\_NUMBER,))

*“tc” is traci.constants*

* 1. self: an instance of class LaneAreaDomain.
  2. " lane ": The name of domain is " lanearea".
  3. tc.CMD\_GET\_ LANEAREA \_VARIABLE *:* A commond that gets lane area detector variable.

CMD\_GET\_LANEAREA\_VARIABLE = 0xad

* 1. tc.CMD\_SET\_ LANEAREA \_VARIABLE: A command sets lane area detector variable.

CMD\_SET\_LANEAREA\_VARIABLE = 0xcd

* 1. tc.CMD\_SUBSCRIBE\_ LANEAREA \_VARIABLE: A command subscribes lane area detector variable.

CMD\_SUBSCRIBE\_LANEAREA\_VARIABLE = 0xdd

* 1. tc.RESPONSE\_SUBSCRIBE\_ LANEAREA\_VARIABLE: A response that subscribe lane area detector variable.

RESPONSE\_SUBSCRIBE\_LANEAREA\_VARIABLE = 0xed

* 1. tc.CMD\_SUBSCRIBE\_ LANEAREA \_CONTEXT: A command subscribes lane area detector context.

CMD\_SUBSCRIBE\_LANEAREA\_CONTEXT = 0x8d

* 1. tc.RESPONSE\_SUBSCRIBE\_ LANEAREA \_CONTEXT: The response that subscribes lane area detector context.

RESPONSE\_SUBSCRIBE\_LANEAREA\_CONTEXT = 0x9d

* 1. \_RETURN\_VALUE\_FUNC is {}.
  2. subscriptionDefault=(tc.LAST\_STEP\_VEHICLE\_NUMBER,): Set subscription default is the vehicle number in last step.

LAST\_STEP\_VEHICLE\_NUMBER = 0x10

1. getJamLengthVehicle(self, detID):

Returns the jam length in vehicles within the last simulation step.

1. getJamLengthMeters(self, detID):

Returns the jam length in meters within the last simulation step.

1. getLastStepMeanSpeed(self, detID):

Returns the current mean speed in m/s of vehicles that were on the named e2.

1. getLastStepVehicleIDs(self, detID):

Returns the list of ids of vehicles that were on the named detector in the last simulation step.

1. getLastStepOccupancy(self, detID):

Returns the percentage of space the detector was occupied by a vehicle [%]

1. getPosition(self, detID):

Returns the starting position of the detector measured from the beginning of the lane in meters.

1. getLaneID(self, detID):

Returns the id of the lane the detector is on.

1. getLength(self, detID):

Returns the length of the detector.

1. getLastStepVehicleNumber(self, detID):

Returns the number of vehicles that were on the named detector within the last simulation step.

1. getLastStepHaltingNumber(self, detID)

Returns the number of vehicles which were halting during the last time step.

# File name: \_meandata.py

**Description:**

This file defines the “MeanDataDomain” class.

***Class:*** MeanDataDomain(Domain)

The parameter “Domain” means that “MeanDataDomain” class inherits from “Domain” class.

***Methods:***

1. \_\_init\_\_(self):

Domain.\_\_init\_\_(self, " MeanDataDomain ", tc.CMD\_GET\_MEANDATA\_VARIABLE, None,

tc.CMD\_SUBSCRIBE\_MEANDATA\_VARIABLE, tc.RESPONSE\_SUBSCRIBE\_MEANDATA\_VARIABLE,

tc.CMD\_SUBSCRIBE\_MEANDATA\_CONTEXT, tc.RESPONSE\_SUBSCRIBE\_MEANDATA\_CONTEXT)

*“tc” is traci.constants*

* 1. self: an instance of class MeanDataDomain.
  2. " MeanDataDomain ": The name of domain is " MeanDataDomain".
  3. tc.CMD\_GET\_ MEANDATA\_VARIABLE *:* A commond that gets meandata variable.

CMD\_GET\_LANEAREA\_VARIABLE = 0xad

* 1. None: cmdSetID is none.
  2. tc.CMD\_SUBSCRIBE\_ MEANDATA \_VARIABLE: A command subscribes meandata variable.

CMD\_SUBSCRIBE\_MEANDATA\_VARIABLE = 0x5a

* 1. tc.RESPONSE\_SUBSCRIBE\_ MEANDATA \_VARIABLE: A response that subscribe meandata variable.

RESPONSE\_SUBSCRIBE\_MEANDATA\_VARIABLE = 0x6a

* 1. tc.CMD\_SUBSCRIBE\_ MEANDATA\_CONTEXT: A command subscribes meandata context.

CMD\_SUBSCRIBE\_MEANDATA\_CONTEXT = 0x0a

* 1. tc.RESPONSE\_SUBSCRIBE\_ MEANDATA\_CONTEXT: The response that subscribes meandata context.

RESPONSE\_SUBSCRIBE\_MEANDATA\_CONTEXT = 0x1a

# File name: \_multientryexit.py

**Description:**

This file defines the “MultiEntryExitDomain” class which contains several methods for objects of MultiEntryExitDomain to call to gain information of the named multi-entry/multi-exit detector.

***Class:*** MultiEntryExitDomain (Domain)

The parameter “Domain” means that “MultiEntryExitDomain” class inherits from “Domain” class.

***Methods:***

1. \_\_init\_\_(self):

Domain.\_\_init\_\_(self, "multientryexit",

tc.CMD\_GET\_MULTIENTRYEXIT\_VARIABLE, tc.CMD\_SET\_MULTIENTRYEXIT\_VARIABLE,

tc.CMD\_SUBSCRIBE\_MULTIENTRYEXIT\_VARIABLE, tc.RESPONSE\_SUBSCRIBE\_MULTIENTRYEXIT\_VARIABLE, tc.CMD\_SUBSCRIBE\_MULTIENTRYEXIT\_CONTEXT, tc.RESPONSE\_SUBSCRIBE\_MULTIENTRYEXIT\_CONTEXT, subscriptionDefault=(tc.LAST\_STEP\_VEHICLE\_NUMBER,))

*“tc” is traci.constants*

* 1. self: an instance of class MultiEntryExitDomain.
  2. " multientryexit ": The name of domain is " multientryexit ".
  3. tc.CMD\_GET\_ MULTIENTRYEXIT \_VARIABLE *:* A commond that gets multi-exit detector variable.

CMD\_GET\_MULTIENTRYEXIT\_VARIABLE = 0xa1

* 1. tc.CMD\_SET\_MULTIENTRYEXIT\_VARIABLE: A command sets l multi-exit detector variable.

CMD\_SET\_MULTIENTRYEXIT\_VARIABLE = 0xc1

* 1. tc.CMD\_SUBSCRIBE\_ MULTIENTRYEXIT \_VARIABLE: A command subscribes multi-exit detector variable.

CMD\_SUBSCRIBE\_MULTIENTRYEXIT\_VARIABLE = 0xd1

* 1. tc.RESPONSE\_SUBSCRIBE\_ MULTIENTRYEXIT \_VARIABLE: A response that subscribe multi-exit detector variable.

RESPONSE\_SUBSCRIBE\_MULTIENTRYEXIT\_VARIABLE = 0xe1

* 1. tc.CMD\_SUBSCRIBE\_ MULTIENTRYEXIT \_CONTEXT: A command subscribes multi-exit detector context.

CMD\_SUBSCRIBE\_MULTIENTRYEXIT\_CONTEXT = 0x81

* 1. tc.RESPONSE\_SUBSCRIBE\_ MULTIENTRYEXIT \_CONTEXT: The response that subscribes multi-exit detector context.

RESPONSE\_SUBSCRIBE\_MULTIENTRYEXIT\_CONTEXT = 0x91

1. getLastStepVehicleNumber(self, detID):

Returns the number of vehicles which were halting during the last time step.

1. getLastStepMeanSpeed(self, detID):

Returns the mean speed in m/s of vehicles that have been within the named multi-entry detector within the last simulation step.

1. getLastStepVehicleIDs(self, detID):

Returns the list of ids of vehicles that have been within the named multi- entry detector in the last simulation step.

1. getLastStepHaltingNumber(self, detID):

Returns the number of vehicles which were halting during the last time step.

# File name: \_overheadwire.py

**Description:**

This file defines the “OverheadWireDomain” class which contains several methods for objects of OverheadWireDomain to call to gain information of the named overheadwire variable.

***Class:*** OverheadWireDomain (Domain)

The parameter “Domain” means that “OverheadWireDomain” class inherits from “Domain” class.

***Methods:***

1. \_\_init\_\_(self):

Domain.\_\_init\_\_(self, "overheadwire", tc.CMD\_GET\_OVERHEADWIRE\_VARIABLE, tc.CMD\_SET\_OVERHEADWIRE\_VARIABLE, tc.CMD\_SUBSCRIBE\_OVERHEADWIRE\_VARIABLE, tc.RESPONSE\_SUBSCRIBE\_OVERHEADWIRE\_VARIABLE, tc.CMD\_SUBSCRIBE\_OVERHEADWIRE\_CONTEXT, tc.RESPONSE\_SUBSCRIBE\_OVERHEADWIRE\_CONTEXT)

*“tc” is traci.constants*

* 1. self: an instance of class overheadwire.
  2. "overheadwire ": The name of domain is " overheadwire".
  3. tc.CMD\_GET\_OVERHEADWIRE\_VARIABLE *:* A commond that gets overheadwire variable.

CMD\_GET\_OVERHEADWIRE\_VARIABLE = 0x2b

* 1. tc.CMD\_SET\_ OVERHEADWIRE\_VARIABLE: A command sets overheadwire variable.

CMD\_SET\_OVERHEADWIRE\_VARIABLE = 0x4b

* 1. tc.CMD\_SUBSCRIBE\_ OVERHEADWIRE\_VARIABLE: A command subscribes overheadwire variable.

CMD\_SUBSCRIBE\_OVERHEADWIRE\_VARIABLE = 0x5b

* 1. tc.RESPONSE\_SUBSCRIBE\_ OVERHEADWIRE \_VARIABLE: A response that subscribe overheadwire variable.

RESPONSE\_SUBSCRIBE\_OVERHEADWIRE\_VARIABLE = 0x6b

* 1. tc.CMD\_SUBSCRIBE\_ OVERHEADWIRE \_CONTEXT: A command subscribes overheadwire context.

CMD\_SUBSCRIBE\_OVERHEADWIRE\_CONTEXT = 0x0b

* 1. tc.RESPONSE\_SUBSCRIBE\_ OVERHEADWIRE \_CONTEXT: The response that subscribes overheadwire context.

RESPONSE\_SUBSCRIBE\_OVERHEADWIRE\_CONTEXT = 0x1b

1. getLaneID(self, stopID):

Returns the lane of this calibrator (if it applies to a single lane).

1. getStartPos(self, stopID):

The starting position of the stop along the lane measured in meter.

1. getEndPos(self, stopID):

The end position of the stop along the lane measured in meter.

1. getName(self, stopID):

Returns the name of this stop.

1. getVehicleCount(self, stopID):

Get the total number of vehicles stopped at the named overhead wire.

1. getVehicleIDs(self, stopID):

Get the IDs of vehicles stopped at the named overhead wire.

# File name: \_parkingarea.py

**Description:**

This file defines the “parkingarea” class which contains several methods for objects of parkingarea to call to gain information of the named parkingarea variable.

***Class:*** parkingarea (Domain)

The parameter “Domain” means that “parkingarea” class inherits from “Domain” class.

***Methods:***

1. \_\_init\_\_(self):

Domain.\_\_init\_\_(self, "parkingarea", tc.CMD\_GET\_PARKINGAREA\_VARIABLE, tc.CMD\_SET\_PARKINGAREA\_VARIABLE, tc.CMD\_SUBSCRIBE\_PARKINGAREA\_VARIABLE, tc.RESPONSE\_SUBSCRIBE\_PARKINGAREA\_VARIABLE, tc.CMD\_SUBSCRIBE\_PARKINGAREA\_CONTEXT, tc.RESPONSE\_SUBSCRIBE\_PARKINGAREA\_CONTEXT)

*“tc” is traci.constants*

* 1. self: an instance of class parkingarea.
  2. " parkingarea": The name of domain is " parkingarea".
  3. tc.CMD\_GET\_PARKINGAREA\_VARIABLE *:* A commond that gets parkingarea variable.

CMD\_GET\_PARKINGAREA\_VARIABLE = 0x24

* 1. tc.CMD\_SET\_PARKINGAREA\_VARIABLE: A command sets parkingarea variable.

CMD\_SET\_PARKINGAREA\_VARIABLE = 0x44

* 1. tc.CMD\_SUBSCRIBE\_ PARKINGAREA \_VARIABLE: A command subscribes parkingarea variable.

CMD\_SUBSCRIBE\_PARKINGAREA\_VARIABLE = 0x54

* 1. tc.RESPONSE\_SUBSCRIBE\_ PARKINGAREA \_VARIABLE: A response that subscribe parkingarea variable.

RESPONSE\_SUBSCRIBE\_PARKINGAREA\_VARIABLE = 0x64

* 1. tc.CMD\_SUBSCRIBE\_ PARKINGAREA \_CONTEXT: A command subscribes parkingarea context.

CMD\_SUBSCRIBE\_PARKINGAREA\_CONTEXT = 0x04

* 1. tc.RESPONSE\_SUBSCRIBE\_ PARKINGAREA\_CONTEXT: The response that subscribes parkingarea context.

RESPONSE\_SUBSCRIBE\_PARKINGAREA\_CONTEXT = 0x14

1. getLaneID(self, stopID):

Returns the lane of this calibrator (if it applies to a single lane).

1. getStartPos(self, stopID):

The starting position of the stop along the lane measured in meter.

1. getEndPos(self, stopID):

he end position of the stop along the lane measured in m.

1. getName(self, stopID):

Returns the name of this stop.

1. getVehicleCount(self, stopID):

Get the total number of vehicles stopped at the named parking area.

1. getVehicleIDs(self, stopID):

Get the IDs of vehicles stopped at the named parking area.

# File name: \_person.py

***Description:***

This file defines the “Reservation” class which contains the attributes of person and “PersonDomain” contains several methods for objects of PersonDomain to call to gain and change information of the named person variable.

***Function:***

\_readReservation(result)

Read and return the result from the Reservation class.

***Variable:***

\_RETURN\_VALUE\_FUNC:

The function that returning value is simulation.\_readStage.

***Class:*** .

1. Reservation (object)

The parameter “object” means that “Reservation” class inherits from “object” class.

* 1. ***Methods****:*
     + 1. \_\_init\_\_(self, id, persons, group, fromEdge, toEdge, departPos, arrivalPos,depart, reservationTime, state):

This method initializes some attributes for reservation class.

* + - 1. \_\_attr\_repr\_\_(self, attrname, default=""):

This method represents the attribute. If the value of attribute does not exist, default value will be returned; if the value is invalid, return “INVALID”; or return the attrname is equal to the value.

* + - 1. def \_\_repr\_\_(self):

This method represents the whole Reservation class with a person instance.

1. PersonDomain(Domain)

The parameter “Domain” means that “PersonDomain” class inherits from “Domain” class.

* 1. ***Methods:***

1. \_\_init\_\_(self):

Domain.\_\_init\_\_(self, "overheadwire", tc.CMD\_GET\_OVERHEADWIRE\_VARIABLE, tc.CMD\_SET\_OVERHEADWIRE\_VARIABLE, tc.CMD\_SUBSCRIBE\_OVERHEADWIRE\_VARIABLE, tc.RESPONSE\_SUBSCRIBE\_OVERHEADWIRE\_VARIABLE, tc.CMD\_SUBSCRIBE\_OVERHEADWIRE\_CONTEXT, tc.RESPONSE\_SUBSCRIBE\_OVERHEADWIRE\_CONTEXT)

*“tc” is traci.constants*

1. self: an instance of class PersonDomain.

1. tc.CMD\_GET\_PERSON\_VARIABLE *:* A commond that gets person variable.

CMD\_GET\_PERSON\_VARIABLE = 0xae

1. tc.CMD\_SET\_ PERSON \_VARIABLE: A command sets person variable.

CMD\_SET\_PERSON\_VARIABLE = 0xce

1. tc.CMD\_SUBSCRIBE\_ PERSON \_VARIABLE: A command subscribes person variable.

CMD\_SUBSCRIBE\_PERSON\_VARIABLE = 0xde

1. tc.RESPONSE\_SUBSCRIBE\_ PERSON \_VARIABLE: A response that subscribe person variable.

RESPONSE\_SUBSCRIBE\_PERSON\_VARIABLE = 0xee

1. tc.CMD\_SUBSCRIBE\_ OVERHEADWIRE \_CONTEXT: A command subscribes person context.

CMD\_SUBSCRIBE\_PERSON\_CONTEXT = 0x8e

1. tc.RESPONSE\_SUBSCRIBE\_ PERSON \_CONTEXT: The response that subscribes person context.

RESPONSE\_SUBSCRIBE\_PERSON\_CONTEXT = 0x9e

1. \_RETURN\_VALUE\_FUNC:

Return the value of function.

1. getSpeed(self, personID):

Returns the speed in m/s of the named person within the last step.

self: an instance of personDomain class(object).

personID: The ID of a person(string).

1. getPosition(self, personID):

Returns the position of the named person within the last step in 2-dimensional coordinates [m,m].

1. getPosition3D(self, personID):

Returns the position of the named person within the last step in [m,m,m].

1. getAngle(self, personID):

Returns the angle in degrees of the named person within the last step.

1. getSlope(self, personID):

Returns the slope at the current position of the person in degrees.

1. getRoadID(self, personID):

Returns the ID of the edge the named person was at within the last step.

1. getLaneID(self, personID):

Returns the id of the lane the named person was at within the last step.

If the current person stage does not provide a lane, "" is returned.

1. getTypeID(self, personID):

Returns the id of the type of the named person(string).

1. getLanePosition(self, personID):

Return the position of the person along the lane measured in meter(double).

1. getColor(self, personID):

Returns the RGBA color of the person(a 1\*4 vector).

1. getLength(self, personID):

Returns the length in meter of the given person.

1. getWaitingTime(self, personID):

The waiting time of a person is defined as the time (in seconds) spent with a speed below 0.1m/s since the last time it was faster than 0.1m/s. (basically, the waiting time of a person is reset to 0 every time it moves).

1. getWidth(self, personID):

Returns the width in m of this person.

1. getMinGap(self, personID):

Returns the offset (gap to front person if halting) of this person.

1. getNextEdge(self, personID):

If the person is walking, returns the next edge on the persons route (including crossing and walkingareas). If there is no further edge or the

person is in another stage, returns the empty string.

1. getEdges(self, personID, nextStageIndex=0):

Returns a list of all edges in the nth next stage.

For waiting stages this is a single edge

For walking stages this is the complete route

For driving stages this is [origin, destination]

nextStageIndex 0 retrieves value for the current stage.

nextStageIndex must be lower then value of getRemainingStages(personID).

1. getStage(self, personID, nextStageIndex=0):

Returns the type of the nth next stage

0 for not-yet-departed

1 for waiting

2 for walking

3 for driving

4 for access to busStop or trainStop

5 for personTrip

nextStageIndex 0 retrieves value for the current stage.

nextStageIndex must be lower then value of getRemainingStages(personID)

1. getRemainingStages(self, personID):

Returns the number of remaining stages (at least 1).

1. getVehicle(self, personID):

Returns the id of the current vehicle if the person is in stage driving and has entered a vehicle.

Return the empty string otherwise

1. getTaxiReservations(self, onlyNew=0):

Returns all reservations. If onlyNew is 1, each reservation is returned only once.

1. splitTaxiReservation(self, reservationID, personIDs):

Splits given list of person ids from the reservation with the given id

and creates a new reservation for these persons. Returns the new

reservation id.

1. removeStages(self, personID):

Removes all stages of the person. If no new phases are appended,

the person will be removed from the simulation in the next simulationStep().

1. add(self, personID, edgeID, pos, depart=tc.DEPARTFLAG\_NOW, typeID="DEFAULT\_PEDTYPE"):

Inserts a new person to the simulation at the given edge, position and time (in s). This function should be followed by appending Stages or the person

will immediately vanish on departure.

1. appendWaitingStage(self, personID, duration, description="waiting", stopID=""):

Appends a waiting stage with duration in s to the plan of the given person.

1. appendWalkingStage(self, personID, edges, arrivalPos, duration=-1, speed=-1, stopID=""):

Appends a walking stage to the plan of the given person

The walking speed can either be specified, computed from the duration parameter (in s) or taken from the

type of the person

1. appendDrivingStage(self, personID, toEdge, lines, stopID=""):

Appends a driving stage to the plan of the given person

The lines parameter should be a space-separated list of line ids.

1. appendStage(self, personID, stage):

Appends a stage object to the plan of the given person

Such an object is obtainable using getStage.

1. replaceStage(self, personID, stageIndex, stage):

Replaces the nth subsequent stage with the given stage object

Such an object is obtainable using getStage.

1. removeStage(self, personID, nextStageIndex):

Removes the nth next stage nextStageIndex must be lower then value of getRemainingStages(personID).

nextStageIndex 0 immediately aborts the current stage and proceeds to the next stage.

1. rerouteTraveltime(self, personID):
2. moveToXY(self, personID, edgeID, x, y, angle=tc.INVALID\_DOUBLE\_VALUE, keepRoute=1):

Place person at the given x,y coordinates and force it's angle to

the given value (for drawing).

If the angle is set to INVALID\_DOUBLE\_VALUE, the vehicle assumes the natural angle of the edge on which it is driving.

If keepRoute is set to 1, the closest position within the existing route is taken. If keepRoute is set to 0, the vehicle may move to any edge in the network but it's route then only consists of that edge.

If keepRoute is set to 2 the person has all the freedom of keepRoute=0 but in addition to that may even move outside the road network.

1. setSpeed(self, personID, speed):

Sets the maximum speed in m/s for the named person for subsequent step.

1. setType(self, personID, typeID):

Sets the id of the type for the named person.

1. setWidth(self, personID, width):

Sets the width in m for this person.

1. setHeight(self, personID, height):

Sets the height in m for this person.

Height: double.

1. setLength(self, personID, length):

Sets the length in m for the given person.

Length:double.

1. setMinGap(self, personID, minGap):

Sets the offset (gap to front person if halting) for this vehicle.

minGap: gap to front person if halting.

1. setColor(self, personID, color):

Sets the color for the vehicle with the given ID, i.e. (255,0,0) for the color red. The fourth component (alpha) is optional.

# File name: \_poi.py

**Description:**

This file defines the “PoiDomain” class which contains several methods for objects of PoiDomain to call to gain and set data on the named Poi(Point of interst).

***Class:*** PoiDomain (Domain)

The parameter “Domain” means that “PoiDomain” class inherits from “Domain” class.

***Methods:***

1. \_\_init\_\_(self)

Domain.\_\_init\_\_(self, "poi", tc.CMD\_GET\_POI\_VARIABLE, tc.CMD\_SET\_POI\_VARIABLE, tc.CMD\_SUBSCRIBE\_POI\_VARIABLE, tc.RESPONSE\_SUBSCRIBE\_POI\_VARIABLE, tc.CMD\_SUBSCRIBE\_POI\_CONTEXT, tc.RESPONSE\_SUBSCRIBE\_POI\_CONTEXT,

*“tc” is traci.constants*

* 1. self: an instance of class PoiDomain.
  2. " poi ": The name of domain is " poi ".
  3. tc.CMD\_GET\_ POI \_VARIABLE *:* A commond that gets variable.

CMD\_GET\_POI\_VARIABLE = 0xa7

* 1. tc.CMD\_SET\_ POI \_VARIABLE: A command sets poi variable.

CMD\_SET\_POI\_VARIABLE = 0xc7

* 1. tc.CMD\_SUBSCRIBE\_ POI \_VARIABLE: A command subscribes poi variable.

CMD\_SUBSCRIBE\_POI\_VARIABLE = 0xd7

* 1. tc.RESPONSE\_SUBSCRIBE\_ POI \_VARIABLE: A response that subscribe poi variable.

RESPONSE\_SUBSCRIBE\_POI\_VARIABLE = 0xe7

* 1. tc.CMD\_SUBSCRIBE\_ POI \_CONTEXT: A command subscribes poi context.

CMD\_SUBSCRIBE\_POI\_CONTEXT = 0x87

* 1. tc.RESPONSE\_SUBSCRIBE\_ POI \_CONTEXT: The response that subscribes poi context.

RESPONSE\_SUBSCRIBE\_POI\_CONTEXT = 0x97

1. getType(self, poiID):

Return the type of POI.

poiID:The ID of POI(String).

1. getPosition(self, poiID):

Returns the position coordinates of the given POI.

1. getColor(self, poiID):

Returns the rgba color of the given poi.

1. getWidth(self, poiID):

Returns the width of the given poi.

1. getHeight(self, poiID):

Returns the height of the given poi.

1. getAngle(self, poiID):

Returns the angle of the given poi.

1. getImageFile(self, poiID):

Returns the image file of the given poi.

1. setType(self, poiID, poiType):

Sets the (abstract) type of the poi.

poiType: the type of POI(String).

1. setPosition(self, poiID, x, y):

Sets the position coordinates of the poi.

x,y: The setting coordinates of POI(double,double).

1. setColor(self, poiID, color):

Sets the rgba color of the poi, i.e. (255,0,0) for the color red. The fourth component (alpha) representing the transparency information of an image is optional.

color: The rgba color wanted to be set((integer, interger, integer), a 1\*3 vector).

1. setWidth(self, poiID, width):

Sets the width of the poi.

width: double.

1. setHeight(self, poiID, height):

Sets the height of the poi.

height:double.

1. setAngle(self, poiID, angle):

Sets the angle of the poi.

angle:double.

1. setImageFile(self, poiID, imageFile):

Sets the image file of the poi.

1. add(self, poiID, x, y, color, poiType="", layer=0, imgFile="", width=1, height=1, angle=0):

Adds a poi with the given values.

layer: the layer of the poi for drawing and selecting(float).

imgFile: A bitmap to use for rendering this poi. If none is given, circle is drawn instead. The bitmap is tinted by the given color unless for white ("1,1,1")(string).

width: width of rendered image in meters(float).

height: height of rendered image in meters(float).

angle: angle of rendered image in degree(float).

1. remove(self, poiID, layer=0):

Removes the poi with the given poiID.

1. highlight(self, poiID, color=(255, 0, 0, 255), size=-1, alphaMax=-1, duration=-1, type=0):

Adds a circle of the given color highlighting the poi. If a positive size [in m] is given the size of the highlight is chosen accordingly, otherwise the image size of the poi is used as reference. If alphaMax and duration are positive, the circle fades in and out within the given duration, otherwise it is permanently added on top of the poi.

# File name: \_polygon.py

**Description:**

This file defines the “PolygonDomain” class which contains several methods for objects of PolygonDomain to call to gain and set data on the named Polygon.

***Class:*** PolygonDomain (Domain)

The parameter “Domain” means that “PolygonDomain” class inherits from “Domain” class.

***Methods:***

* + 1. \_\_init\_\_(self)

Domain.\_\_init\_\_(self, "polygon",

tc.CMD\_GET\_POLYGON\_VARIABLE,

tc.CMD\_SET\_POLYGON\_VARIABLE,

tc.CMD\_SUBSCRIBE\_POLYGON\_VARIABLE, tc.RESPONSE\_SUBSCRIBE\_POLYGON\_VARIABLE,

tc.CMD\_SUBSCRIBE\_POLYGON\_CONTEXT, tc.RESPONSE\_SUBSCRIBE\_POLYGON\_CONTEXT,

\_RETURN\_VALUE\_FUNC)

1. self: an instance of class PolyonDomain.
2. " polygon": The name of domain is " polygon".
3. tc.CMD\_GET\_ POLYGON \_VARIABLE *:* A command that gets polygon variable.
4. tc.CMD\_SET\_ POLYGON \_VARIABLE: A command sets polygon variable.
5. tc.CMD\_SUBSCRIBE\_ POLYGON \_VARIABLE: A command subscribes polygon variable.
6. tc.RESPONSE\_SUBSCRIBE\_ POLYGON \_VARIABLE: A response that subscribe polygon variable.
7. tc.CMD\_SUBSCRIBE\_ POLYGON \_CONTEXT: A command subscribes polygon context.
8. tc.RESPONSE\_SUBSCRIBE\_ POLYGON\_CONTEXT: The response that subscribes polygon context.
9. \_RETURN\_VALUE FUNC: {tc.VAR\_FILL: **lambda** result: bool(result.read(**"!i"**)[0])}
   * 1. getType(self, polygonID):

Return the type of ploygon.

polygonID:The ID of ploygon(String).

* + 1. getShape(self, polygonID):

Returns the shape of the given

* + 1. polygon.getColor(self, polygonID):

Returns the rgba color of the given polyon.

* + 1. getFilled(self, polyonID):

Returns whether the polygon is filled

* + 1. getLineWidth(self, polyonID):

Returns drawing width of unfilled polygon.

* + 1. setType(self, polygonID, polygonType):

set the abstract type of the polygon

* + 1. setShape(self, polygonID, shape):

set the shape of the given polygon

* + 1. setColor(self, polygonID, color):

set the rgba colot of the given polygon

* + 1. setFilled(self, polygonID, filled)

set the filled status of the polygon

* + 1. setlineWidth(self, polygonID, lineWidth)

set the line width for drawing unfilled polygon

* + 1. add(self, plygonID, shape, color, fill=False, polygonType=” ”, layer=0, LineWidth=1):

add a new polygon by the given information.

* + 1. addDynamics(self, polygonID, trackedObjectID="", timeSpan=(), alphaSpan=(), looped=False, rotate=True):

1. polygonID: the id of a polygon
2. trackedObjectID: ID of a SUMO traffic object, which shall be tracked by the polygon
3. timeSpan: list of time points for timing the animation keyframes (must start with element zero). If it has length zero, no animation is taken into account.
4. alphaSpan - list of alpha values to be attained at keyframes intermediate values are obtained by linear interpolation. Must have length equal to timeSpan, or zero if no alpha animation is desired.
5. looped - Whether the animation should restart when the last keyframe is reached. In that case the animation jumps to the first keyframe as soon as the last is reached. If looped==false, the controlled polygon is removed as soon as the timeSpan elapses.
6. rotate - Whether, the polygon should be rotated with the tracked object (only applies when such is given) The center of rotation is the object's position.
   * 1. Remove(self, polygnID, layer=0):

Remove a polygon with the given ID.

# File name: rerouter.py

***Description:***

This file defines the “RerouterDomain” class to initialize the

***Class:*** PolygonDomain (Domain)

The parameter “Domain” means that “PolygonDomain” class inherits from “Domain” class.

***Methods:***

Domain.\_\_init\_\_( self, "rerouter",

tc.CMD\_GET\_REROUTER\_VARIABLE,

tc.CMD\_SET\_REROUTER\_VARIABLE,

tc.CMD\_SUBSCRIBE\_REROUTER\_VARIABLE,

tc.RESPONSE\_SUBSCRIBE\_REROUTER\_VARIABLE,

tc.CMD\_SUBSCRIBE\_REROUTER\_CONTEXT,

tc.RESPONSE\_SUBSCRIBE\_REROUTER\_CONTEXT)

# File name: vehicle.py

***Description:***

This file is to read, store, and analysis all kinds of information of different kinds of vehicles.

***Class: StopData***

***Methods:***

* + - * 1. \_\_init\_\_(self,  
            lane=**""**,  
            startPos=-1,  
            endPos=-1,  
            stoppingPlaceID=**""**,  
            stopFlags=0,  
            duration=-1,  
            until=-1,  
            intendedArrival=-1,  
            arrival=-1,  
            depart=-1,  
            split=**""**,  
            join=**""**,  
            actType=**""**,  
            tripId=**""**,  
            line=**""**,  
            speed=0)

Initialize the stopdata class.

*Input Variables*

1. lane: the name of name(string)
2. startPos: start position(int)
3. endPos: end position(int)
4. stoppingPlaceID: the stopping place(string)
5. stopingFlags: the variable is defined as(integer):

*STOP\_DEFAULT = 0x00  
STOP\_PARKING = 0x01  
STOP\_TRIGGERED = 0x02  
STOP\_CONTAINER\_TRIGGERED = 0x04  
STOP\_BUS\_STOP = 0x08  
STOP\_CONTAINER\_STOP = 0x10  
STOP\_CHARGING\_STATION = 0x20  
STOP\_PARKING\_AREA = 0x40  
STOP\_OVERHEAD\_WIRE = 0x80*

1. duration: the duration time of the vehicle travel time, default -1(integer)
2. until: the time of the vehicle stops, default -1(integer)
3. intendedArrival: destination
4. arrival: the arrival time of the vehicle in seconds(integer)
5. depart: (integer)
6. speed: the speed of the vehicle
   * + - 1. \_\_attr\_repr\_\_(self, attrname, default=**""**)

Represent the attribute.

1. self: the class StopData(class)
2. attrname: the attribute name”string”
3. default: “”(string)
   * + - 1. \_\_repr\_\_(self)

Represent the whole class

* + - * 1. \_readStopData

Read the stop data in the file.

\_readStopData(result)

Input variables:

result: the file storing the data of vehicles

Output variables

nextStops: The data of all the vehicles stop(tuple)

* + - * 1. \_readBestLanes

Read the stop data in the file.

\_readBestLanes(result)

Input variables:

1. result: the file storing the data of vehicles

Output variables

lanes: The data of all the lanes(tuple)

*Functions which call this function*

\_RETURN\_VALUE\_FUNC

*Functions which are called by this function*

None

***Class: VehicleDomain(Domain)***

***function Name***

getSpeed (self, vehID)

*Input Variables*

1. self: instance of vehicle class
2. vehID:ID of named vehicle (string)

*Output*

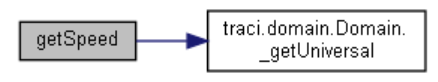
*The (longitudinal) speed in m/s of the named vehicle within the last step.*

*Functions which call this function*

*None*

*Functions which are called by this function*

\_getUniversal



***Function Name***

getLateralSpeed(self, vehID)

*Input Variables*

1. self: instance of vehicle class
2. vehID:ID of named vehicle (string)

*Output Variables*

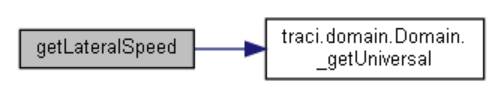
*The lateral speed in m/s of the named vehicle within the last step.*

*Functions which call this function*

*None*

*Functions which are called by this function*

\_getUniversal



***Function Name***

getAcceleration(self, vehID)

*Input Variables*

1. self: instance of vehicle class

2. vehID:ID of named vehicle (string)

*Output Variables*

*the acceleration in m/s^2 of the named vehicle within the last step.*

*Functions which call this function*

*None*

*Functions which are called by this function*

getUniversal



***Function Name***

getSpeedWithoutTraCI(self, vehID)

*Input Variables*

1. self: instance of vehicle class

2. vehID:ID of named vehicle (string)

*Output Variables*

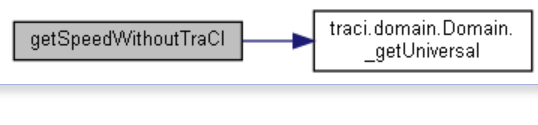
*Returns the speed that the vehicle would drive if no speed-influencing command such as setSpeed or slowDown was given.*

*Functions which call this function*

*None*

*Functions which are called by this function*

getUniversal



***Function Name***

getPosition(self, vehID)

*Input Variables*

1. self: instance of vehicle class

2. vehID:ID of named vehicle (string)

*Output Variables*

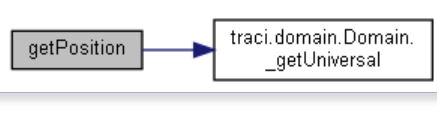
*The position of the named vehicle within the last step in the 2-dimension coordinates [m,m].*

*Functions which call this function*

*None*

*Functions which are called by this function*

\_getUniversal



***Function Name***

getPosition3D(self, vehID)

*Input Variables*

1. self: instance of vehicle class

2. vehID:ID of named vehicle (string)

*Output Variables*

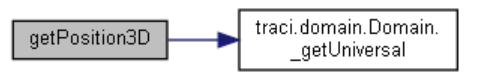
*The position of the named vehicle within the last step in the 3-dimension coordinates [m,m,m].*

*Functions which call this function*

*None*

*Functions which are called by this function*

\_getUniversal



***Function Name***

getAngle(self, vehID)

*Input Variables*

1. self: instance of vehicle class

2. vehID:ID of named vehicle (string)

*Output Variables*

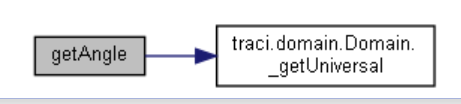
*Returns the angle in degrees of the named vehicle within the last step.*

*Functions which call this function*

*None*

*Functions which are called by this function*

\_getUniversal



***Function Name***

getColor(self, vehID)

*Input Variables*

1. self: instance of vehicle class

2. vehID:ID of named vehicle (string)

*Output Variables*

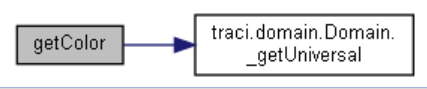
*Returns the RGBA color of vehicle*

*Functions which call this function*

*None*

*Functions which are called by this function*

\_getUniversal



***Function Name***

getTau(self, vehID)

*Input Variables*

1. self: instance of vehicle class

2. vehID:ID of named vehicle (string)

*Output Variables*

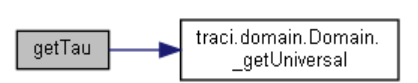
*Returns the driver's desired (minimum) headway time in s for this vehicle.*

*Functions which call this function*

*None*

*Functions which are called by this function*

\_getUniversal



***Function Name***

getLeader(self, vehID, dist=0.)

*Input Variables*

1. self: instance of vehicle class

2. vehID:ID of named vehicle (string)

3. dist: defines the minimum lookahead, default value is 0, which calculates a lookahead from the brake gap(scalar)

*Output Variables*

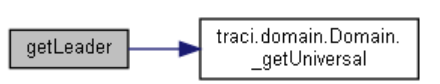
*The leading vehicle id together with the distance. The distance is measured from the front + minGap to the back of the leader, so it does not include the minGap of the vehicle.*

*Functions which call this function*

*None*

*Functions which are called by this function*

\_getUniversal



***Function Name***

getFollower(self, vehID, dist=0.)

*Input Variables*

1. self: instance of vehicle class

2. vehID:ID of named vehicle (string)

3. dist: defines the minimum lookback, default value is 0, which calculates the lookback distance from the braking distance at 4.5m/s^2 at 2\*roadSpeedLimit (scalar)

*Output Variables*

*Return the following vehicle id together with the distance. The distance is measured from the front + minGap of the follower to the back of vehID, so it does not include the minGap of the follower.*

Extra explanation of the function

Due to junctions and lane merges, there may be multiple followers. In this case, the "critical" follower is returned. This is the follower where the value of (getSecureGap - gap) is maximal, which means the closest follower will be returned.

*Functions which call this function*

*None*

*Functions which are called by this function*

\_getNeighbors



***Function Name***

getNeighbors(self, vehID, mode)

*Input Variables*

1. self: instance of vehicle class

2. vehID:ID of named vehicle (string)

3. mode: *The parameter mode is a bitset (UBYTE), specifying the following:  
 bit 1: query lateral direction (left:0, right:1)  
 bit 2: query longitudinal direction (followers:0, leaders:1)  
 bit 3: blocking (return all:0, return only blockers:1)*

(8\*2 Matrix)

To get right followers, the bitset is 10 which is binary translating into decimal 2; thus, set mode = 2.

*Output Variables*

*The returned list contains pairs (ID, dist) for all lane change relevant neighboring leaders, respectively followers, along with their longitudinal distance to the ego vehicle (egoFront - egoMinGap to leaderBack, respective followerFront - followerMinGap to egoBack. The value can be negative for overlapping neighbors). For the non-sublane case, the lists will contain at most one entry.*

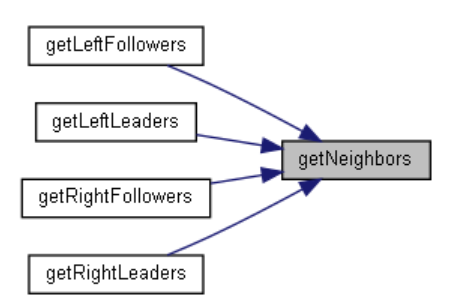
*Functions which call this function*

*getLeftFollowers*

*getLeftLeaders*

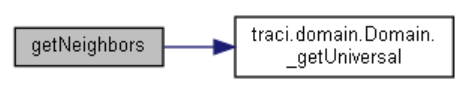
*getRightFollowers*

*getRightLeaders*

**

*Functions which are called by this function*

\_getUniversal



***Function Name***

getAccumulatedWaitingTime(self, vehID)

*Input Variables*

1. self: instance of vehicle class

2. vehID:ID of named vehicle (string)

*Output Variables*

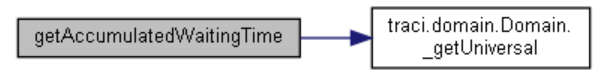
*Return the waiting time of vehicle over a certain time interval.*

*Functions which call this function*

*None*

*Functions which are called by this function*

\_getUniversal



***Function Name***

add(self, vehID, routeID, typeID="DEFAULT\_VEHTYPE", depart=None, departLane="first", departPos="base", departSpeed="0", arrivalLane="current", arrivalPos="max", arrivalSpeed="current", fromTaz="", toTaz="", line="", personCapacity=0, personNumber=0)

*Input Variables*

1. self: instance of vehicle class

2. vehID: ID of named vehicle (string)

3. routeID: ID of route that inserted vehicle locating at(string)

4.typeID: the ID of type of vehicle, such as bus, car and bicycle(string)

5.depart: Represents the states of departure(string)

6.departLane: Constraint which lane the vehicle is in when departing.

7.departPos: the position where the vehicle starts to depart.

8.departSpeed: the speed of the vehicle starting to depart.

9.arrivalLane: the lane where the vehicle arrives at the destination in.

*Output Variables*

*Return nothing.*

*Extra description of the function:*

*This function will add a new vehicle (new style with all possible parameters).*

*Functions which call this function*

*None*

*Functions which are called by this function*

\_setCmd



***Function Name***

changeLane(self, vehID, laneIndex, duration)

*Input Variables*

1. self: instance of vehicle class

2. vehID: ID of named vehicle (string)

3. laneIndex: the index of lane that the named vehicle will change to(string)

4. duration: the amount of time that vehicle to change lane during(scalar)

*Output Variables*

*Return nothing.*

*Extra description of the function:*

*Forces a lane change to the lane with the given index; if successful, the lane will be chosen for the given amount of time.*

*Functions which call this function*

*None*

*Functions which are called by this function*

\_setCmd



***Function Name***

setLaneChangeMode(self, vehID, lcm)

*Input Variables*

1. self: instance of vehicle class

2. vehID: ID of named vehicle (string)

3. lcm: lane change mode of vehicle is the bitset(12\*2 matrix)

*Output Variables*

*Return nothing.*

*Extra description of the function:*

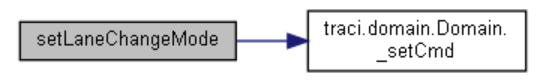
*Sets the lane change mode of vehicle as a bitset.*

*Functions which call this function*

*None*

*Functions which are called by this function*

\_setCmd



***Function Name***

setColor(self, vehID, color)

*Input Variables*

1. self: instance of vehicle class

2. vehID: ID of named vehicle (string)

3. color: It is the RGB color whose value is specified with rgb(red, green, blue). Each parameter (red, green, and blue) defines the intensity of the color as an integer between 0 and 255. For example, rgb(0, 0, 255) is rendered as blue, because the blue parameter is set to its highest value (255) and the others are set to 0.(1\*3 vector or 1\*4 vector)

*Output Variables*

*Return nothing.*

*Extra description of the function:*

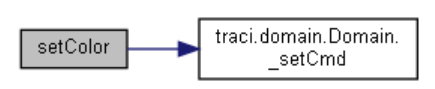
*This function will set the color of named vehicle according to vehicle ID.*

*Functions which call this function*

*None*

*Functions which are called by this function*

\_setCmd



# File name: \_trafficlight.py

***Class: TrafficLightDomain(Domain)***

Version: 07. 07. 2021

***function Name***

getRedYellowGreenState(self, tlsID)

*Input Variables*

1. self: instance of trafficLightDomain class

2. tlsID: ID of named traffic light(string)

*Output*

*Returns the state of named traffic light as a tuple of light definitions from rRgGyYoO, for red, green, yellow, off, where lower case letters mean that the stream has to decelerate.*

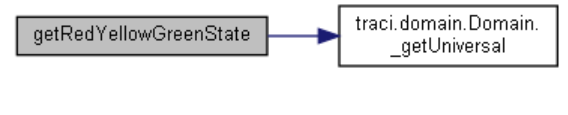
*Functions which call this function*

*setLinkState*

**

*Functions which are called by this function*

\_getUniversal



***function Name***

getControlledLanes(self, tlsID)

*Input Variables*

1. self: instance of trafficLightDomain class

2. tlsID:ID of named traffic light(string) getControlledLanes(self, tlsID)

*Output*

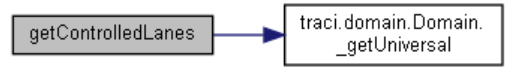
*Return the list of lanes which are controlled by the named traffic light.*

*Functions which call this function*

*None*

*Functions which are called by this function*

\_getUniversal



***function Name***

setRedYellowGreenState(self, tlsID, state)

*Input Variables*

1. self: instance of trafficLightDomain class

2. tlsID:ID of named traffic light(string)

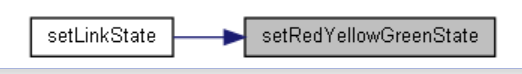
3. state: a tuple of light definitions from rRgGyYoO, for red, green, yellow, off, where lower case letters mean that the stream has to decelerate（String）

*Output*

*Return nothing*

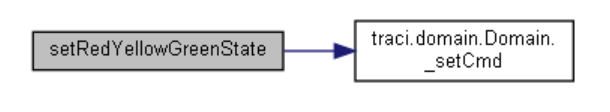
*Functions which call this function*

*setLinkState*

**

*Functions which are called by this function*

*\_*setCmd



# File name: \_person.py

Version: 09. 07. 2021

***Class name: Reservation(object)***

***Function Name: \_\_init\_\_***

Initialize the class.

\_\_init\_\_(self, id, persons, group, fromEdge, toEdge, departPos, arrivalPos,  
 depart, reservationTime, state)

Input variables:

1. self: the class itself
2. id: the id of a person
3. persons:
4. group:
5. fromEdge:
6. toEdge:
7. arrivalPos:
8. departPos:
9. depart:
10. reservationTime:
11. state:

0: return all reservations regardless of state

1: return only new reservations

2: return reservations already retrieved

4: return reservations that have been assigned to a taxi

8: return reservations that have been picked up

Output variables:

None

*Functions which call this function*

None

*Functions which are called by this function*

None

***Function Name: \_\_attr\_repr\_\_***

Described before

***Function Name: \_\_repr\_\_***

Described before

***Function Name: \_readReservation***

Read the data of person

\_readReservation(result)

Input:

Result: the data of person in file(string)

Output:

Reservation: the Reservation class containing the data of person.

*Functions which call this function*

getTaxiReservations(self, onlyNew=0):

*Functions which are called by this function*

None

***Class name: PersonDomain(Domain)***

***Function Name: \_\_init\_\_***

Initialize the PersonDomain class

\_\_init\_\_(self)

Input:

1. self: the PersonDomain class itself

Output:

None

*Functions which call this function*

None

*Functions which are called by this function*

None

***Function Name: getSpeed***

Get the speed in m/s of the named person

getSpeed(self, personID)

Input:

1. self: the class PersonDomain itself
2. personID: the ID of the named person(string)

Output:

None

**

***Function Name: getPosition()***

Get the speed in m/s of the named person

getSpeed(self, personID)

Input:

1. self: the class PersonDomain itself
2. personID: the ID of the named person(string)

Output:

None



***Function Name: getAngle()***

Get the angle of the named person

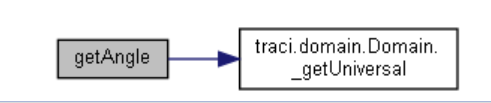
getSpeed(self, personID)

Input:

1. self: the class PersonDomain itself
2. personID: the ID of the named person(string)

Output:

The angle in degrees of the named person(string)



***Function Name: getSlope()***

Get the angle of slope of the named person

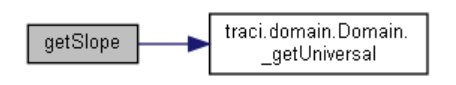
getSlope(self, personID)

Input:

1. self: the class PersonDomain itself
2. personID: the ID of the named person(string)

Output:

The angle of the slope in degree of the named person(string)



***Function Name: getTaxiReservations***

Get the taxi reservation of person.

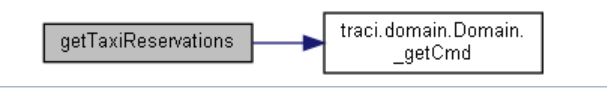
getTaxiReservations(self, onlyNew=0)

Input:

1. self: the class PersonDomain itself
2. onlyNew: default to be 0. If is 1, the reservation is returned only once.

Output:

tuple(result): the result of all the reservations in the tuple type.



# File name: \_edge.py

***Class name: EdgeDomain(Domain)***

***Function Name:***

\_\_init\_\_(self)

Input:

1. self: the class EdgeDomain itself

Output:

None

***Function Name: getAdaptedTraveltime***

Return the travel time value in second on the edge

getAdaptedTraveltime(self, edgeID, time)

Input:

1. self: the class EdgeDomain itself(class)
2. edgeID: the id of the edge(string)
3. time: the given time point(int)

Output:

The travel time on the edge at the given time(string)

***Function Name:***

getWaitingTime(self, edgeID)

get the sum of the waiting time of all vehicles on the edge

Input:

1. self: the class EdgeDomain it self
2. edgeID: the id of the edge(string)

Output:

Return the sum of the waiting time of all vehicles on the edge.(string)

# File name: domain.py

***Function Name:***

\_readParameterWithKey(result)

Read the typed strings in the file

Input:

1. result: file data

Output:

1. key:
2. val:

***Function Name:***

\_parse(valueFunc, varID, data)

Parse the data in the file into the relevant type.

If the type is not defined, the system will display unknown variable.

Input:

1. valueFunc: the container string
2. varID: the index in the valueFunc
3. data: the data need to be paresd

Output:

Parse the data by the connection before them into the wanted type.

***Class name: SubscriptionResults***

Write the read the data of subscription of user.

***Class name: Domain***

***Function Name:***

\_\_init\_\_(self, name, cmdGetID, cmdSetID,  
 subscribeID, subscribeResponseID,  
 contextID, contextResponseID,  
 retValFunc=**None**, deprecatedFor=**None**,  
 subscriptionDefault=(tc.TRACI\_ID\_LIST,))

Initialize the data in the domain class.

***Function Name:***

\_getUniversal(self, varID, objectID=**""**, format=**""**, \*values)

# File name: connection.py

***Description:***

This file use socket to connect the program with the Internet to realize communication.

Function:

\_readParameterWithKey(result)

Read the typed strings in the file

Input:

1. result: file data

Output:

1. key:
2. val:

***Function Name:***

\_parse(valueFunc, varID, data)

Parse the data in the file into the relevant type.

If the type is not defined, the system will display unknown variable.

Input:

1. valueFunc: the container string
2. varID: the index in the valueFunc
3. data: the data need to be paresd

Output:

Parse the data by the connection before them into the wanted type.

***Class name: SubscriptionResults***

Write the read the data of subscription of user.

***Class name: Domain***

***Function Name:***

\_\_init\_\_(self, name, cmdGetID, cmdSetID,  
 subscribeID, subscribeResponseID,  
 contextID, contextResponseID,  
 retValFunc=**None**, deprecatedFor=**None**,  
 subscriptionDefault=(tc.TRACI\_ID\_LIST,))

Initialize the data in the domain class.

***Function Name:***

\_getUniversal(self, varID, objectID=**""**, format=**""**, \*values)

# File name: exceptions.py

***Description:***

This file is about the exceptions for all TraCI errors.

***Class: TraCIException(Exception)***

The class TraCIException inherit the class exception, for all TraCI errors, and ensure the connection intact.

***Methods:***

1. \_\_init\_\_(self, desc, command=None, errorTypr=None)
2. getCommand(self)
3. getType(self)

***Class: FatalTraCIError(Exception)***

This class FatalTraCIError inherit the class exception, for the errors which do not follow for continuation.

***Methods:***

1. \_\_init\_\_(self, desc)

# File name: storage.py

***Description:***

This file is to read the data of specific information in the storage file.

***Class: Storage***

Methods:

1. \_\_init\_\_(self, content)
2. Read(self, format)
3. readInt(self)
4. readTypedInt(self)
5. readTypedByte(self)
6. readDouble(self)
7. readTypedDouble(self)
8. readLength(self)
9. readString(self)
10. readTypedString(self)
11. readStringList(self)
12. readTypedStringList(self)
13. readShape(self)
14. readCompound(self, expectedSize=None)
15. ready(self)
16. printDebug(self)