

---

# TraSMAPy

*Release 1.0.0*

**João de Jesus Costa, Ana Inês Oliveira de Barros, João António C**

**Jun 16, 2023**



## INTRODUCTION:

<b>1</b>	<b>Installing TraSMAPy</b>	<b>3</b>
1.1	Virtual environment . . . . .	3
1.2	Development mode . . . . .	3
<b>2</b>	<b>Quickstart with TraSMAPy</b>	<b>5</b>
2.1	Generating our first network . . . . .	5
2.2	Using TraSMAPy for the first time . . . . .	5
2.3	Adding vehicles to the simulation . . . . .	6
2.4	Examining the network . . . . .	6
2.5	Introduction to queries . . . . .	7
2.6	The next steps . . . . .	8
<b>3</b>	<b>API Reference</b>	<b>9</b>
3.1	trasmapy . . . . .	9
3.2	SignalColor . . . . .	12
3.3	_TLPhase . . . . .	13
3.4	_TLProgram . . . . .	13
3.5	_Link . . . . .	14
3.6	_TrafficLight . . . . .	14
3.7	Toll . . . . .	16
3.8	_Control . . . . .	16
3.9	_PublicServices . . . . .	17
3.10	_Fleet . . . . .	17
3.11	RemoveReason . . . . .	18
3.12	ScheduledStop . . . . .	19
3.13	VehicleClass . . . . .	19
3.14	MoveReason . . . . .	21
3.15	_VehicleStop . . . . .	21
3.16	_Users . . . . .	22
3.17	_VehicleType . . . . .	23
3.18	_Vehicle . . . . .	24
3.19	_Route . . . . .	27
3.20	StopType . . . . .	27
3.21	Color . . . . .	28
3.22	_Colorable . . . . .	28
3.23	_Edge . . . . .	29
3.24	_Detector . . . . .	30
3.25	_BusStop . . . . .	31
3.26	_Lane . . . . .	31
3.27	_Stop . . . . .	33

3.28	<code>_ParkingArea</code> . . . . .	34
3.29	<code>_ChargingStation</code> . . . . .	34
3.30	<code>_Network</code> . . . . .	35
3.31	<code>_LaneStop</code> . . . . .	36
3.32	<code>_StopLocation</code> . . . . .	36
<b>Python Module Index</b>		<b>39</b>
<b>Index</b>		<b>41</b>

TraSMAPy is a high-level object-oriented python API focused on both simplicities of usage by non-developers and feature completeness for the [SUMO traffic simulator](#). The API is intended to be used by both software developers and engineers without an informatics background. Users only require minimal knowledge of the Python programming language to use the API, making it ideal for engineers of non-informatics-related fields.

The open-source code is available on [GitHub](#) and licensed under an MIT license.



## INSTALLING TRASMAPY

You can install TraSMAPy using the following command:

```
pip install "git+https://github.com/JoaoCostaIFG/TraSMAPy.git"
```

This will install TraSMAPy and all its dependencies.

### 1.1 Virtual environment

Optionally, you can create a virtual environment and install TraSMAPy inside it:

```
python3 -m venv venv
source venv/bin/activate
pip install "git+https://github.com/JoaoCostaIFG/TraSMAPy.git"
```

Don't forget to activate the virtual environment before using TraSMAPy, and to add the *venv* directory to your *.gitignore* file.

### 1.2 Development mode

If you want to contribute to TraSMAPy, you can clone the repository, and install it in development mode:

```
git clone "https://github.com/JoaoCostaIFG/TraSMAPy.git"
cd TraSMAPy
python3 -m venv .venv
source .venv/bin/activate
pip install -e .
```





## QUICKSTART WITH TRASMAPY

### 2.1 Generating our first network

After installing TraSMAPy, you need a network to work with. The *netgenerate* utility from SUMO can be used to generate a simple network. The following command will generate a random network:

```
netgenerate --rand -o rand.net.xml
```

The file *rand.net.xml* contains our network. Now we need a sumo configuration file to run the simulation. You can put the following in the a *rand.sumocfg* file:

```
<configuration>
  <input>
    <net-file value="rand.net.xml"/>
  </input>
</configuration>
```

Note that you can also consult the [official SUMO documentation](#) for more information about importing, building, and customizing networks.

### 2.2 Using TraSMAPy for the first time

With this, we are ready to write our runner script and use TraSMAPy. Create a *runner.py* file with the following content:

```
#!/usr/bin/env python

from trasmapy import TraSMAPy

def run(trasMAPy: TraSMAPy):
    """execute the TraCI control loop"""
    while trasMAPy.minExpectedNumber > 0:
        trasMAPy.doSimulationStep()

    trasMAPy.closeSimulation()

if __name__ == "__main__":
    trasMAPy = TraSMAPy("rand.sumocfg")
    run(trasMAPy)
```

Running this python script will open the sumo-gui. You'll notice that if you start the simulation (by pressing the play button), the simulation will end immediately. This is because we have not added any vehicles to the simulation and are we only ticking the simulation until there are no vehicles.

## 2.3 Adding vehicles to the simulation

We can add vehicles to the simulation by changing our *runner.py* file like so:

```
def run(traSMAPy: TraSMAPy):
    for i in range(100):
        traSMAPy.users.createVehicle(f"v{i}")

    """execute the TraCI control loop"""
    while traSMAPy.minExpectedNumber > 0:
        traSMAPy.doSimulationStep()

    traSMAPy.closeSimulation()
```

This will spawn 100 vehicles in the simulation. If you run the simulation again, you'll notice that the vehicles will move around the network. You can also use the TraSMAPy API to control the vehicles. For example, you can change the speed of all vehicles like so:

```
def run(traSMAPy: TraSMAPy):
    for i in range(100):
        v = traSMAPy.users.createVehicle(f"v{i}")
        v.speed = 10

    """execute the TraCI control loop"""
    while traSMAPy.minExpectedNumber > 0:
        for vehicle in traSMAPy.users.getVehicles():
            vehicle.setSpeed(10)
        traSMAPy.doSimulationStep()

    traSMAPy.closeSimulation()
```

As you can see, vehicle, just like everything in TraSMAPy, are objects. This abstracts away the complexity of the [TraCI API](#) and makes it easier to use.

## 2.4 Examining the network

TraSMAPy also provides an API to examine the network. For example, you can get a sum of all CO2Emissions in all edges in the network for each simulation tick like so:

```
def run(traSMAPy: TraSMAPy):
    """execute the TraCI control loop"""
    for i in range(100):
        v = traSMAPy.users.createVehicle(f"v{i}")
        v.speed = 10

    while traSMAPy.minExpectedNumber > 0:
```

(continues on next page)

(continued from previous page)

```

traSMAPy.doSimulationStep()

edges = traSMAPy.network.edges
co2Emissions = 0
for edge in edges:
    co2Emissions += edge.CO2Emissions
print(co2Emissions)

traSMAPy.closeSimulation()

```

You'll probably notice that this makes the simulation run very slowly. This is because you are iterating all network edges for each simulation tick.

## 2.5 Introduction to queries

TraSMAPy provides a query API to make it easier to query the network and aggregate statistics. For this, there are two query mechanisms available: Python functions, and the [Pyflwor query language](#). The Pyflwor query language is a query language that is inspired by the [XQuery language](#), and is probably the easiest way to make simple queries. Let's convert the previous example to a Pyflwor query:

```

def run(traSMAPy: TraSMAPy):
    """execute the TraCI control loop"""
    for i in range(100):
        v = traSMAPy.users.createVehicle(f"v{i}")
        v.speed = 10

    while traSMAPy.minExpectedNumber > 0:
        traSMAPy.doSimulationStep()

        print(traSMAPy.query("return sum(<network/edges/CO2Emissions>)))

    traSMAPy.closeSimulation()

```

Since we are interested in collecting this statistic for each simulation tick, we can register the query to be executed every simulation tick. This can be done by using the `registerQuery` method of the `TraSMAPy` class. Let's register the previous query (note that you need to provide a name for registered queries):

```

def run(traSMAPy: TraSMAPy):
    """execute the TraCI control loop"""
    for i in range(100):
        v = traSMAPy.users.createVehicle(f"v{i}")
        v.speed = 10

    traSMAPy.registerQuery("Total CO2 Emissions", "return sum(<network/edges/
↪CO2Emissions>)")

    while traSMAPy.minExpectedNumber > 0:
        traSMAPy.doSimulationStep()

        print(traSMAPy.collectedStatistics)

```

(continues on next page)

(continued from previous page)

```
traSMAPy.closeSimulation()
```

As you can see, the *collectedStatistics* attribute of the *TraSMAPy* class contains all the statistics collected by the registered queries, organized by tick and name.

We can also take advantage of the *registerQuery* method to register a query that doesn't run every simulation tick, thus having a smaller performance hit. For example, we can register a query that runs every 10 simulation ticks:

```
def run(traSMAPy: TraSMAPy):
    """execute the TraCI control loop"""
    for i in range(100):
        v = traSMAPy.users.createVehicle(f"v{i}")
        v.speed = 10

        traSMAPy.registerQuery("Total CO2 Emissions", "return sum(<network/edges/
↪CO2Emissions>)", tickInterval=10)

        while traSMAPy.minExpectedNumber > 0:
            traSMAPy.doSimulationStep()

            print(traSMAPy.collectedStatistics)

    traSMAPy.closeSimulation()
```

## 2.6 The next steps

This is just a small introduction to TraSMAPy. For more information, please refer to the [API reference](#), and to the [examples](#) on the TraSMAPy repository.

## API REFERENCE

This page contains auto-generated API reference documentation<sup>1</sup>.

### 3.1 trasmapy

#### 3.1.1 Submodules

`trasmapy.TraSMAPy`

##### Module Contents

##### Classes

---

*TraSMAPy*

---

##### Attributes

---

*tools*

---

`trasmapy.TraSMAPy.tools`

**class** `trasmapy.TraSMAPy.TraSMAPy`(*sumoCfg: str, useGui: bool = True, log: bool = False*)

**property** `network`: `trasmapy.network._Network.Network`

**property** `users`: `trasmapy.users._Users.Users`

**property** `publicServices`: `trasmapy.publicservices._PublicServices.PublicServices`

**property** `control`: `trasmapy.control._Control.Control`

**property** `step`: `int`

---

<sup>1</sup> Created with `sphinx-autoapi`

**property stepLength: float**

The length of one simulation step (s).

**property time: float**

The current simulation time (s).

**property minExpectedNumber: int**

The minimum number of vehicles expected to be in the simulation.

**property collectedStatistics: dict[int, dict]**

The accumulated statistics of the queries.

**query(query: Union[str, Callable]) → dict**

Run a query once and get its current result.

**registerQuery(queryName: str, query: Union[str, Callable], tickInterval: int = 1) → None**

Register query to be run every tick (by default). The tickInterval param can be customized to change the frequency of the statistics collection. Results are accumulated and can be obtained through the collectedStatistics property.

**doSimulationStep() → None**

**closeSimulation() → None**

**\_genQueryMap() → dict**

**\_startSimulation(sumoCfg: str, useGui: bool, log: bool) → None**

## trasmapy.\_IdentifiedObject

### Module Contents

#### Classes

---

*IdentifiedObject*

---

**class** trasmapy.\_IdentifiedObject.**IdentifiedObject**(id: str)

**property id: str**

**\_\_repr\_\_()** → str

Return repr(self).

## trasmapy.\_Query

### Module Contents

#### Classes

---

*Query*

---

---

```
class trasmapy._Query.Query(queryFunc: Callable, tickInterval: int = 1)
```

```
    tick() → bool
```

```
        Ticks the counter. Returns True if it is time to call the query.
```

```
    __call__(*args: Any, **kws: Any) → Any
```

```
trasmapy._Regulator
```

## Module Contents

### Classes

---

*[\\_BorgSingleton](#)*

---

*[Regulator](#)*

---

```
class trasmapy._Regulator._BorgSingleton
```

```
    Bases: object
```

```
    _shared_borg_state
```

```
class trasmapy._Regulator.Regulator
```

```
    Bases: \_BorgSingleton
```

```
trasmapy._SimUpdatable
```

## Module Contents

### Classes

---

*[SimUpdatable](#)*

---

```
class trasmapy._SimUpdatable.SimUpdatable
```

```
    abstract _doSimulationStep(*args, step: int, time: float) → None
```

## 3.1.2 Package Contents

### Classes

---

*[TraSMAPy](#)*

---

```
class trasmapy.TraSMAPy(sumoCfg: str, useGui: bool = True, log: bool = False)
```

**property network:** `trasmapy.network._Network.Network`

**property users:** `trasmapy.users._Users.Users`

**property publicServices:** `trasmapy.publicservices._PublicServices.PublicServices`

**property control:** `trasmapy.control._Control.Control`

**property step:** `int`

**property stepLength:** `float`  
The length of one simulation step (s).

**property time:** `float`  
The current simulation time (s).

**property minExpectedNumber:** `int`  
The minimum number of vehicles expected to be in the simulation.

**property collectedStatistics:** `dict[int, dict]`  
The accumulated statistics of the queries.

**query**(*query: Union[str, Callable]*) → dict  
Run a query once and get its current result.

**registerQuery**(*queryName: str, query: Union[str, Callable], tickInterval: int = 1*) → None  
Register query to be run every tick (by default). The tickInterval param can be customized to change the frequency of the statistics collection. Results are accumulated and can be obtained through the collected-Statistics property.

**doSimulationStep**() → None

**closeSimulation**() → None

**\_genQueryMap**() → dict

**\_startSimulation**(*sumoCfg: str, useGui: bool, log: bool*) → None

## 3.2 SignalColor

### 3.2.1 Module Contents

#### Classes

---

<i>SignalColor</i>	Generic enumeration.
--------------------	----------------------

---

**class** `SignalColor.SignalColor`

Bases: `enum.Enum`

Generic enumeration.

Derive from this class to define new enumerations.

**RED\_LIGHT** = `r`

red light - vehicles must stop



**YELLOW\_LIGHT = y**

yellow light - vehicles start to decelerate if far away, otherwise they shall pass

**GREEN\_LIGHT\_NO\_PRIORITY = g**

green light, no priority - vehicle may pass the junction if there is not a vehicle using a higher prioritised stream, otherwise they let it pass. They always decelerate on approach until they are within the configured visibility distance

**GREEN\_LIGHT\_PRIORITY = G**

green light, priority - vehicle may pass the junction

**GREEN\_RIGHT\_TURN = s**

green right-turn arrow requires stopping - vehicles may pass the junction if no vehicle uses a higher prioritised foe stream. They always stop before passing. This is only generated for junction type traffic\_light\_right\_on\_red.

**ORANGE\_LIGHT = u**

red + yellow light - indicates upcoming green light. However, vehicles may not pass yet.

**BROWN\_LIGHT = o**

off, blinking - signal is switched off, blinking indicates that vehicles have to yield

**BLUE\_LIGHT = 0**

off, no signal - signal is switched off, vehicles have the right of way

## 3.3 \_TLPhase

### 3.3.1 Module Contents

#### Classes

---

*TLPhase*

---

```
class _TLPhase.TLPhase(duration: int, colors: list[trasmapy.control.SignalColor.SignalColor], minDur: int = -1, maxDur: int = -1, next=tuple(), name: str = "")
```

Bases: traci.\_trafficlight.Phase

```
classmethod tlPhase(phase: traci._trafficlight.Phase)
```

```
setState(colors: list[trasmapy.control.SignalColor.SignalColor])
```

## 3.4 \_TLProgram

### 3.4.1 Module Contents

#### Classes

---

*TLProgram*

---

```
class _TLProgram.TLProgram(id: str, progType: int, currentPhaseIndex: int, phases:
                           list[trasmapy.control._TLPhase.TLPhase] = [], parameters={})

    property programId: str
        Returns the id of the program.

    property typeP: int
        Returns the type of the program.

    property currentPhaseIndex: int
        Returns the index of the current phase.

    property phases: list[trasmapy.control._TLPhase.TLPhase]
        Returns the list of phases.

    property parameters
        Returns the a dictionary of parameters.

    classmethod tlProg(prog: traci._trafficlight.Logic)

    __repr__ ()
        Return repr(self).
```

## 3.5 \_Link

### 3.5.1 Module Contents

#### Classes

---

*Link*

---

```
class _Link.Link(incomingId: str, outgoingId: str, viaLaneId: str)

    __repr__ ()
        Return repr(self).
```

## 3.6 \_TrafficLight

### 3.6.1 Module Contents

#### Classes

---

*TrafficLight*

---

```
class _TrafficLight.TrafficLight(id: str)
    Bases: trasmapy._IdentifiedObject.IdentifiedObject
```

**property state:** `list[trasmapy.control.SignalColor.SignalColor]`  
Returns the named traffic lights state.

**property phaseIndex:** `int`  
Returns the index of the current phase in the current program.

**property phaseDuration:** `float`  
Returns a default total duration of the active phase (s).

**property phaseName:** `str`  
Returns the name of the current phase in the current program.

**property nextSwitchTime:** `float`  
Returns the absolute simulation time at which the traffic light is schedule to switch to the next phase (s).

**property timeTillNextSwitch:** `float`  
Returns the time left for the next switch (s).

**property controlledLinkIds:** `dict[int, list[trasmapy.control._Link.Link]]`  
Returns a dictionary of links controlled by the traffic light, where the key is the tls link index of the connection.

**property controlledLaneIds:** `list[str]`  
Returns the list of lanes which are controlled by the named traffic light. Returns at least one entry for every element of the phase state (signal index).

**property programSet:** `list[trasmapy.control._TLProgram.TLProgram]`  
Returns the list of programs of the traffic light. Each program is encoded as a TrafficLogic object.

**property programId:** `str`  
“Returns the id of the current program.

**property program:** `trasmapy.control._TLProgram.TLProgram`  
“Returns the current program.

**getProgram**(*programId: str*) → `trasmapy.control._TLProgram.TLProgram`  
Returns the program with the given id.

**getBlockingVehiclesIds**(*linkIndex: int*) → `list[str]`  
Returns the ids of vehicles that occupy the subsequent rail signal block.

**getRivalVehiclesIds**(*linkIndex: int*) → `list[str]`  
Returns the ids of vehicles that are approaching the same rail signal block.

**getPriorityVehiclesIds**(*linkIndex: int*) → `list[str]`  
Returns the ids of vehicles that are approaching the same rail signal block with higher priority.

**setRedYellowGreenState**(*colors: list[trasmapy.control.SignalColor.SignalColor]*)  
Sets the phase definition. Accepts a list of SignalColors that represent light definitions. After this call, the program of the traffic light will be set to online, and the state will be maintained until the next call of `setRedYellowGreenState()` or until setting another program with `setProgram()`

**turnOff**()  
Turns off the traffic light.

**isPhaseInProgram**(*programId: str, phaseIndex: int*) → `bool`  
Returns true if the program with the given Id contains a phase with at the given index.

## 3.7 Toll

### 3.7.1 Module Contents

#### Classes

---

*Toll*

---

```
class Toll.Toll(id: str, detectors: list[trasmapy.network._Detector.Detector])
    Bases: trasmapy._IdentifiedObject.IdentifiedObject
    property detectors: list[trasmapy.network._Detector.Detector]
    abstract roadPricingScheme(detectedVehicles)
```

## 3.8 \_Control

### 3.8.1 Module Contents

#### Classes

---

*Control*

---

```
class _Control.Control
    Bases: trasmapy._SimUpdatable.SimUpdatable
    property trafficlights: list[trasmapy.control._TrafficLight.TrafficLight]
    property tolls: list[trasmapy.control.Toll.Toll]
    getTrafficLight(id: str) → trasmapy.control._TrafficLight.TrafficLight
    registerToll(toll: trasmapy.control.Toll.Toll) → None
    getToll(id: str) → trasmapy.control.Toll.Toll
        Returns the registered Toll with the given ID or raises KeyError if none is found.
    _doSimulationStep(*args, step: int, time: float) → None
```

## 3.9 \_PublicServices

### 3.9.1 Module Contents

#### Classes

---

#### *PublicServices*

---

**class** `_PublicServices.PublicServices`(*users: trasmapy.users.\_Users.Users*)

Bases: `trasmapy._SimUpdatable.SimUpdatable`

**property** `fleets`: `dict[str, trasmapy.publicservices._Fleet.Fleet]`

**createFleet**(*fleetId: str, fleetRoute: Union[trasmapy.users.\_Route.Route, None], vehicleType: trasmapy.users.\_VehicleType.VehicleType, fleetStops: list[trasmapy.users.ScheduledStop.ScheduledStop], period: float, start: float = 0, end: float = INVALID\_DOUBLE\_VALUE*) → `trasmapy.publicservices._Fleet.Fleet`

Create a fleet. If the fleetRoute is None, a Route is calculated from the given fleetStops. If the fleetRoute is None, the list of FleetStops can't be empty.

**getFleet**(*fleetId: str*) → `trasmapy.publicservices._Fleet.Fleet`

**\_doSimulationStep**(\*args, *step: int, time: float*) → None

## 3.10 \_Fleet

### 3.10.1 Module Contents

#### Classes

---

#### *Fleet*

---

**class** `_Fleet.Fleet`(*fleetId: str, fleetRoute: trasmapy.users.\_Route.Route, vehicleType:*

*trasmapy.users.\_VehicleType.VehicleType, fleetStops:*

*list[trasmapy.users.ScheduledStop.ScheduledStop], period: float, start: float = 0, end: float = INVALID\_DOUBLE\_VALUE*)

Bases: `trasmapy._IdentifiedObject.IdentifiedObject, trasmapy._SimUpdatable.SimUpdatable`

**property** `vehicleType`: `trasmapy.users._VehicleType.VehicleType`

**property** `route`: `trasmapy.users._Route.Route`

**property** `fleetStops`: `list[trasmapy.users.ScheduledStop.ScheduledStop]`

**property** `end`: `float`

**property** `period`: `float`

**property start: float**

**property lastSpawnTime: float**

Last spawn vehicle simulation time. -1 means no spawn yet.

**property nextSpawnTime: float**

Simulation time of the next vehicle spawn. Note that due to update rates, the spawn might occur later than this time. -1 means first spawn.

**property spawnedVehiclesIds: list[str]**

Ids of all vehicles spawned until the current simulation step.

**property vehicles: list[trasmapy.users.\_Vehicle.Vehicle]**

The vehicles that are currently present in the simulation.

**\_doSimulationStep(\*args, step: int, time: float) → None**

**\_spawnVehicle(time: float, users: trasmapy.users.\_Users.Users)**

## 3.11 RemoveReason

### 3.11.1 Module Contents

#### Classes

---

*RemoveReason*

Enum where members are also (and must be) ints

---

**class RemoveReason.RemoveReason**

Bases: `enum.IntEnum`

Enum where members are also (and must be) ints

**TELEPORT = 0**

Vehicle started teleport

**PARKING = 1**

Vehicle removed while parking

**ARRIVED = 2**

Vehicle arrived

**VAPORIZED = 3**

Vehicle was vaporized

**TELEPORT\_ARRIVED = 4**

Vehicle finished route during teleport

## 3.12 ScheduledStop

### 3.12.1 Module Contents

#### Classes

---

*ScheduledStop*

---

```
class ScheduledStop.ScheduledStop(stop: trasmapy.network._Stop.Stop, duration: Union[float, str] = 0.0,
                                   until: Union[float, str] = INVALID_DOUBLE_VALUE, stopParams:
                                   list[trasmapy.users.StopType.StopType] = [])
```

```
    property stop: trasmapy.network._Stop.Stop
```

```
    property stopParams: list[trasmapy.users.StopType.StopType]
```

```
    property stopTypes: list[trasmapy.users.StopType.StopType]
```

```
    property duration: float
```

```
    property until: float
```

```
    hasDuration() → bool
```

```
    hasUntilTime() → bool
```

```
    shiftUntilTime(timeReference: float) → None
```

Changes the until time of the ScheduledStop to start counting after the given timeReference. Useful when establishing public transport schedules: the until time of transports after the first should start counting on the moment it has departed.

```
    _timeStr2Sec(timeStr: str) → float
```

```
    _checkParamsValidaty()
```

## 3.13 VehicleClass

### 3.13.1 Module Contents

#### Classes

---

*VehicleClass*

---

Generic enumeration.

---

```
class VehicleClass.VehicleClass
```

```
    Bases: enum.Enum
```

```
    Generic enumeration.
```

```
    Derive from this class to define new enumerations.
```

IGNORING = ignoring  
PRIVATE = private  
EMERGENCY = emergency  
AUTHORITY = authority  
ARMY = army  
VIP = vip  
PEDESTRIAN = pedestrian  
PASSENGER = passenger  
HOV = hov  
TAXI = taxi  
BUS = bus  
COACH = coach  
DELIVERY = delivery  
TRUCK = truck  
TRAILER = trailer  
MOTORCYCLE = motorcycle  
MOPED = moped  
BICYCLE = bicycle  
EVEHICLE = evehicle  
TRAM = tram  
RAIL\_URBAN = rail\_urban  
RAIL = rail  
RAIL\_ELECTRIC = rail\_electric  
RAIL\_FAST = rail\_fast  
SHIP = ship  
CUSTOM1 = custom1  
CUSTOM2 = custom2



## 3.14 MoveReason

### 3.14.1 Module Contents

#### Classes

---

*MoveReason*

Enum where members are also (and must be) ints

---

**class** MoveReason.**MoveReason**

Bases: enum.IntEnum

Enum where members are also (and must be) ints

**AUTOMATIC = 0**

Infer reason from move distance.

**TELEPORT = 1**

Vehicle teleports to another location

**NORMAL = 2**

vehicle moved normally

## 3.15 \_VehicleStop

### 3.15.1 Module Contents

#### Classes

---

*VehicleStop*

---

**class** \_VehicleStop.**VehicleStop**(stopData: traci.\_vehicle.StopData)

**property** stop: str

**property** duration: float

**property** until: float

**property** arrival: float

**property** intendedArrival: float

**property** depart: float

**property** stopTypes: list[trasmapy.users.StopType.StopType]

**hasArrived()** → bool

**hasDeparted()** → bool

**\_\_repr\_\_()**

Return repr(self).

## 3.16 \_Users

### 3.16.1 Module Contents

#### Classes

---

##### *Users*

---

#### **class \_Users.Users**

Bases: *trasmapy.\_SimUpdatable.SimUpdatable*

**property vehicles:** *list[trasmapy.users.\_Vehicle.Vehicle]*

Retrieves an object for each vehicle currently in the simulation. The API doesn't keep track of the liveness of the references returned from this method. As such, the values returned from this method should only be kept for one tick of the simulation (e.g., for queries).

**property pendingVehicles:** *list[trasmapy.users.\_Vehicle.Vehicle]*

Retrieves an object for each pending vehicle currently in the simulation. The API doesn't keep track of the liveness of the references returned from this method. As such, the values returned from this method should only be kept for one tick of the simulation (e.g., for queries).

**property vehicleTypes:** *list[trasmapy.users.\_VehicleType.VehicleType]*

**getAllVehicleIds()** → *list[str]*

**getAllPendingVehicleIds()** → *list[str]*

**getAllVehicleTypeIds()** → *list[str]*

**getVehicleType(vehicleTypeId: str)** → *trasmapy.users.\_VehicleType.VehicleType*

Retrieves an object for each vehicle type currently in the simulation.

**getVehicle(vehicleId: str)** → *trasmapy.users.\_Vehicle.Vehicle*

Retrieve a registered vehicle reference to a vehicle in the network. See `createVehicle`.

**createVehicle(vehicleId: str, route: Union[trasmapy.users.\_Route.Route, None] = None, vehicleType: trasmapy.users.\_VehicleType.VehicleType = VehicleType('DEFAULT\_VEHTYPE'), personNumber: int = 0, personCapacity: int = 0, departTime: Union[str, float] = 'now')** → *trasmapy.users.\_Vehicle.Vehicle*

Creates a registered vehicle and adds it to the network. Registered vehicles are vehicle objects whose liveness is checked (safe). If the route is None, the vehicle will be added to a random network edge. If the route consists of two disconnected edges, the vehicle will be treated like a <trip> and use the fastest route between the two edges. If depart time is the string 'now', the depart time is the same as the vehicle spawn. Negative values for depart time have special meanings:

-1: 'triggered' -2: 'containerTriggered'

**getRoute(routeId: str)** → *trasmapy.users.\_Route.Route*

**createRouteFromIds(routeId: str, edgesIds: list[str])** → *trasmapy.users.\_Route.Route*

**createRouteFromEdges(routeId: str, edges: list[trasmapy.network.\_Edge.Edge])** → *trasmapy.users.\_Route.Route*

`_registerVehicle(vehicleId) → trasmapy.users._Vehicle.Vehicle`

`_doSimulationStep(*args, step: int, time: float) → None`

## 3.17 \_VehicleType

### 3.17.1 Module Contents

#### Classes

---

*VehicleType*

---

**class** `_VehicleType.VehicleType`(*typeId: str*)

Bases: `trasmapy._IdentifiedObject.IdentifiedObject`, `trasmapy.color._Colorable.Colorable`

**property** `length: float`

Returns the length of the vehicles of this type (m).

**property** `maxSpeed: float`

Returns the maximum speed of the vehicles of this type (m/s).

**property** `maxLateralSpeed: float`

Returns the maximum lateral speed of the vehicles of this type (m/s).

**property** `maxAcceleration: float`

Returns the maximum acceleration of the vehicles of this type (m/s<sup>2</sup>).

**property** `maxDeceleration: float`

Returns the maximum deceleration of the vehicles of this type (m/s<sup>2</sup>).

**property** `vehicleClass: trasmapy.users.VehicleClass.VehicleClass`

**property** `emissionClass: str`

**property** `shape: str`

**property** `minGap: float`

Returns the offset (gap to front vehicle if halting) of vehicles of this type (m).

**property** `minLateralGap: float`

Returns the desired lateral gap of vehicles of this type at 50 km/h (m).

**property** `width: float`

Returns the width of vehicles of this type (m).

**property** `height: float`

Returns the height of vehicles of this type (m).

**property** `personCapacity: float`

Returns the total number of people that can ride in a vehicle of this type at the same time.

**property scale: float**

Returns the traffic scaling factor of vehicles of this type.

**property color: trasmapy.color.\_Colorable.Color**

**duplicate(cloneId: str)**

## 3.18 \_Vehicle

### 3.18.1 Module Contents

#### Classes

---

*Vehicle*

---

**class \_Vehicle.Vehicle(vehicleId: str)**

Bases: *trasmapy.\_IdentifiedObject.IdentifiedObject*, *trasmapy.color.\_Colorable.Colorable*

**property vehicleClass: trasmapy.users.VehicleClass.VehicleClass**

**property vehicleType: trasmapy.users.\_VehicleType.VehicleType**

**property emissionClass: str**

**property shapeClass: str**

**property personCapacity: int**

Returns the person capacity of the vehicle.

**property personCount: int**

Returns the number of people inside the vehicle.

**property speed: float**

Returns the speed of the vehicle within the last step (m/s). Error value:  $-2^{30}$

**property lateralSpeed: float**

Returns the lateral speed of the vehicle within the last step (m/s). Error value:  $-2^{30}$

**property allowedSpeed: float**

Returns the maximum allowed speed of the lane the vehicle is in (m/s).

**property acceleration: float**

Returns the acceleration in the previous time step ( $\text{m/s}^2$ ).

**property doRerouting: bool**

Returns whether the vehicle is able to do automatic rerouting.

**property edgeId: str**

Returns the ID of the edge the vehicle was in the previous time step.

**property laneId: str**

Returns the ID of the lane the vehicle was in the previous time step.

**property drivenDistance: float**

Returns the distance the vehicle has already driven (m). Error value:  $-2^{30}$

**property CO2Emissions: float**

Returns the vehicle's CO2 emissions during this time step (mg/s). To get the value for one step multiply with the step length. Error value:  $-2^{30}$

**property COEmissions: float**

Returns the vehicle's CO emissions during this time step (mg/s). To get the value for one step multiply with the step length. Error value:  $-2^{30}$

**property HCEmissions: float**

Returns the vehicle's HC emissions during this time step (mg/s). To get the value for one step multiply with the step length. Error value:  $-2^{30}$

**property PMxEmissions: float**

Returns the vehicle's PMx emissions during this time step (mg/s). To get the value for one step multiply with the step length. Error value:  $-2^{30}$

**property NOxEmissions: float**

Returns the vehicle's NOx emissions during this time step (mg/s). To get the value for one step multiply with the step length. Error value:  $-2^{30}$

**property fuelConsumption: float**

Returns the vehicle's NOx emissions during this time step (ml/s). To get the value for one step multiply with the step length. Error value:  $-2^{30}$

**property electricityConsumption: float**

Returns the vehicle's electricity consumption during this time step (Wh/s). To get the value for one step multiply with the step length. Error value:  $-2^{30}$

**property noiseEmission: float**

Returns the noise generated by the vehicle (dBA). Error value:  $-2^{30}$

**property timeLoss: float****property color: trasmapy.color.\_Colorable.Color****property via: list[str]**

Returns the list of IDs via edges (edge it needs to pass through in the route) for the vehicle.

**static \_checkVehicleExistance(*method*)****setAcceleration(*newAccel: float, duration: float*) → None**

Sets the vehicle acceleration ( $\text{m/s}^2$ ) for the given amount of time.

**rerouteByTravelTime() → None**

Computes a new route to the current destination that minimizes travel time. The assumed values for each edge in the network can be customized in various ways. See [Simulation/Routing#Travel-time\\_values\\_for\\_routing](#). Replaces the current route by the found.

**rerouteByEffort() → None**

Computes a new route using the vehicle's internal and the global edge effort information. Replaces the current route by the found.

**isDead() → bool****isPending() → bool**

**\_getStopState()** → int

**isStoppedAnyReason()** → bool

Returns whether the vehicle's is stopped state for any reason (any stopped state)

**isStopped()** → bool

Returns whether the vehicle's stop state is: stopped

**isParking()** → bool

Returns whether the vehicle's stop state is: parking

**isTriggered()** → bool

Returns whether the vehicle's stop state is: triggered

**isContainerTriggered()** → bool

Returns whether the vehicle's stop state is: containerTriggered

**isAtBusStop()** → bool

Returns whether the vehicle's stop state is: atBusStop

**isAtContainerStop()** → bool

Returns whether the vehicle's stop state is: atContainerStop

**isAtChargingStation()** → bool

Returns whether the vehicle's stop state is: atChargingStation

**isAtParkingArea()** → bool

Returns whether the vehicle's stop state is: atParkingArea

**getStops()** → list[trasmapy.users.\_VehicleStop.VehicleStop]

**stop**(*scheduledStop: trasmapy.users.ScheduledStop.ScheduledStop*) → None

Stops the vehicle at the given location with the given schedule. Re-issuing a stop command with the same location allows changing the duration. Setting the duration to 0 cancels an existing stop. Note that it might not be possible for a vehicle to stop at a given place because of access restrictions.

**stopFor**(*stop: trasmapy.network.\_Stop.Stop, duration: float, stopParams: list[trasmapy.users.StopType.StopType] = []*) → None

Stops the vehicle at the given position in the given edge for the given duration (s). See documentation for the stop(...) method.

**stopUntil**(*stop: trasmapy.network.\_Stop.Stop, until: float, stopParams: list[trasmapy.users.StopType.StopType] = []*) → None

Stops the vehicle at the given position in the given edge until a given simulation time (s). See documentation for the stop(...) method.

**resume()** → None

Resumes the march of a stopped vehicle. Throws exception if the vehicle isn't stopped.

**moveTo**(*laneId: str, pos: float, reason: trasmapy.users.MoveReason.MoveReason = MoveReason.AUTOMATIC*) → None

Move a vehicle to a new position along its current route.

**remove**(*reason: trasmapy.users.RemoveReason.RemoveReason = RemoveReason.VAPORIZED*) → None

**changeTargetEdge**(*targetedEdge: trasmapy.network.\_Edge.Edge*) → None

## 3.19 \_Route

### 3.19.1 Module Contents

#### Classes

---

*Route*

---

```
class _Route.Route(routeId: str)
    Bases: trasmapy._IdentifiedObject.IdentifiedObject
    property edgesIds: list[str]
```

## 3.20 StopType

### 3.20.1 Module Contents

#### Classes

---

<i>StopType</i>	Enum where members are also (and must be) ints
-----------------	--

---

```
class StopType.StopType
    Bases: enum.IntEnum
    Enum where members are also (and must be) ints
    DEFAULT = 0
        Stops on the lane.
    PARKING = 1
        Whether the vehicle stops on the road or beside.
    TRIGGERED = 2
        Whether a person may end the stop.
    CONTAINER_TRIGGERED = 4
    BUS_STOP = 8
        If given, containerStop, chargingStation, edge, lane, startPos and endPos are not allowed.
    CONTAINER_STOP = 16
        If given, busStop, chargingStation, edge, lane, startPos and endPos are not allowed.
    CHARGING_STATION = 32
        If given, busStop, containerStop, edge, lane, startPos and endPos are not allowed.
    PARKING_AREA = 64
        //sumo.dlr.de/docs/Simulation/ParkingArea.html#letting_vehicles_stop_at_a_parking_area.
    Type
        Stops at a parking area. See
```

Type  
https

OVERHEAD\_WIRE = 128

## 3.21 Color

### 3.21.1 Module Contents

#### Classes

---

*Color*

---

```
class Color.Color(r: int, g: int, b: int, a: int = 255)
    property colorTuple: tuple[int, int, int]
    property colorTupleA: tuple[int, int, int, int]
    classmethod grayscale(gray: int, a: int = 255)
    classmethod hsv(h: float, s: float, v: float, a: int = 255)
    classmethod hls(h: float, l: float, s: float, a: int = 255)
    classmethod yiq(y: float, i: float, q: float, a: int = 255)
    __repr__() → str
        Return repr(self).
```

## 3.22 \_Colorable

### 3.22.1 Module Contents

#### Classes

---

*Colorable*

---

```
class _Colorable.Colorable
    abstract property color: trasmapy.color.Color.Color
```



## 3.23 \_Edge

### 3.23.1 Module Contents

#### Classes

---

*Edge*

---

**class** \_Edge.**Edge**(edgeId: str, laneList: list[trasmapy.network.\_Lane.Lane])

Bases: *trasmapy.\_IdentifiedObject.IdentifiedObject*

**property** streetName: str

Returns the street name of the edge.

**property** travelTime: float

Returns the estimated travel time for the last time step on the edge (s).

**property** CO2Emissions: float

Sum of CO2 emissions on this edge during this time step (mg).

**property** COEmissions: float

Sum of CO emissions on this edge during this time step (mg).

**property** HCEmissions: float

Sum of HC emissions on this edge during this time step (mg).

**property** PMxEmissions: float

Sum of PMx emissions on this edge during this time step (mg).

**property** NOxEmissions: float

Sum of NOx emissions on this edge during this time step (mg).

**property** fuelConsumption: float

Sum of fuel consumption on this edge during this time step (ml).

**property** electricityConsumption: float

Sum of electricity consumption on this edge during this time step (kWh).

**property** vehicleCount: int

The number of vehicles on this edge within the last time step.

**property** vehicleMeanSpeed: float

Returns the mean speed of vehicles that were on this edge within the last simulation step (m/s).

**property** vehicleIds: list[str]

Returns the list of ids of vehicles that were on the edge in the last simulation step. The order is from rightmost to leftmost lane and downstream for each lane.

**property** occupancy: float

Returns the percentage of time the edge was occupied by a vehicle (%).

**property** vehicleMeanLength: float

Returns the mean length of the vehicles on the edge in the last time step (m).

**property vehicleWaitingTime: float**

Returns the sum of the waiting times for all vehicles on the edge (s).

**property vehicleHaltCount: int**

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

**property lanes: list[trasmapy.network.\_Lane.Lane]**

**property stops: list[trasmapy.network.\_Stop.Stop]**

**getLane**(*laneId*) → *trasmapy.network.\_Lane.Lane*

**getAdaptedTravelTime**(*time: float*) → *float*

Returns the edge travel time for the given time as stored in the global container. If no such value exists, -1 is returned.

**setAdaptedTravelTime**(*travelTime: float, beginTime: float = None, endTime: float = None*) → *None*

**getEffort**(*time: float*) → *float*

Returns the edge effort for the given time as stored in the global container. If no such value exists, -1 is returned.

**setEffort**(*travelTime: float, beginTime: float = None, endTime: float = None*) → *None*

Inserts the information about the effort of the named edge valid from begin time to end time into the global edge weights container.

**setMaxSpeed**(*maxSpeed: float*) → *None*

Sets the maximum speed for the vehicles in this edge (for all lanes) to the given value.

**limitMaxSpeed**(*maxSpeed: float*) → *None*

Limits the maximum speed for the vehicles in this edge to the given value. Only affects lanes with higher maximum vehicle speeds than the given value.

**setAllowed**(*allowedVehicleClasses: list[trasmapy.users.VehicleClass.VehicleClass]*) → *None*

Set the classes of vehicles allowed to move on this edge.

**setDisallowed**(*disallowedVehicleClasses: list[trasmapy.users.VehicleClass.VehicleClass]*) → *None*

Set the classes of vehicles disallowed to move on this edge.

**allowAll**() → *None*

Allow all vehicle classes to move on this edge.

**forbidAll**() → *None*

Forbid all vehicle classes to move on this edge.

## 3.24 \_Detector

### 3.24.1 Module Contents

#### Classes

---

*Detector*

---

---

```

class _Detector.Detector(detectorId: str)
    Bases: trasmapy._IdentifiedObject.IdentifiedObject, trasmapy._SimUpdatable.SimUpdatable

    property timeSinceLastDetection: float
        Returns how many seconds elapsed since the last detection.

    property laneId: str
        Returns the ID of the lane where the detector is placed.

    property position: float
        Returns the position of the detection on its containing lane.

    listen(listener)
        Hooks into the detector. The given function will be called with the IDs of the detected vehicles there's a
        detection.

    _doSimulationStep(*args, step: int, time: float) → None

```

## 3.25 \_BusStop

### 3.25.1 Module Contents

#### Classes

---

*BusStop*

---

```

class _BusStop.BusStop(busStopId: str)
    Bases: trasmapy.network._StopLocation.StopLocation

    property name: str

    property startPos: float

    property endPos: float

    property vehicleIds: list[str]

    property personIds: list[str]

    stopType :trasmapy.users.StopType.StopType

```

## 3.26 \_Lane

### 3.26.1 Module Contents

#### Classes

---

*Lane*

---

```
class _Lane.Lane(laneId: str, stopList: list[trasmapy.network._Stop.Stop])
    Bases: trasmapy._IdentifiedObject.IdentifiedObject

    property parentEdge

    property stops: list[trasmapy.network._Stop.Stop]

    property linkCount: int
        Returns the number of links outgoing from this lane.

    property length: float
        Returns the length of the named lane (m).

    property width: float
        Returns the width of the named lane (m).

    property CO2Emissions: float
        Sum of CO2 emissions on this lane in mg during this time step (mg).

    property COEmissions: float
        Sum of CO emissions on this lane in mg during this time step (mg).

    property HCEmissions: float
        Sum of HC emissions on this lane in mg during this time step (mg).

    property PMxEmissions: float
        Sum of PMx emissions on this lane in mg during this time step (mg).

    property NOxEmissions: float
        Sum of NOx emissions on this lane in mg during this time step (mg).

    property fuelConsumption: float
        Sum of fuel consumption on this lane in ml during this time step (ml).

    property noiseEmissions: float
        Sum of noise generated on this lane (dBA).

    property electricityConsumption: float
        Sum of electricity consumption on this edge during this time step (kWh).

    property vehicleCount: int
        The number of vehicles on this lane within the last time step.

    property vehicleMeanSpeed: float
        Returns the mean speed of vehicles that were on this lane within the last simulation step (m/s)

    property vehicleIds: list[str]
        Returns the list of ids of vehicles that were on this lane in the last simulation step.

    property occupancy: float
        Returns the total lengths of vehicles on this lane during the last simulation step divided by the length of this
        lane (%).

    property vehicleMeanLength: float
        Returns the mean length of the vehicles which were on this lane in the last step (m).

    property vehicleWaitingTime: float
        Returns the sum of the waiting times for all vehicles on the lane (s).
```

**property travelTime: float**

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

**property vehicleHaltCount: int**

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.

**property maxSpeed: float**

Returns the maximum speed allowed on this lane (m/s).

**\_setParent**(*parentEdge*) → None

**limitMaxSpeed**(*maxSpeed: float*) → None

Limits the maximum speed for the vehicles in this lane. Only changes the value if it needs to be lowered.

**allowedVehicles**() → list[trasmapy.users.VehicleClass.VehicleClass]

List of allowed vehicle classes on this lane.

**disallowedVehicles**() → list[trasmapy.users.VehicleClass.VehicleClass]

List of disallowed vehicle classes on this lane.

**\_setAllowed**(*allowedVehicleClasses: list[str]*) → None

Set the classes of vehicles allowed to move on this lane.

**\_setDisallowed**(*disallowedVehicleClasses: list[str]*) → None

Set the classes of vehicles disallowed to move on this lane.

**setAllowed**(*allowedVehicleClasses: list[trasmapy.users.VehicleClass.VehicleClass]*) → None

Set the classes of vehicles allowed to move on this lane.

**setDisallowed**(*disallowedVehicleClasses: list[trasmapy.users.VehicleClass.VehicleClass]*) → None

Set the classes of vehicles disallowed to move on this lane.

**allowAll**() → None

Allow all vehicle classes to move on this lane.

**forbidAll**() → None

Forbid all vehicle classes to move on this lane.

## 3.27 \_Stop

### 3.27.1 Module Contents

#### Classes

---

*Stop*

---

**class** \_Stop.Stop(*stopId: str*)

Bases: *trasmapy.\_IdentifiedObject.IdentifiedObject*

**abstract property laneIndex: int**

```
abstract property lane
abstract property startPos: float
abstract property endPos: float
stopType :trasmapy.users.StopType.StopType
```

## 3.28 `_ParkingArea`

### 3.28.1 Module Contents

#### Classes

---

*ParkingArea*

---

```
class _ParkingArea.ParkingArea(parkingAreaId: str)
    Bases: trasmapy.network._StopLocation.StopLocation
    property name: str
    property startPos: float
    property endPos: float
    property vehicleIds: list[str]
    stopType :trasmapy.users.StopType.StopType
```

## 3.29 `_ChargingStation`

### 3.29.1 Module Contents

#### Classes

---

*ChargingStation*

---

```
class _ChargingStation.ChargingStation(chargingStationId: str)
    Bases: trasmapy.network._StopLocation.StopLocation
    property name: str
    property startPos: float
    property endPos: float
    property vehicleIds: list[str]
    stopType :trasmapy.users.StopType.StopType
```

## 3.30 `_Network`

### 3.30.1 Module Contents

#### Classes

---

#### *Network*

---

#### **class** `_Network.Network`

Bases: `trasmapy._SimUpdatable.SimUpdatable`

**property** `edges`: `list[trasmapy.network._Edge.Edge]`

Returns a list of all edges in the network.

**property** `lanes`: `list[trasmapy.network._Lane.Lane]`

Returns a list of all edges in the network.

**property** `stops`: `list[trasmapy.network._Stop.Stop]`

Returns a list of all stops in the network.

**`_indexStops`**(*laneToStopMap*, *StopClass*, *traciModule*)

**`getEdge`**(*edgeId*: *str*) → `trasmapy.network._Edge.Edge`

Returns an object representing the edge with the given ID in the network. Raises `KeyError` if the given edge doesn't exist.

**`getLane`**(*laneId*: *str*) → `trasmapy.network._Lane.Lane`

Returns an object representing the lane with the given ID in the network. Raises `KeyError` if the given lane doesn't exist in any edge.

**`getStop`**(*stopId*: *str*) → `trasmapy.network._Stop.Stop`

Returns an object representing the lane with the given ID in the network. Raises `KeyError` if the given lane doesn't exist in any edge.

**`getDetector`**(*detectorId*: *str*) → `trasmapy.network._Detector.Detector`

Returns an object representing the inductionloop (E1) with the given ID in the network. Raises `KeyError` if the given inductionloop doesn't exist in any lane.

**`createLaneStop`**(*laneId*: *str*, *endPos*: *float* = 0, *startPos*: *float* = `INVALID_DOUBLE_VALUE`) → `trasmapy.network._LaneStop.LaneStop`

**`createEdgeStop`**(*edgeId*: *str*, *endPos*: *float* = 0, *startPos*: *float* = `INVALID_DOUBLE_VALUE`) → `trasmapy.network._LaneStop.LaneStop`

**`_doSimulationStep`**(\**args*, *step*: *int*, *time*: *float*) → `None`

## 3.31 \_LaneStop

### 3.31.1 Module Contents

#### Classes

---

*LaneStop*

---

```
class _LaneStop.LaneStop(lane: trasmapy.network._Lane.Lane, endPos: float = 0, startPos: float =  
                        INVALID_DOUBLE_VALUE)
```

```
    Bases: trasmapy.network._Stop.Stop
```

```
    property laneIndex: int
```

The lane index to stop at. Traci uses the pair (edge ID, lane index) for lane stops (instead of lane ID). There isn't a way to get the index of a lane in traci. As such, we use the index of the order at which lanes are found. If a user wants to, it is possible to define lanes' indexes, in the simulation's XML files, in an order that doesn't match their positions in the lane. If this happens, the lane index used for the stop can match with a different lane (in the same edge).

```
    property lane: trasmapy.network._Lane.Lane
```

```
    property startPos: float
```

```
    property endPos: float
```

```
    stopType :trasmapy.users.StopType.StopType
```

## 3.32 \_StopLocation

### 3.32.1 Module Contents

#### Classes

---

*StopLocation*

---

```
class _StopLocation.StopLocation(stopId: str)
```

```
    Bases: trasmapy.network._Stop.Stop
```

```
    property parentLane
```

```
    property laneIndex: int
```

The lane index is ignored on all stops beside LaneStops (StopType DEFAULT).

```
    property lane
```

```
    property stopTypes: trasmapy.users.StopType.StopType
```

```
    abstract property name: str
```



```
abstract property vehicleIds: list[str]
stopType :trasmapy.users.StopType.StopType
_setParent(parentLane) → None
```

- genindex
- modindex



## PYTHON MODULE INDEX

- 
- [\\_BusStop](#), 31
- [\\_ChargingStation](#), 34
- [\\_Colorable](#), 28
- [\\_Control](#), 16
- [\\_Detector](#), 30
- [\\_Edge](#), 29
- [\\_Fleet](#), 17
- [\\_Lane](#), 31
- [\\_LaneStop](#), 36
- [\\_Link](#), 14
- [\\_Network](#), 35
- [\\_ParkingArea](#), 34
- [\\_PublicServices](#), 17
- [\\_Route](#), 27
- [\\_Stop](#), 33
- [\\_StopLocation](#), 36
- [\\_TLPhase](#), 13
- [\\_TLProgram](#), 13
- [\\_TrafficLight](#), 14
- [\\_Users](#), 22
- [\\_Vehicle](#), 24
- [\\_VehicleStop](#), 21
- [\\_VehicleType](#), 23

### C

[Color](#), 28

### m

[MoveReason](#), 21

### r

[RemoveReason](#), 18

### S

[ScheduledStop](#), 19

[SignalColor](#), 12

[StopType](#), 27

### t

[Toll](#), 16

[trasmapy](#), 9

[trasmapy.\\_IdentifiedObject](#), 10

[trasmapy.\\_Query](#), 10

[trasmapy.\\_Regulator](#), 11

[trasmapy.\\_SimUpdatable](#), 11

[trasmapy.TraSMAPy](#), 9

### V

[VehicleClass](#), 19



## Symbols

- `_BorgSingleton` (*class in traspapy.\_Regulator*), 11
- `_BusStop`
  - module, 31
- `_ChargingStation`
  - module, 34
- `_Colorable`
  - module, 28
- `_Control`
  - module, 16
- `_Detector`
  - module, 30
- `_Edge`
  - module, 29
- `_Fleet`
  - module, 17
- `_Lane`
  - module, 31
- `_LaneStop`
  - module, 36
- `_Link`
  - module, 14
- `_Network`
  - module, 35
- `_ParkingArea`
  - module, 34
- `_PublicServices`
  - module, 17
- `_Route`
  - module, 27
- `_Stop`
  - module, 33
- `_StopLocation`
  - module, 36
- `_TLPhase`
  - module, 13
- `_TLProgram`
  - module, 13
- `_TrafficLight`
  - module, 14
- `_Users`
  - module, 22
- `_Vehicle`
  - module, 24
- `_VehicleStop`
  - module, 21
- `_VehicleType`
  - module, 23
- `__call__()` (*traspapy.\_Query.Query method*), 11
- `__repr__()` (*Color.Color method*), 28
- `__repr__()` (*\_Link.Link method*), 14
- `__repr__()` (*\_TLProgram.TLProgram method*), 14
- `__repr__()` (*\_VehicleStop.VehicleStop method*), 21
- `__repr__()` (*traspapy.\_IdentifiedObject.IdentifiedObject method*), 10
- `_checkParamsValidaty()` (*Scheduled-Stop.ScheduledStop method*), 19
- `_checkVehicleExistance()` (*\_Vehicle.Vehicle static method*), 25
- `_doSimulationStep()` (*\_Control.Control method*), 16
- `_doSimulationStep()` (*\_Detector.Detector method*), 31
- `_doSimulationStep()` (*\_Fleet.Fleet method*), 18
- `_doSimulationStep()` (*\_Network.Network method*), 35
- `_doSimulationStep()` (*\_PublicServices.PublicServices method*), 17
- `_doSimulationStep()` (*\_Users.Users method*), 23
- `_doSimulationStep()` (*traspapy.\_SimUpdatable.SimUpdatable method*), 11
- `_genQueryMap()` (*traspapy.TraSMAPy method*), 12
- `_genQueryMap()` (*traspapy.TraSMAPy.TraSMAPy method*), 10
- `_getStopState()` (*\_Vehicle.Vehicle method*), 25
- `_indexStops()` (*\_Network.Network method*), 35
- `_registerVehicle()` (*\_Users.Users method*), 22
- `_setAllowed()` (*\_Lane.Lane method*), 33
- `_setDisallowed()` (*\_Lane.Lane method*), 33
- `_setParent()` (*\_Lane.Lane method*), 33
- `_setParent()` (*\_StopLocation.StopLocation method*), 37
- `_shared_borg_state` (*traspapy.\_Regulator.\_BorgSingleton attribute*), 11
- `_spawnVehicle()` (*\_Fleet.Fleet method*), 18
- `_startSimulation()` (*traspapy.TraSMAPy method*),

12  
 \_startSimulation() (*trasmapy.TraSMAPy.TraSMAPy*  
*method*), 10  
 \_timeStr2Sec() (*ScheduledStop.ScheduledStop*  
*method*), 19

## A

acceleration (*\_Vehicle.Vehicle* property), 24  
 allowAll() (*\_Edge.Edge* method), 30  
 allowAll() (*\_Lane.Lane* method), 33  
 allowedSpeed (*\_Vehicle.Vehicle* property), 24  
 allowedVehicles() (*\_Lane.Lane* method), 33  
 ARMY (*VehicleClass.VehicleClass* attribute), 20  
 arrival (*\_VehicleStop.VehicleStop* property), 21  
 ARRIVED (*RemoveReason.RemoveReason* attribute), 18  
 AUTHORITY (*VehicleClass.VehicleClass* attribute), 20  
 AUTOMATIC (*MoveReason.MoveReason* attribute), 21

## B

BICYCLE (*VehicleClass.VehicleClass* attribute), 20  
 BLUE\_LIGHT (*SignalColor.SignalColor* attribute), 13  
 BROWN\_LIGHT (*SignalColor.SignalColor* attribute), 13  
 BUS (*VehicleClass.VehicleClass* attribute), 20  
 BUS\_STOP (*StopType.StopType* attribute), 27  
 BusStop (class in *\_BusStop*), 31

## C

changeTargetEdge() (*\_Vehicle.Vehicle* method), 26  
 CHARGING\_STATION (*StopType.StopType* attribute), 27  
 ChargingStation (class in *\_ChargingStation*), 34  
 closeSimulation() (*trasmapy.TraSMAPy* method), 12  
 closeSimulation() (*trasmapy.TraSMAPy.TraSMAPy*  
*method*), 10  
 CO2Emissions (*\_Edge.Edge* property), 29  
 CO2Emissions (*\_Lane.Lane* property), 32  
 CO2Emissions (*\_Vehicle.Vehicle* property), 25  
 COACH (*VehicleClass.VehicleClass* attribute), 20  
 COEmissions (*\_Edge.Edge* property), 29  
 COEmissions (*\_Lane.Lane* property), 32  
 COEmissions (*\_Vehicle.Vehicle* property), 25  
 collectedStatistics (*trasmapy.TraSMAPy* property),  
 12  
 collectedStatistics  
 (*trasmapy.TraSMAPy.TraSMAPy* property), 10  
 Color  
 module, 28  
 color (*\_Colorable.Colorable* property), 28  
 color (*\_Vehicle.Vehicle* property), 25  
 color (*\_VehicleType.VehicleType* property), 24  
 Color (class in *Color*), 28  
 Colorable (class in *\_Colorable*), 28  
 colorTuple (*Color.Color* property), 28  
 colorTupleA (*Color.Color* property), 28  
 CONTAINER\_STOP (*StopType.StopType* attribute), 27

CONTAINER\_TRIGGERED (*StopType.StopType* attribute),  
 27  
 Control (class in *\_Control*), 16  
 control (*trasmapy.TraSMAPy* property), 12  
 control (*trasmapy.TraSMAPy.TraSMAPy* property), 9  
 controlledLaneIds (*\_TrafficLight.TrafficLight* prop-  
 erty), 15  
 controlledLinkIdIds (*\_TrafficLight.TrafficLight* prop-  
 erty), 15  
 createEdgeStop() (*\_Network.Network* method), 35  
 createFleet() (*\_PublicServices.PublicServices*  
*method*), 17  
 createLaneStop() (*\_Network.Network* method), 35  
 createRouteFromEdges() (*\_Users.Users* method), 22  
 createRouteFromIds() (*\_Users.Users* method), 22  
 createVehicle() (*\_Users.Users* method), 22  
 currentPhaseIndex (*\_TLProgram.TLProgram* prop-  
 erty), 14  
 CUSTOM1 (*VehicleClass.VehicleClass* attribute), 20  
 CUSTOM2 (*VehicleClass.VehicleClass* attribute), 20

## D

DEFAULT (*StopType.StopType* attribute), 27  
 DELIVERY (*VehicleClass.VehicleClass* attribute), 20  
 depart (*\_VehicleStop.VehicleStop* property), 21  
 Detector (class in *\_Detector*), 30  
 detectors (*Toll.Toll* property), 16  
 disallowedVehicles() (*\_Lane.Lane* method), 33  
 doRerouting (*\_Vehicle.Vehicle* property), 24  
 doSimulationStep() (*trasmapy.TraSMAPy* method),  
 12  
 doSimulationStep() (*trasmapy.TraSMAPy.TraSMAPy*  
*method*), 10  
 drivenDistance (*\_Vehicle.Vehicle* property), 24  
 duplicate() (*\_VehicleType.VehicleType* method), 24  
 duration (*\_VehicleStop.VehicleStop* property), 21  
 duration (*ScheduledStop.ScheduledStop* property), 19

## E

Edge (class in *\_Edge*), 29  
 edgeId (*\_Vehicle.Vehicle* property), 24  
 edges (*\_Network.Network* property), 35  
 edgesIds (*\_Route.Route* property), 27  
 electricityConsumption (*\_Edge.Edge* property), 29  
 electricityConsumption (*\_Lane.Lane* property), 32  
 electricityConsumption (*\_Vehicle.Vehicle* property),  
 25  
 EMERGENCY (*VehicleClass.VehicleClass* attribute), 20  
 emissionClass (*\_Vehicle.Vehicle* property), 24  
 emissionClass (*\_VehicleType.VehicleType* property),  
 23  
 end (*\_Fleet.Fleet* property), 17  
 endPos (*\_BusStop.BusStop* property), 31

endPos (*\_ChargingStation.ChargingStation* property), 34  
 endPos (*\_LaneStop.LaneStop* property), 36  
 endPos (*\_ParkingArea.ParkingArea* property), 34  
 endPos (*\_Stop.Stop* property), 34  
 EVEHICLE (*VehicleClass.VehicleClass* attribute), 20

## F

Fleet (class in *\_Fleet*), 17  
 fleets (*\_PublicServices.PublicServices* property), 17  
 fleetStops (*\_Fleet.Fleet* property), 17  
 forbidAll() (*\_Edge.Edge* method), 30  
 forbidAll() (*\_Lane.Lane* method), 33  
 fuelConsumption (*\_Edge.Edge* property), 29  
 fuelConsumption (*\_Lane.Lane* property), 32  
 fuelConsumption (*\_Vehicle.Vehicle* property), 25

## G

getAdaptedTravelTime() (*\_Edge.Edge* method), 30  
 getAllPendingVehicleIds() (*\_Users.Users* method), 22  
 getAllVehicleIds() (*\_Users.Users* method), 22  
 getAllVehicleTypeIds() (*\_Users.Users* method), 22  
 getBlockingVehiclesIds() (*\_TrafficLight.TrafficLight* method), 15  
 getDetector() (*\_Network.Network* method), 35  
 getEdge() (*\_Network.Network* method), 35  
 getEffort() (*\_Edge.Edge* method), 30  
 getFleet() (*\_PublicServices.PublicServices* method), 17  
 getLane() (*\_Edge.Edge* method), 30  
 getLane() (*\_Network.Network* method), 35  
 getPriorityVehiclesIds() (*\_TrafficLight.TrafficLight* method), 15  
 getProgram() (*\_TrafficLight.TrafficLight* method), 15  
 getRivalVehiclesIds() (*\_TrafficLight.TrafficLight* method), 15  
 getRoute() (*\_Users.Users* method), 22  
 getStop() (*\_Network.Network* method), 35  
 getStops() (*\_Vehicle.Vehicle* method), 26  
 getToll() (*\_Control.Control* method), 16  
 getTrafficLight() (*\_Control.Control* method), 16  
 getVehicle() (*\_Users.Users* method), 22  
 getVehicleType() (*\_Users.Users* method), 22  
 grayscale() (*Color.Color* class method), 28  
 GREEN\_LIGHT\_NO\_PRIORITY (*SignalColor.SignalColor* attribute), 13  
 GREEN\_LIGHT\_PRIORITY (*SignalColor.SignalColor* attribute), 13  
 GREEN\_RIGHT\_TURN (*SignalColor.SignalColor* attribute), 13

## H

hasArrived() (*\_VehicleStop.VehicleStop* method), 21

hasDeparted() (*\_VehicleStop.VehicleStop* method), 21  
 hasDuration() (*ScheduledStop.ScheduledStop* method), 19  
 hasUntilTime() (*ScheduledStop.ScheduledStop* method), 19  
 HCEmissions (*\_Edge.Edge* property), 29  
 HCEmissions (*\_Lane.Lane* property), 32  
 HCEmissions (*\_Vehicle.Vehicle* property), 25  
 height (*\_VehicleType.VehicleType* property), 23  
 hls() (*Color.Color* class method), 28  
 HOV (*VehicleClass.VehicleClass* attribute), 20  
 hsv() (*Color.Color* class method), 28

## I

id (*trasmapy.\_IdentifiedObject.IdentifiedObject* property), 10  
 IdentifiedObject (class in *trasmapy.\_IdentifiedObject*), 10  
 IGNORING (*VehicleClass.VehicleClass* attribute), 19  
 intendedArrival (*\_VehicleStop.VehicleStop* property), 21  
 isAtBusStop() (*\_Vehicle.Vehicle* method), 26  
 isAtChargingStation() (*\_Vehicle.Vehicle* method), 26  
 isAtContainerStop() (*\_Vehicle.Vehicle* method), 26  
 isAtParkingArea() (*\_Vehicle.Vehicle* method), 26  
 isContainerTriggered() (*\_Vehicle.Vehicle* method), 26  
 isDead() (*\_Vehicle.Vehicle* method), 25  
 isParking() (*\_Vehicle.Vehicle* method), 26  
 isPending() (*\_Vehicle.Vehicle* method), 25  
 isPhaseInProgram() (*\_TrafficLight.TrafficLight* method), 15  
 isStopped() (*\_Vehicle.Vehicle* method), 26  
 isStoppedAnyReason() (*\_Vehicle.Vehicle* method), 26  
 isTriggered() (*\_Vehicle.Vehicle* method), 26

## L

lane (*\_LaneStop.LaneStop* property), 36  
 lane (*\_Stop.Stop* property), 33  
 lane (*\_StopLocation.StopLocation* property), 36  
 Lane (class in *\_Lane*), 31  
 laneId (*\_Detector.Detector* property), 31  
 laneId (*\_Vehicle.Vehicle* property), 24  
 laneIndex (*\_LaneStop.LaneStop* property), 36  
 laneIndex (*\_Stop.Stop* property), 33  
 laneIndex (*\_StopLocation.StopLocation* property), 36  
 lanes (*\_Edge.Edge* property), 30  
 lanes (*\_Network.Network* property), 35  
 LaneStop (class in *\_LaneStop*), 36  
 lastSpawnTime (*\_Fleet.Fleet* property), 18  
 lateralSpeed (*\_Vehicle.Vehicle* property), 24  
 length (*\_Lane.Lane* property), 32  
 length (*\_VehicleType.VehicleType* property), 23

`limitMaxSpeed()` (*\_Edge.Edge method*), 30  
`limitMaxSpeed()` (*\_Lane.Lane method*), 33  
`Link` (class in *\_Link*), 14  
`linkCount` (*\_Lane.Lane property*), 32  
`listen()` (*\_Detector.Detector method*), 31

## M

`maxAcceleration` (*\_VehicleType.VehicleType property*), 23  
`maxDeceleration` (*\_VehicleType.VehicleType property*), 23  
`maxLateralSpeed` (*\_VehicleType.VehicleType property*), 23  
`maxSpeed` (*\_Lane.Lane property*), 33  
`maxSpeed` (*\_VehicleType.VehicleType property*), 23  
`minExpectedNumber` (*trasmapy.TraSMAPy property*), 12  
`minExpectedNumber` (*trasmapy.TraSMAPy.TraSMAPy property*), 10  
`minGap` (*\_VehicleType.VehicleType property*), 23  
`minLateralGap` (*\_VehicleType.VehicleType property*), 23  
`module`  
    *\_BusStop*, 31  
    *\_ChargingStation*, 34  
    *\_Colorable*, 28  
    *\_Control*, 16  
    *\_Detector*, 30  
    *\_Edge*, 29  
    *\_Fleet*, 17  
    *\_Lane*, 31  
    *\_LaneStop*, 36  
    *\_Link*, 14  
    *\_Network*, 35  
    *\_ParkingArea*, 34  
    *\_PublicServices*, 17  
    *\_Route*, 27  
    *\_Stop*, 33  
    *\_StopLocation*, 36  
    *\_TLPhase*, 13  
    *\_TLProgram*, 13  
    *\_TrafficLight*, 14  
    *\_Users*, 22  
    *\_Vehicle*, 24  
    *\_VehicleStop*, 21  
    *\_VehicleType*, 23  
    *Color*, 28  
    *MoveReason*, 21  
    *RemoveReason*, 18  
    *ScheduledStop*, 19  
    *SignalColor*, 12  
    *StopType*, 27  
    *Toll*, 16  
    *trasmapy*, 9

*trasmapy.\_IdentifiedObject*, 10  
*trasmapy.\_Query*, 10  
*trasmapy.\_Regulator*, 11  
*trasmapy.\_SimUpdatable*, 11  
*trasmapy.TraSMAPy*, 9  
*VehicleClass*, 19

*MOPED* (*VehicleClass.VehicleClass attribute*), 20  
*MOTORCYCLE* (*VehicleClass.VehicleClass attribute*), 20  
*MoveReason*  
    module, 21  
*MoveReason* (class in *MoveReason*), 21  
*moveTo()* (*\_Vehicle.Vehicle method*), 26

## N

*name* (*\_BusStop.BusStop property*), 31  
*name* (*\_ChargingStation.ChargingStation property*), 34  
*name* (*\_ParkingArea.ParkingArea property*), 34  
*name* (*\_StopLocation.StopLocation property*), 36  
*Network* (class in *\_Network*), 35  
*network* (*trasmapy.TraSMAPy property*), 11  
*network* (*trasmapy.TraSMAPy.TraSMAPy property*), 9  
*nextSpawnTime* (*\_Fleet.Fleet property*), 18  
*nextSwitchTime* (*\_TrafficLight.TrafficLight property*), 15  
*noiseEmission* (*\_Vehicle.Vehicle property*), 25  
*noiseEmissions* (*\_Lane.Lane property*), 32  
*NORMAL* (*MoveReason.MoveReason attribute*), 21  
*NOxEmissions* (*\_Edge.Edge property*), 29  
*NOxEmissions* (*\_Lane.Lane property*), 32  
*NOxEmissions* (*\_Vehicle.Vehicle property*), 25

## O

*occupancy* (*\_Edge.Edge property*), 29  
*occupancy* (*\_Lane.Lane property*), 32  
*ORANGE\_LIGHT* (*SignalColor.SignalColor attribute*), 13  
*OVERHEAD\_WIRE* (*StopType.StopType attribute*), 28

## P

*parameters* (*\_TLProgram.TLProgram property*), 14  
*parentEdge* (*\_Lane.Lane property*), 32  
*parentLane* (*\_StopLocation.StopLocation property*), 36  
*PARKING* (*RemoveReason.RemoveReason attribute*), 18  
*PARKING* (*StopType.StopType attribute*), 27  
*PARKING\_AREA* (*StopType.StopType attribute*), 27  
*ParkingArea* (class in *\_ParkingArea*), 34  
*PASSENGER* (*VehicleClass.VehicleClass attribute*), 20  
*PEDESTRIAN* (*VehicleClass.VehicleClass attribute*), 20  
*pendingVehicles* (*\_Users.Users property*), 22  
*period* (*\_Fleet.Fleet property*), 17  
*personCapacity* (*\_Vehicle.Vehicle property*), 24  
*personCapacity* (*\_VehicleType.VehicleType property*), 23  
*personCount* (*\_Vehicle.Vehicle property*), 24  
*personIds* (*\_BusStop.BusStop property*), 31



- phaseDuration (*\_TrafficLight.TrafficLight* property), 15  
 phaseIndex (*\_TrafficLight.TrafficLight* property), 15  
 phaseName (*\_TrafficLight.TrafficLight* property), 15  
 phases (*\_TLProgram.TLProgram* property), 14  
 PMxEmissions (*\_Edge.Edge* property), 29  
 PMxEmissions (*\_Lane.Lane* property), 32  
 PMxEmissions (*\_Vehicle.Vehicle* property), 25  
 position (*\_Detector.Detector* property), 31  
 PRIVATE (*VehicleClass.VehicleClass* attribute), 20  
 program (*\_TrafficLight.TrafficLight* property), 15  
 programId (*\_TLProgram.TLProgram* property), 14  
 programId (*\_TrafficLight.TrafficLight* property), 15  
 programSet (*\_TrafficLight.TrafficLight* property), 15  
 PublicServices (class in *\_PublicServices*), 17  
 publicServices (*trasmapy.TraSMAPy* property), 12  
 publicServices (*trasmapy.TraSMAPy.TraSMAPy* property), 9
- ## Q
- Query (class in *trasmapy.\_Query*), 10  
 query() (*trasmapy.TraSMAPy* method), 12  
 query() (*trasmapy.TraSMAPy.TraSMAPy* method), 10
- ## R
- RAIL (*VehicleClass.VehicleClass* attribute), 20  
 RAIL\_ELECTRIC (*VehicleClass.VehicleClass* attribute), 20  
 RAIL\_FAST (*VehicleClass.VehicleClass* attribute), 20  
 RAIL\_URBAN (*VehicleClass.VehicleClass* attribute), 20  
 RED\_LIGHT (*SignalColor.SignalColor* attribute), 12  
 registerQuery() (*trasmapy.TraSMAPy* method), 12  
 registerQuery() (*trasmapy.TraSMAPy.TraSMAPy* method), 10  
 registerToll() (*\_Control.Control* method), 16  
 Regulator (class in *trasmapy.\_Regulator*), 11  
 remove() (*\_Vehicle.Vehicle* method), 26  
 RemoveReason module, 18  
 RemoveReason (class in *RemoveReason*), 18  
 rerouteByEffort() (*\_Vehicle.Vehicle* method), 25  
 rerouteByTravelTime() (*\_Vehicle.Vehicle* method), 25  
 resume() (*\_Vehicle.Vehicle* method), 26  
 roadPricingScheme() (*Toll.Toll* method), 16  
 route (*\_Fleet.Fleet* property), 17  
 Route (class in *\_Route*), 27
- ## S
- scale (*\_VehicleType.VehicleType* property), 23  
 ScheduledStop module, 19  
 ScheduledStop (class in *ScheduledStop*), 19  
 setAcceleration() (*\_Vehicle.Vehicle* method), 25  
 setAdaptedTravelTime() (*\_Edge.Edge* method), 30  
 setAllowed() (*\_Edge.Edge* method), 30  
 setAllowed() (*\_Lane.Lane* method), 33  
 setDisallowed() (*\_Edge.Edge* method), 30  
 setDisallowed() (*\_Lane.Lane* method), 33  
 setEffort() (*\_Edge.Edge* method), 30  
 setMaxSpeed() (*\_Edge.Edge* method), 30  
 setRedYellowGreenState() (*\_TrafficLight.TrafficLight* method), 15  
 setState() (*\_TLPhase.TLPhase* method), 13  
 shape (*\_VehicleType.VehicleType* property), 23  
 shapeClass (*\_Vehicle.Vehicle* property), 24  
 shiftUntilTime() (*ScheduledStop.ScheduledStop* method), 19  
 SHIP (*VehicleClass.VehicleClass* attribute), 20  
 SignalColor module, 12  
 SignalColor (class in *SignalColor*), 12  
 SimUpdatable (class in *trasmapy.\_SimUpdatable*), 11  
 spawnedVehiclesIds (*\_Fleet.Fleet* property), 18  
 speed (*\_Vehicle.Vehicle* property), 24  
 start (*\_Fleet.Fleet* property), 17  
 startPos (*\_BusStop.BusStop* property), 31  
 startPos (*\_ChargingStation.ChargingStation* property), 34  
 startPos (*\_LaneStop.LaneStop* property), 36  
 startPos (*\_ParkingArea.ParkingArea* property), 34  
 startPos (*\_Stop.Stop* property), 34  
 state (*\_TrafficLight.TrafficLight* property), 14  
 step (*trasmapy.TraSMAPy* property), 12  
 step (*trasmapy.TraSMAPy.TraSMAPy* property), 9  
 stepLength (*trasmapy.TraSMAPy* property), 12  
 stepLength (*trasmapy.TraSMAPy.TraSMAPy* property), 9  
 stop (*\_VehicleStop.VehicleStop* property), 21  
 Stop (class in *\_Stop*), 33  
 stop (*ScheduledStop.ScheduledStop* property), 19  
 stop() (*\_Vehicle.Vehicle* method), 26  
 stopFor() (*\_Vehicle.Vehicle* method), 26  
 StopLocation (class in *\_StopLocation*), 36  
 stopParams (*ScheduledStop.ScheduledStop* property), 19  
 stops (*\_Edge.Edge* property), 30  
 stops (*\_Lane.Lane* property), 32  
 stops (*\_Network.Network* property), 35  
 StopType module, 27  
 stopType (*\_BusStop.BusStop* attribute), 31  
 stopType (*\_ChargingStation.ChargingStation* attribute), 34  
 stopType (*\_LaneStop.LaneStop* attribute), 36  
 stopType (*\_ParkingArea.ParkingArea* attribute), 34  
 stopType (*\_Stop.Stop* attribute), 34  
 stopType (*\_StopLocation.StopLocation* attribute), 37  
 StopType (class in *StopType*), 27

stopTypes (*\_StopLocation.StopLocation* property), 36  
stopTypes (*\_VehicleStop.VehicleStop* property), 21  
stopTypes (*ScheduledStop.ScheduledStop* property), 19  
stopUntil() (*\_Vehicle.Vehicle* method), 26  
streetName (*\_Edge.Edge* property), 29

## T

TAXI (*VehicleClass.VehicleClass* attribute), 20  
TELEPORT (*MoveReason.MoveReason* attribute), 21  
TELEPORT (*RemoveReason.RemoveReason* attribute), 18  
TELEPORT\_ARRIVED (*RemoveReason.RemoveReason* attribute), 18  
tick() (*trasmapy.\_Query.Query* method), 11  
time (*trasmapy.TraSMAPy.TraSMAPy* property), 12  
time (*trasmapy.TraSMAPy.TraSMAPy* property), 10  
timeLoss (*\_Vehicle.Vehicle* property), 25  
timeSinceLastDetection (*\_Detector.Detector* property), 31  
timeTillNextSwitch (*\_TrafficLight.TrafficLight* property), 15  
TLPhase (class in *\_TLPhase*), 13  
tlPhase() (*\_TLPhase.TLPhase* class method), 13  
tlProg() (*\_TLProgram.TLProgram* class method), 14  
TLProgram (class in *\_TLProgram*), 14  
Toll  
    module, 16  
Toll (class in *Toll*), 16  
tolls (*\_Control.Control* property), 16  
tools (in module *trasmapy.TraSMAPy*), 9  
TrafficLight (class in *\_TrafficLight*), 14  
trafficlights (*\_Control.Control* property), 16  
TRAILER (*VehicleClass.VehicleClass* attribute), 20  
TRAM (*VehicleClass.VehicleClass* attribute), 20  
trasmapy  
    module, 9  
TraSMAPy (class in *trasmapy*), 11  
TraSMAPy (class in *trasmapy.TraSMAPy*), 9  
trasmapy.\_IdentifiedObject  
    module, 10  
trasmapy.\_Query  
    module, 10  
trasmapy.\_Regulator  
    module, 11  
trasmapy.\_SimUpdatable  
    module, 11  
trasmapy.TraSMAPy  
    module, 9  
travelTime (*\_Edge.Edge* property), 29  
travelTime (*\_Lane.Lane* property), 32  
TRIGGERED (*StopType.StopType* attribute), 27  
TRUCK (*VehicleClass.VehicleClass* attribute), 20  
turnOff() (*\_TrafficLight.TrafficLight* method), 15  
typeP (*\_TLProgram.TLProgram* property), 14

## U

until (*\_VehicleStop.VehicleStop* property), 21  
until (*ScheduledStop.ScheduledStop* property), 19  
Users (class in *\_Users*), 22  
users (*trasmapy.TraSMAPy.TraSMAPy* property), 12  
users (*trasmapy.TraSMAPy.TraSMAPy* property), 9

## V

VAPORIZED (*RemoveReason.RemoveReason* attribute), 18  
Vehicle (class in *\_Vehicle*), 24  
VehicleClass  
    module, 19  
vehicleClass (*\_Vehicle.Vehicle* property), 24  
vehicleClass (*\_VehicleType.VehicleType* property), 23  
VehicleClass (class in *VehicleClass*), 19  
vehicleCount (*\_Edge.Edge* property), 29  
vehicleCount (*\_Lane.Lane* property), 32  
vehicleHaltCount (*\_Edge.Edge* property), 30  
vehicleHaltCount (*\_Lane.Lane* property), 33  
vehicleIds (*\_BusStop.BusStop* property), 31  
vehicleIds (*\_ChargingStation.ChargingStation* property), 34  
vehicleIds (*\_Edge.Edge* property), 29  
vehicleIds (*\_Lane.Lane* property), 32  
vehicleIds (*\_ParkingArea.ParkingArea* property), 34  
vehicleIds (*\_StopLocation.StopLocation* property), 36  
vehicleMeanLength (*\_Edge.Edge* property), 29  
vehicleMeanLength (*\_Lane.Lane* property), 32  
vehicleMeanSpeed (*\_Edge.Edge* property), 29  
vehicleMeanSpeed (*\_Lane.Lane* property), 32  
vehicles (*\_Fleet.Fleet* property), 18  
vehicles (*\_Users.Users* property), 22  
VehicleStop (class in *\_VehicleStop*), 21  
vehicleType (*\_Fleet.Fleet* property), 17  
vehicleType (*\_Vehicle.Vehicle* property), 24  
VehicleType (class in *\_VehicleType*), 23  
vehicleTypes (*\_Users.Users* property), 22  
vehicleWaitingTime (*\_Edge.Edge* property), 29  
vehicleWaitingTime (*\_Lane.Lane* property), 32  
via (*\_Vehicle.Vehicle* property), 25  
VIP (*VehicleClass.VehicleClass* attribute), 20

## W

width (*\_Lane.Lane* property), 32  
width (*\_VehicleType.VehicleType* property), 23

## Y

YELLOW\_LIGHT (*SignalColor.SignalColor* attribute), 12  
yiiq() (*Color.Color* class method), 28