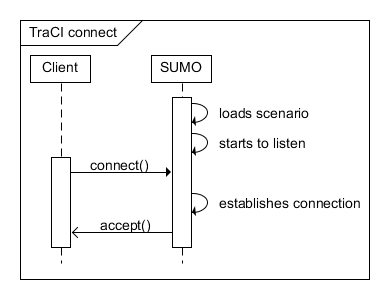
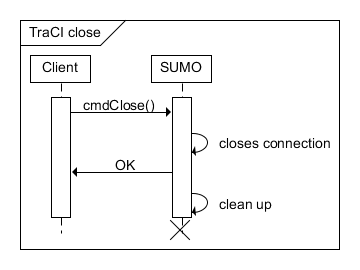
Sumo message flow

**TraCI: establishing a connection to SUMO**



The client application sends commands to [SUMO](http://sumo.dlr.de/wiki/SUMO) to control the simulation run, to influence single vehicle's behavior or to ask for environmental details. [SUMO](http://sumo.dlr.de/wiki/SUMO) answers with a Status-response to each command and additional results that depend on the given command.

**TraCI: closing a connection to SUMO**

A TCP message acts as container for a list of commands or results. Therefore, each TCP message consists of a small header that gives the overall message size and a set of commands that are put behind it. The length and identifier of each command is placed in front of the command. A scheme of this container is depicted below:

0 7 8 15

+--------------------------------------+

| Message Length including this header |

+--------------------------------------+

| (Message Length, continued) |

+--------------------------------------+ \

| Length | Identifier | |

+--------------------------------------+ > Command\_0

| Command\_0 content | |

+--------------------------------------+ /

...

+--------------------------------------+ \

| Length | Identifier | |

+--------------------------------------+ > Command\_n-1

| Command\_n-1 content | |

+--------------------------------------+ /

The class tcpip::Sockethandles a TCP connection on server as well as on client side. The class tcpip::Storage is designed to hold a list of elementary data types. By using the methodtcpip::Socket::sendExact(tcpip::Storage) and tcpip::Socket::receiveExact(Storage), messages can be send and received as Storage objects.

**Status Response**

|  |  |
| --- | --- |
| ubyte | string |
| **Result** | **Description** |

Each command is acknowledged by a status response, included are a *Result* and a *Description*. The identifier refers to the identifier of the respective command that is acknowledged.   
*Result* can have the following values:

* 0x00 in case of succes
* 0xFF if the requested command failed
* 0x01 if the requested command is not implemented in the network simulator (in addition, a *Description* text must be added)

#### Position Representations

**2DPosition (*ubyte* identifier: 0x01)**

A cartesian 2D position within the simulation network, described by two double values (x and y coordinate).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ubyte | | | | | | | | double | double |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **X** | **Y** |

#### Boundary Box (ubyte identifier: 0x05)

A boundary box represented by its lower left and upper right corner.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ubyte | | | | | | | | double | double | double | double |
| **0** | **0** | **0** | **0** | **0** | **1** | **0** | **1** | **LowerLeftX** | **LowerLeftY** | **UpperRightX** | **UpperRightY** |

#### Traffic Light Phase List (ubyte identifier: 0x0D)

This type is used to report the different phases of a traffic light. A total of *Length* phases is reported together with the preceding and succeeding roads that are affected by the respective light phase.

The following identifiers for a phase exist:

* 0x01: red
* 0x02: yellow
* 0x03: green
* 0x04: traffic light is off and blinking
* 0x05: traffic light is off, not blinking

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ubyte | | | | | | | | ubyte | string | string | ubyte |  |
| **0** | **0** | **0** | **0** | **1** | **1** | **0** | **1** | **Length** | **PrecRoad** | **SuccRoad** | **Phase** | **...** |

Example for get vehicle value:

public static void main(String[] args) {

SumoTraciConnection conn = new SumoTraciConnection(

"test/sumo\_maps/box1l/test.sumo.cfg", // config file

12345 // random seed

);

try {

conn.runServer();

// the first two steps of this simulation have no vehicles.

conn.nextSimStep();

conn.nextSimStep();

Collection<Vehicle> vehicles = conn.getVehicleRepository().getAll().values();

Vehicle aVehicle = vehicles.iterator().next();

System.out.println("Vehicle " + aVehicle

+ " will traverse these edges: "

+ aVehicle.getCurrentRoute());

conn.close();

}

catch(Exception e) {

e.printStackTrace();

}

}

initialization of read queries:

addReadQuery(Variable.SPEED, new ReadObjectVarQuery.DoubleQ (dis, dos, it.polito.appeal.traci.protocol.Constants.CMD\_GET\_VEHICLE\_VARIABLE, id, it.polito.appeal.traci.protocol.Constants.VAR\_SPEED

));

addReadQuery(Variable.POSITION,

new ReadObjectVarQuery.PositionQ (dis, dos,

it.polito.appeal.traci.protocol.Constants.CMD\_GET\_VEHICLE\_VARIABLE,

id,

it.polito.appeal.traci.protocol.Constants.VAR\_POSITION

));

initialization of change state queries:

csqvar\_ChangeSpeed = new ChangeSpeedQuery(dis, dos, id)

{

@Override

void pickResponses(java.util.Iterator<it.polito.appeal.traci.protocol.ResponseContainer> responseIterator)

throws TraCIException {

super.pickResponses(responseIterator);

queryReadSpeed().setObsolete();

}

};

@Override

public void nextStep(double step) {

getReadQuery(Variable.SPEED).setObsolete();

getReadQuery(Variable.POSITION).setObsolete();

}

/\*\*

\* @return the instance of {@link ChangeSpeedQuery} relative to this query.

\*/

public ChangeSpeedQuery queryChangeSpeed() {

return csqvar\_ChangeSpeed;

}

/\*\*

\* Execute an instance of ChangeSpeedQuery set to the given value.

\*

\* This setter method is equivalent to queryChangeSpeed().setValue(value).run().

\*/

public void changeSpeed(Double value) throws IOException {

ChangeSpeedQuery q = csqvar\_ChangeSpeed;

q.setValue(value);

q.run();

}