Definition of Vehicles, Vehicle Types, and Routes

Definition of Vehicles, Vehicle Types, and Routes

Filename extension	.rou.xml
Type of content	Vehicles, Vehicle Types, and Routes
Open format?	Yes
SUMO specific?	Yes
XML Schema	routes_file.xsd (http://sumo.dlr.de/xsd/routes_file.xsd)

There are various applications that can be used to define vehicular demand for SUMO. Of course it is also possible to define the demand file manually or to edit generated files with a text editor. Before starting, it is important to know that a vehicle in SUMO consists of three parts:

- a vehicle type which describes the vehicle's physical properties,
- a route the vehicle shall take,
- and the vehicle itself.

Both routes and vehicle types can be shared by several vehicles. It is not mandatory to define a vehicle type. If not given, a default type is used. The driver of a vehicle does not have to be modelled explicitly. For the simulation of persons which walk around or ride in vehicles, additional definitions are necessary.

目录

Vehicles and Routes

Initially, we will define a vehicle with a route owned by him only:

```
<routes>
   <vType id="type1" accel="0.8" decel="4.5" sigma="0.5" length="5" maxSpeed="70"/>

   <vehicle id="0" type="type1" depart="0" color="1,0,0">
      <route edges="beg middle end rend"/>
   </vehicle>

</routes>
```

By giving such a route definition to SUMO (or SUMO-GUI), SUMO will build a red (color=1,0,0) vehicle of type "type1" named "0" which starts at time 0. The vehicle will drive along the streets "beg", "middle", "end", and as soon as it has approached the edge "rend" it will be removed from the simulation.

This vehicle has its own internal route which is not shared with other vehicles. It is also possible to define two vehicles using the same route. In this case the route must be "externalized" - defined before being referenced by the vehicles. Also, the route must be named by giving it an id. The vehicles using the route refer it using the "route"-attribute. The complete change looks like this:

```
<routes>
   <vType id="type1" accel="0.8" decel="4.5" sigma="0.5" length="5" maxSpeed="70"/>

   <route id="route0" color="1,1,0" edges="beg middle end rend"/>

   <vehicle id="0" type="type1" route="route0" depart="0" color="1,0,0"/>
   <vehicle id="1" type="type1" route="route0" depart="0" color="0,1,0"/>

</routes>
```

A vehicle may be defined using the following attributes:

Attribute Name	Value Type	Description
id	id (string)	The name of the vehicle
type	id	The id of the vehicle type to use for this vehicle.
route	id	The id of the route the vehicle shall drive along
color	color	This vehicle's color
depart	float (s) or one of *triggered*, *containerTriggered*	The time step at which the vehicle shall enter the network; see #depart. Alternatively the vehicle departs once a person enters or a container is loaded
departLane	int/string (≥0, "random", "free", "allowed", "best", "first")	The lane on which the vehicle shall be inserted; see #departLane. *default: "first"*
departPos	float(m)/string ("random", "free", "random_free", "base", "last")	The position at which the vehicle shall enter the net; see #departPos. *default: "base"*
departSpeed	float(m/s)/string (≥0, "random", "max", "desired", "speedLimit")	The speed with which the vehicle shall enter the network; see #departSpeed. *default: 0*
arrivalLane	int/string (≥0,"current")	The lane at which the vehicle shall leave the network; see

		#arrivalLane. *default: "current"*
`arrivalPos`	float(m)/string (≥0[1], "random", "max")	The position at which the vehicle shall leave the network; see #arrivalPos. *default: "max"*
`arrivalSpeed`	float(m/s)/string (≥0,"current")	The speed with which the vehicle shall leave the network; see #arrivalSpeed. *default: "current"*
`line`	string	A string specifying the id of a public transport line which can be used when specifying person rides
`personNumber`	int (in [0,personCapacity])	The number of occupied seats when the vehicle is inserted. *default: 0*
`containerNumber`	int (in [0,containerCapacity])	The number of occupied container places when the vehicle is inserted. *default: 0*
`reroute`	bool	Whether the vehicle should be equipped with a rerouting device (setting this to *false* does not take precedence over other assignment options)
`via`	id list	List of intermediate edges that shall be passed on rerouting. **Note:** when *via* is not set, any `<stop>`-elements that belong to this route will automatically be used as intermediate edges. Otherwise *via* takes precedence.
`departPosLat`	float(m)/string ("random", "free", "random_free", "left", "right", "center")	The lateral position on the departure lane at which the vehicle shall enter the net; see Simulation/SublaneModel. *default: "center"*
`arrivalPosLat`	float(m)/string ("left", "right", "center")	The lateral position on the arrival lane at which the vehicle shall arrive; see Simulation/SublaneModel. by default the vehicle does not care about lateral arrival position

Caution:

Any vehicle types or routes referenced by the attributes **type** or **route** must be defined **before** they are used. Loading order is described here.

Repeated vehicles (Flows)

It is possible to define repeated vehicle emissions ("flow"s), which have the same parameters as the vehicle except for the departure time. The id of the created vehicles is "flowId.runningNumber" and they are distributed either equally or randomly in the given interval. The following additional parameters are known:

Attribute Name	Value Type	Description
begin	float(s)	first vehicle departure time
end	float(s)	end of departure interval (if undefined, defaults to 24 hours)
vehsPerHour	float(#/h)	number of vehicles per hour, equally spaced (not together with period or probability)
period	float(s)	insert equally spaced vehicles at that period (not together with vehsPerHour or probability)
probability	float([0,1])	probability for emitting a vehicle each second (not together with vehsPerHour or period), see also Simulation/Randomness
number	int(#)	total number of vehicles, equally spaced

```
<flow id="type1" color="1,1,0"  begin="0" end= "7200" period="900" type="BUS">
    <route edges="beg middle end rend"/>
    <stop busStop="station1" duration="30"/>
</flow>
```

Routes

One may notice, that the route itself also got a color definition, so the attributes of a route are:

Attribute Name	Value Type	Description
id	id (string)	The name of the route
edges	id list	The edges the vehicle shall drive along, given as their ids, separated using spaces
color	color	This route's color

There are a few important things to consider when building your own routes:

- Routes have to be connected. At the moment the simulation raises an error if the next edge of the current route is not a successor of the current edge or if the vehicle is not allowed to drive on any of the lanes. If you want the old behavior where a vehicle simply stopped at the end of the current edge and was possibly "teleported" to the next

edge after a waiting time, use the Option **--ignore-route-errors**.
- Routes have to contain at least one edge.
- The route file has to be sorted by starting times. In fact this is only relevant, when you define a lot of routes or have large gaps between departure times. The simulation parameter **--route-steps**, which defaults to 200, defines the size of the time interval with which the simulation loads its routes. That means by default at startup, only routes with departure times <200 are loaded, if all the vehicles have departed, the routes up to departure time 400 are loaded etc. pp. This works only if the route file is sorted. This behavior may be disabled by specifying --route-steps 0. It is possible to load unsorted route files as an additional file which will load the whole file at once.

The first two conditions can be checked using <***SUMO_HOME***>/tools/route/routecheck.py (https://github.com/eclipse/sumo/blob/master/tools/route/routecheck.py), the third can be "fixed" using <***SUMO_HOME***>/tools/route/sort_routes.py (https://github.com/eclipse/sumo/blob/master/tools/route/sort_routes.py).

Caution:
sumo may enter an infinite loop when given an unsorted route file with person definitions.

Incomplete Routes (trips and flows)

Demand information for the simulation may also take the form of origin and destination edges instead of a complete list of edges. In this case the simulation performs fastest-path routing based on the traffic conditions found in the network at the time of departure/flow begin. Optionally, a list of intermediate edges can be specified with the `via` attribute. The input format is exactly the same as that for the DUAROUTER application and can be found here.

```
<routes>
  <trip id="t" depart="0" from="beg" to="end"/>
  <flow id="f" begin="0" end="100" number="23" from="beg" to="end"/>
  <flow id="f2" begin="0" end="100" number="23" from="beg" to="end" via="e1 e23 e7"/>
</routes>
```

For more details on how to handle routing errors and influence the routing in this case see Demand/Automatic_Routing.

For supported attributes for flows and trips see here.

Traffic assignement zones (TAZ)

It is also possible to let vehicles depart and arrive at traffic assignment zones (TAZ). This allows the departure and arrival edges to be selected from a predefined list of edges. Those edges are used which minimize the travel time from origin TAZ to destination TAZ. When loading trips into DUAROUTER the loaded travel times are used (with empty-network travel times as default). When loading trips into SUMO, the current travel times in the network are used as determined by the rerouting device.

```
<routes>
  <trip id="t" depart="0" fromTaz="taz1" toTaz="taz2"/>
</routes>
```

```
<additional>
   <taz id="<TAZ_ID>" edges="<EDGE_ID> <EDGE_ID> ..."/>
   ...
</additional>
```

Note:
When used in DUAROUTER or SUMO, edge weights within TAZ are ignored.

When loading `<taz>` in SUMO-GUI the optional attribute `shape` can be used to draw an arbitrary polygon border for visualizing the traffic assignment zone.

Caution:
When using TAZ with SUMO and DUAROUTER, their edges will be selected to minimize travel time. This is different from TAZ usage in OD2TRIPS where edges are selected according to a probability distribution.

A Vehicle's depart and arrival parameter

Using the `depart...` and `arrival...`-attributes, it is possible to control how a vehicle is inserted into the network and how it leaves it.

depart

Determines the time at which the vehicle enters the network (for `<flow` the value of **begin** is used instead). If there is not enough space in the network, the actual depart time may be later.

- When using option **--max-depart-delay** ***<TIME>*** the vehicle is discarded if unable to depart after the given delay
- A random offset to the specified depart time is added for each vehicle when using option **--random-depart-offset** ***<TIME>***
- When using the special value *triggered*, the vehicle will depart as soon as a person enters it.

departLane

Determines on which lane the vehicle is tried to be inserted;

- `≥0`: the index of the lane, starting with rightmost=0
- "`random`": a random lane is chosen; please note that a vehicle insertion is not retried if it could not be inserted
- "`free`": the most free (least occupied) lane is chosen
- "`allowed`": the "free" lane (see above) of those lane of the depart edge which allow vehicles of the class the vehicle belongs to
- "`best`": the "free" lane of those who allow the vehicle the longest ride without the need to lane change
- "`first`": the rightmost lane the vehicle may use

BTW, I like "best" at most - dkrajzew

departPos

Determines the position on the chosen departure lane at which the vehicle is tried to be inserted;

- `≥0`: the position on the lane, starting at the lane's begin; must be smaller than the starting lane's length
- `"random"`: a random position is chosen; it is not retried to insert the vehicle if the first try fails
- `"free"`: a free position (if existing) is used
- `"random_free"`: at first, the "random" position is tried, then the "free", if the first one failed
- `"base"`: the vehicle is tried to be inserted at the position which lets its back be at the beginning of the lane (vehicle's front position=vehicle length)
- `"last"`: the vehicle is inserted with the given speed as close as possible behind the last vehicle on the lane. If the lane is empty it is inserted at the end of the lane instead

departSpeed

Determines the speed of the vehicle at insertion;

- `≥0`: The vehicle is tried to be inserted using the given speed. If that speed is unsafe, departure is delayed.
- "`random`": A random speed between 0 and MIN(vehicle's maximum velocity, lane's maximum velocity) is used, the speed may be adapted to ensure a safe distance to the leader vehicle.
- "`max`": The maximum safe velocity (speedLimit * speedFactor)
- "`desired`": The maximum velocity (speedLimit * speedFactor). If that speed is unsafe, departure is delayed.
- "`speedLimit`": The speedLimit is used. If that speed is unsafe, departure is delayed.

arrivalLane

Determines the speed at which the vehicle should end its route;

- "`current`": the vehicle will not change it's lane when nearing arrival. It will use whatever lane is more convenient to reach its arrival position. *(default behavior)*
- `≥0`: the vehicle changes lanes to end it's route on the specified lane

arrivalPos

Determines the position along the destination edge where the vehicle is conisdered to have arrived;

- "`max`": the vehicle will drive up to the end of its final lane. *(default behavior)*
- *<FLOAT>*: the position on the lane, starting at the lane's begin; Negative values count from the end of the lane
- "`random`": a random position is chosen at departure; If vehicle is rerouted a new random position is selected.

arrivalSpeed

Determines the speed at which the vehicle should end its route;

- "`current`": the vehicle will not modify it's speed when nearing arrival. It will drive as fast as (safely) possible. *(default behavior)*
- `≥0`: the vehicle approaches it's arrival position to end with the specified speed

Vehicle Types

A vehicle is defined using the `vType`-element as shown below:

```
<routes>
    <vType id="type1" accel="2.6" decel="4.5" sigma="0.5" length="5" maxSpeed="70"/>
</routes>
```

Having defined this, one can build vehicles of type "type1". The values used above are the ones most of the examples use. They resemble a standard vehicle as used within the Stefan Krauß' thesis.

```
<routes>
    <vType id="type1" accel="2.6" decel="4.5" sigma="0.5" length="5" maxSpeed="70"/>
    <vehicle id="veh1" type="type1" depart="0">
       <route edges="edge1 edge2 edge3"/>
    </vehicle>
</routes>
```

This definition is the initial one which includes both, the definition of the vehicle's "purely physical" parameters, such as its length, its color, or its maximum velocity, and also the used car-following model's parameters. Please note that even though the car-following parameters are describing values such as max. acceleration, or max. deceleration, they mostly do not correspond to what one would assume. The maximum acceleration for example is not the car's maximum acceleration possibility but rather the maximum acceleration a driver choses - even if you have a Jaguar, you probably are not trying to go to 100km/h in 5s when driving through a city.

The default car following model is based on the work of Krauß but other models can be selected as well. Model selection and parameterization is done by setting further `vType`-attribures as shown below. The models and their parameters are described in the following.

```
<routes>
    <vType id="type1" length="5" maxSpeed="70" carFollowModel="Krauss" accel="2.6" decel="4.5"
sigma="0.5"/>
</routes>
```

Available vType Attributes

These values have the following meanings:

Attribute Name	Value Type	Default	Description
id	id (string)	-	The name of the vehicle type

`accel`	float	2.6	The acceleration ability of vehicles of this type (in m/s^2)
`decel`	float	4.5	The deceleration ability of vehicles of this type (in m/s^2)
`apparentDecel`	float	`==decel`	The apparent deceleration of the vehicle as used by the standard model (in m/s^2). The follower uses this value as expected maximal deceleration of the leader.
`emergencyDecel`	float	`==decel`	The maximal physically possible deceleration for the vehicle (in m/s^2).
`sigma`	float	0.5	Car-following model parameter, see below
`tau`	float	1.0	Car-following model parameter, see below
`length`	float	5.0	The vehicle's **netto**-length (length) (in m)
`minGap`	float	2.5	Empty space after leader [m]
`maxSpeed`	float	55.55 (200 km/h) for vehicles, 1.39 (5 km/h) for pedestrians	The vehicle's maximum velocity (in m/s)
`speedFactor`	float	1.0	The vehicles expected multiplicator for lane speed limits
`speedDev`	float	0.0	The deviation of the speedFactor; see below for details
`color`	RGB-color	"1,1,0" (yellow)	This vehicle type's color
`vClass`	class (enum)	"passenger"	An abstract vehicle class (see below). By default vehicles represent regular passenger cars.

`emissionClass`	emission class (enum)	"PC_G_EU4"	An emission class (see below). By default a gasoline passenger car conforming to emission standard *EURO 4* is used.
`guiShape`	shape (enum)	"unknown"	a vehicle shape for drawing. By default a standard passenger car body is drawn.
`width`	float	1.8	The vehicle's width [m] (used only for visualization with the default model, affects sublane model)
`imgFile`	filename (string)	""	Image file for rendering vehicles of this type (should be grayscale to allow functional coloring)
`osgFile`	filename (string)	""	Object file for rendering with OpenSceneGraph (any of the file types supported by the available OSG-plugins)
`laneChangeModel`	lane changing model name (string)	'LC2013'	The model used for changing lanes
`carFollowModel`	car following model name (string)	'Krauss'	The model used for car following
`personCapacity`	int	4	The number of persons (excluding an autonomous driver) the vehicle can transport.
`containerCapacity`	int	0	The number of containers the vehicle

			can transport.
`boardingDuration`	float	0.5	The time required by a person to board the vehicle.
`loadingDuration`	float	90.0	The time required to load a container onto the vehicle.
`latAlignment`	string	*center*	The preferred lateral alignment when using the sublane-model. One of (*left, right, center, compact, nice, arbitrary*).
`minGapLat`	float	0.6	The desired minimum lateral gap when using the sublane-model
`maxSpeedLat`	float	1.0	The maximum lateral speed when using the sublane-model
`actionStepLength`	float	global default (defaults to the simulation step, configurable via **--default.action-step-length**)	The interval length for which vehicle performs its decision logic (acceleration and lane-changing). The given value is processed to the closest (if possible smaller) positive multiple of the simulation step length.

Besides values which describe the vehicle's car-following properties, one can find definitions of the assigned vehicles' shapes, emissions, and assignment to abstract vehicle classes. These concepts will be described in the following. Also, you may find further descriptions of implemented car-following models in the subsection #Car-Following Models.

Speed Distributions

The desired driving speed usually varies among the vehicle of a fleet. In SUMO this is modeled by a speed distribution using the attributes **speedFactor** or **speedDev**. as explained below.

Note:
Since version 1.0.0 speed distributions are used by default (speedDev="0.1"). In older version, speed distributions had to be defined for every vehicle type to avoid homogeneous

speeds (and consequently invalid driving behavior because vehicles would never catch up with their leader vehicle)

Global Configuration

Instead of configuring speed distributions in a `<vType>` definition (as explained below), the SUMO-option **--default.speeddev** *<FLOAT>* can be used to set a global default. Seeting this value to 0 restores pre-1.0.0 behavior.

Defining speed limit violations explicitly

Each vehicle has an individual speed factor which is multiplied with the speed limit (edge speed) to determine the desired driving speed (default 1.0). A vehicle with speed factor 1.2 drives up to 20% above the speed limit whereas a vehicle with speed factor 0.8 would always stay below the speed limit by 20%. By setting attributes **speedFactor** and **speedDev** as show below this individual speed factor for all vehicles of a type can be set to a fixed value.

```
<vType id="example" speedFactor="1.2" speedDev="0"
```

Defining a normal distribution for vehicle speeds

The desired driving speed usually varies among the vehicle of a fleet. While this could be modeled by defining a new type for each vehicle and assigning a distinct speed factor for each type (as above) this would be quite cumbersome. Instead the attribute **speedFactor** can also be used to sample a vehicle specific speed factor from a normal distribution. The parameter can be given as "norm(mean, dev)" or "normc(mean, dev, min, max)". Using **speedFactor**="normc(1,0.1,0.2,2)" will result in a speed distribution where 95% of the vehicles drive between 80% and 120% of the legal speed limit. For flows, every inserted vehicle will draw an individual chosen speed multiplier as well. The resulting values in this example are capped at 20% of speedFactor at the low end to prevent extreme dawdling and at twice the recommended speed. A vehicle keeps its chosen speed multiplier for the whole simulation and multiplies it with edge speeds to compute the actual speed for driving on this edge. Thus vehicles can exceed edge speeds. However, vehicle speeds are still capped at the vehicle type's **maxSpeed**.

Caution:
In order to use mean values below 0.2 or above 2.0, the 4-parameter version must be used to modify the cut-off parameters as well.

Defining a normal distribution (old style)

An alternative way to specify speed distributions is to use numerical values for **speedFactor** and **speedDev**. In this case **speedFactor** defines the expected value and **speedDev** defines the deviation. When using this style, capping cannot be controlled and will always default to 20% and 200%. Thus the above example can also be defined as **speedFactor**="1" **speedDev**="0.1".

Additional remarks on speed distributions

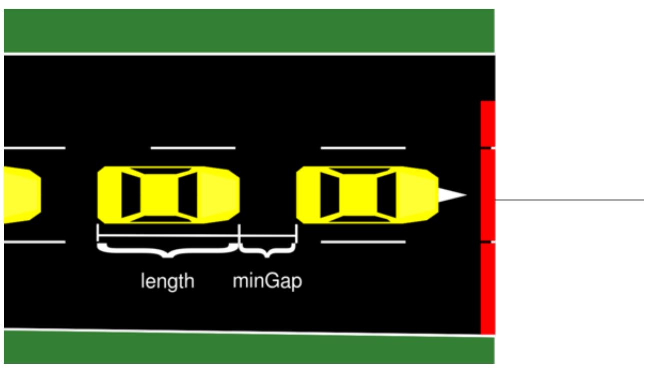
Note:
When used for pedestrians, the *speedFactor* attribute is applied directly to the maximum speed of the vType since speed limits are not applicable to pedestrians

Note:
If the specified departSpeed of a vehicle exceeds the speed limit and it's vType has a speedFactor deviation > 0, the indivial chosen speed multiplier is at least high enough to accommodate the stated depart speed.

Vehicle Length

Due to the work on car following models, we decided to use two values for vehicle length. The `length`-attribute describes the length of the vehicle itself. Additionally, the `minGap`-attribute describes the offset to the leading vehicle when standing in a jam.

This is illustrated in the following image:



Within the simulation, each vehicle needs - when ignoring the safe gap - `length+minGap`. But only `length` of the road should be marked as being occupied.

Abstract Vehicle Class

A SUMO vehicle may be assigned to an "abstract vehicle class", defined by using the attribute `vClass`. These classes are used in lane definitions and allow/disallow the usage of lanes for certain vehicle types. One may think of having a road with three lanes, where the rightmost may only be used by "taxis" or "buses". The default vehicle class is **passenger** (denoting normal passenger cars).

Caution:
Routing or insertion may fail due to a mismatch between a vehicles `vClass` and the road permissions. This can be diagnosed in SUMO-GUI buy highlighting edges according to their permissions.

The following vehicle classes exist:

vClass	bitmask bit	comment
`ignoring`	- (all bits set to 0)	may drive on all lanes regardless of set permissions.
`private`	0	
`emergency`	1	
`authority`	2	
`army`	3	
`vip`	4	
`pedestrian`	5	lanes which only allow this class are considered to be 'sidewalks' in NETCONVERT
`passenger`	6	This is the default vehicle class and denotes regular passenger traffic
`hov`	7	High-occupancy vehicle (https://en.wikipedia.org/wiki/High-occupancy_vehicle_lane)
`taxi`	8	
`bus`	9	urban line traffic
`coach`	10	overland transport
`delivery`	11	Allowed on service roads that are not meant for public traffic
`truck`	12	
`trailer`	13	truck with trailer
`motorcycle`	14	
`moped`	15	motorized 2-wheeler which may not drive on motorways
`bicycle`	16	
`evehicle`	17	future mobility concepts such as electric vehicles which may get special access rights
`tram`	18	
`rail_urban`	19	heavier than 'tram' but distinct from 'rail'. Encompasses Light Rail (http://en.wikipedia.org/wiki/Light_Rail) and S-Bahn (http://en.wikipedia.org/wiki/S-Bahn)
`rail`	20	heavy rail
`rail_electric`	21	heavy rail vehicle that may only drive on electrified tracks
`rail_fast`	22	High-speed-rail (https://en.wikipedia.org/wiki/High-speed_rail)

ship	23	basic class for navigating waterways
custom1	24	reserved for user-defined semantics
custom2	25	reserved for user-defined semantics

These values are a "best guess" of somehow meaningful values, surely worth to be discussed. Though, in parts, they represent classes found in imported formats. They are "abstract" in the means that they are just names only, one could build a .5m long bus.

Note:
vClass values are mainly used for determining access restrictions for lanes and edges. Since version 0.21.0 they will also affect the defaults of some other vType parameters. These defaults are documented at Vehicle_Type_Parameter_Defaults.

The following vehicle deprecated classes exist for maintaining backward compatibility:

deprecated vClass	replacement
public_emergency	deprecated. use 'emergency'
public_authority	deprecated, use 'authority'
public_army	deprecated, use 'army'
public_transport	deprecated, use 'bus'
transport	deprecated, use 'truck'
lightrail	deprecated, use 'tram'
cityrail	deprecated, use 'rail_urban'
rail_slow	deprecated, use 'rail'

Vehicle Emission Classes

The emission class represents a certain emission class. It is defined using the emissionClass attribute. Possible values are given in Models/Emissions and its subsections.

Visualization

For a nicer visualization of the traffic, the appearance of a vehicle type's vehicles may be changed by assigning them a certain shape using the guiShape attribute. These shapes are used when setting the drawing mode for vehicles to **simple shapes**. The following shapes are known:

- "pedestrian"
- "bicycle"
- "motorcycle"
- "passenger"
- "passenger/sedan"

- "passenger/hatchback"
- "passenger/wagon"
- "passenger/van"
- "delivery"
- "truck"
- "truck/semitrailer"
- "truck/trailer"
- "bus"
- "bus/city"
- "bus/flexible" (8.25)
- "bus/overland" (8.25)
- "rail" (24.5)
- "rail/light" (16.85)
- "rail/city" (5.71)
- "rail/slow" (9.44)
- "rail/fast" (24.775)
- "rail/cargo" (13.86)
- "evehicle"
- "ship"

Some of these classes are drawn as a sequence of carriages. The length of a single carriage is indicated in parentheses after the type. For these types, the length of the vehicleType is used as the overall length of the train (all carriages combined). For example, a vehicle with shape `rail/cargo` and length 70m will have 5 carriages. The number of carriages will always be a whole number and no carriage will be shorter than the length given in brackets but may be longer to meet the length requirements of the whole vehicle. When drawing vehicles with raster images, the image will be repeated for each carriage.

In addition, one can determine the width of the vehicle using the attribute `width`. When using shapes, one should consider that different vehicle classes (passenger vehicles or buses) have different lengths. Passenger vehicles with more than 10m length look quite odd, buses with 2m length, too.

Caution:
Not all of these named shapes are implemented.

Car-Following Models

The car-following models currently implemented in SUMO are given in the following table.

Element Name *(deprecated)*	Attribute Value *(when declaring as attribute)*	Description
`carFollowing-Krauss`	Krauss	The Krauß-model with some modifications which is the default model used in SUMO

`carFollowing-KraussOrig1`	KraussOrig1	The original Krauß-model
`carFollowing-PWagner2009`	PWagner2009	A model by Peter Wagner, using Todosiev's action points
`carFollowing-BKerner`	BKerner	A model by Boris Kerner **Caution:** currently under work
`carFollowing-IDM`	IDM	The Intelligent Driver Model by Martin Treiber **Caution:** Default parameters result in very conservative lane changing gap acceptance
`carFollowing-IDMM`	IDMM	Variant of IDMM **Caution:** lacking documentation
`carFollowing-KraussPS`	KraussPS	the default Krauss model with consideration of road slope
`carFollowing-KraussAB`	KraussAB	the default Krauss model with bounded acceleration (only relevant when using PHEM classes)
`carFollowing-SmartSK`	SmartSK	Variant of the default Krauss model **Caution:** lacking documentation
`carFollowing-Wiedemann`	Wiedemann	Car following model by Wiedemann (2-Parameters)
`carFollowing-W99`	W99	Car following model by Wiedemann, 10-Parameter version (http://w99demo.com/)
`carFollowing-Daniel1`	Daniel1	Car following model by Daniel Krajzewicz **Caution:** lacking documentation
`carFollowing-ACC`	ACC	Car following model by Milanés V. and Shladover S.E.
`carFollowing-CACC`	CACC	Car following model by Milanés V. and Shladover S.E.

Car-Following Model Parameters

Mostly, each model uses its own set of parameters. The following table lists which parameter are used by which model(s). Details on car-following models and their parameters can be found here.

Attribute	Description	Models
`minGap`	Minimum Gap when standing (m)	all models
`accel`	The acceleration ability of vehicles of this type (in m/s^2)	SUMOKrauß, SKOrig, PW2009, Kerner, IDM, ACC, CACC
`decel`	The deceleration ability of vehicles of this type (in m/s^2)	SUMOKrauß, SKOrig, PW2009, Kerner, IDM, ACC, CACC
`emergencyDecel`	The maximum deceleration ability of vehicles of this type in case of emergency (in m/s^2)	SUMOKrauß, SKOrig, PW2009, Kerner, IDM, ACC, CACC
`sigma`	The driver imperfection (between 0 and 1)	SUMOKrauß, SKOrig
`tau`	The driver's desired (minimum) time headway. Exact interpretation varies by model. For the default model *Krauss* this is based on the net space between leader back and follower front). For limitations, see Car-Following-Models#tau).	all Models
`minGap`		SUMOKrauß, SKOrig, PW2009, Kerner, IDM
`k`		Kerner
`phi`		Kerner
`delta`	acceleration exponent	IDM
`stepping`	the number of iterations per second when	IDM

	computing follow speed	
`security`	desire for security	Wiedemann
`estimation`	accuracy of situation estimation	Wiedemann
`speedControlGain`	The control gain determining the rate of speed deviation (Speed control mode)	ACC
`gapClosingControlGainSpeed`	The control gain determining the rate of speed deviation (Gap closing control mode)	ACC
`gapClosingControlGainSpace`	The control gain determining the rate of positioning deviation (Gap closing control mode)	ACC
`gapControlGainSpeed`	The control gain determining the rate of speed deviation (Gap control mode)	ACC
`gapControlGainSpace`	The control gain determining the rate of positioning deviation (Gap control mode)	ACC
`collisionAvoidanceGainSpeed`	The control gain determining the rate of speed deviation (Collision avoidance mode)	ACC
`collisionAvoidanceGainSpace`	The control gain determining the rate of positioning deviation (Collision avoidance mode)	ACC
`speedControlGainCACC`	The control gain determining the rate of speed deviation (Speed control mode)	CACC
`gapClosingControlGainGap`	The control gain determining the rate of positioning deviation (Gap closing control mode)	CACC

gapClosingControlGainGapDot	The control gain determining the rate of the positioning deviation derivative (Gap closing control mode)	CACC
gapControlGainGap	The control gain determining the rate of positioning deviation (Gap control mode)	CACC
gapControlGainGapDot	The control gain determining the rate of the positioning deviation derivative (Gap control mode)	CACC
collisionAvoidanceGainGap	The control gain determining the rate of positioning deviation (Collision avoidance mode)	CACC
collisionAvoidanceGainGapDot	The control gain determining the rate of the positioning deviation derivative (Collision avoidance mode)	CACC
CC1	Spacing Time - s	W99
CC2	Following Variation - m	W99
CC3	Threshold for Entering "Following" - s	W99
CC4	Negative "Following" Threshold - m/s	W99
CC5	Positive "Following" Threshold - m/s	W99
CC6	Speed Dependency of Oscillation - 10^-4 rad/s	W99
CC7	Oscillation Acceleration - m/s^2	W99
CC8	Standstill Acceleration - m/s^2	W99
CC9	Acceleration at 80km/h - m/s^2	W99

To select a car following model the following syntax should be used:

```
<vType id="idmAlternative" length="5" minGap="2" carFollowModel="IDM" tau="1.0" .../>
```

Default *Krauss* Model Description

The default model is a modification of the model defined by Stefan Krauß in Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics (http://sumo.dlr.de/pdf/KraussDiss.pdf). The implemented model follows the same idea as that of Krauß, namely: Let vehicles drive as fast as possibly while maintaining perfect safety (always being able to avoid a collision if the leader starts braking within leader and follower maximum acceleration bounds). The implemented model as in <***SUMO_HOME***>/src/microsim/cfmodels/MSCFModel_Krauss.cpp (https://github.com/eclipse/sumo/blob/master/src/microsim/cfmodels/MSCFModel_Krauss.cpp) has the following differences:

- Different deceleration capabilities among the vehicles are handled without violating safety (the original model allowed for collisions in this case)
- The formula for safe velocity was adapted to maintain safety when using the *Euler*-position update rule. This was done by discretizing some of the continuous terms. The original model was defined for the *Ballistic*-position updated rule and would produce collisions when using *Euler*. See also Simulation/Basic_Definition#Defining_the_Integration_Method.

Lane-Changing Models

The lane-changing models currently implemented in SUMO are given in the following table.

Attribute Value	Description
LC2013	The default car following model, developed by Jakob Erdmann based on DK2008 (see SUMO's Lane-Changing Model (http://elib.dlr.de/102254/)). This is the default model.
SL2015	Lane-changing model for sublane-simulation (used by default when setting option **--lateral-resolution** ***<FLOAT>***). This model can only be used with the sublane-extension. **Caution:** This model may technically be used without activating sublane-simulation but this usage has not been fully tested and may not work as expected.
DK2008	The original lane-changing model of sumo until version 0.18.0, developed by Daniel Krajzewicz (see Traffic Simulation with SUMO – Simulation of Urban Mobility (http://link.springer.com/chapter/10.1007/978-1-4419-6142-6_7)).

Mostly, each model uses its own set of parameters. The following table lists which parameter are used by which model(s).

Attribute	Description	Models
`lcStrategic`	The eagerness for performing strategic lane changing. Higher values result in earlier lane-changing. *default: 1.0, range*	LC2013, SL2015

	[0-inf[	
`lcCooperative`	The willingness for performing cooperative lane changing. Lower values result in reduced cooperation. *default: 1.0, range [0-1]*	LC2013, SL2015
`lcSpeedGain`	The eagerness for performing lane changing to gain speed. Higher values result in more lane-changing. *default: 1.0, range [0-inf[*	LC2013, SL2015
`lcKeepRight`	The eagerness for following the obligation to keep right. Higher values result in earlier lane-changing. *default: 1.0, range [0-inf[*	LC2013, SL2015
`lcOvertakeRight`	The probability for violating rules gainst overtaking on the right *default: 0, range [0-1[*	LC2013
`lcOpposite`	The eagerness for overtaking through the opposite-direction lane. Higher values result in more lane-changing. *default: 1.0, range [0-inf[*	LC2013
`lcLookaheadLeft`	Factor for configuring the strategic lookahead distance when a change to the left is necessary (relative to right lookahead). *default: 2.0, range]0-inf[*	LC2013, SL2015
`lcSpeedGainRight`	Factor for configuring the treshold asymmetry when changing to the left or to the right for speed gain. By default the decision for changing to the right takes more deliberation. Symmetry is achieved when set to 1.0. *default: 0.1, range]0-inf[*	LC2013, SL2015
`lcSublane`	The eagerness for using the configured lateral alignment within the lane. Higher values result in increased willingness to sacrifice speed	SL2015

	for alignment. *default: 1.0, range [0-inf]*	
`lcPushy`	Willingness to encroach laterally on other drivers. *default: 0, range 0 to 1*	SL2015
`lcPushyGap`	Minimum lateral gap when encroaching laterally on other drives (alternative way to define lcPushy). *default: minGapLat, range 0 to minGapLat*	SL2015
`lcAssertive`	Willingness to accept lower front and rear gaps on the target lane. The required gap is divided by this value. *default: 1, range: positive reals*	LC2013,SL2015
`lcImpatience`	dynamic factor for modifying lcAssertive and lcPushy. *default: 0 (no effect) range -1 to 1*. Impatience acts as a multiplier. At -1 the multiplier is 0.5 and at 1 the multiplier is 1.5.	SL2015
`lcTimeToImpatience`	Time to reach maximum impatience (of 1). Impatience grows whenever a lane-change manoeuvre is blocked.. *default: infinity (disables impatience growth)*	SL2015
`lcAccelLat`	maximum lateral acceleration per second. *default: 1.0*	SL2015
`lcTurnAlignmentDistance`	Distance to an upcoming turn on the vehicles route, below which the alignment should be dynamically adapted to match the turn direction. *default: 0.0 (i.e., disabled)*	SL2015
`lcMaxSpeedLatStanding`	Upper bound on lateral speed when standing. *default: maxSpeedLat (i.e., disabled)*	SL2015
`lcMaxSpeedLatFactor`	Upper bound on lateral speed while moving computed as	SL2015

		lcMaxSpeedLatStanding + lcMaxSpeedLatFactor * getSpeed(). *default: 1.0*	

The parameters are set within the `<vType>`:

```
<vType id="myType" lcStrategic="0.5" lcCooperative="0.0"/>
```

Junction Model Parameters

The behavior at intersections may be configured with the parameters listed below.

Note:
These parameters are not available in version 0.30.0 and older

Attribute	Value Type	Default	Description
`jmCrossingGap`	float >= 0 (m)	10	Minimum distance to pedestrians that are walking towards the conflict point with the ego vehicle. If the pedestrians are further away the vehicle may drive across the pedestrian crossing.
`jmIgnoreKeepClearTime`	float (s)	-1	The accumulated waiting time (see Option **--waiting-time-memory**) after which a vehicle will drive onto an intersection even though this might cause jamming. For negative values, the vehicle will always try to keep the junction clear.
`jmDriveAfterRedTime`	float (s)	-1	This value causes vehicles to violate a red light if the duration of the red phase is lower than the given threshold. When set to 0, vehicles will always drive at yellow but will try to brake at red. If this behavior causes a vehicle to drive so fast that stopping is not possible any more it will not attempt to stop. This value also applies

			to the default pedestrian model.
`jmDriveAfterYellowTime`	float (s)	-1	This value causes vehicles to violate a yellow light if the duration of the yellow phase is lower than the given threshold. Vehicles that are too fast to brake always drive at yellow..
`jmDriveRedSpeed`	float (m/s)	*maxSpeed*	This value causes vehicles affected by *jmDriveAfterRedTime* to slow down when violating a red light. The given speed will not be exceeded when entering the intersection.
`jmIgnoreFoeProb`	float	0	This value causes vehicles to ignore foe vehicles that have right-of-way with the given probability. The check is performed anew every simulation step. (range [0,1]).
`jmIgnoreFoeSpeed`	float (m/s)	0	This value is used in conjunction with *jmIgnoreFoeProb*. Only vehicles with a speed below or equal to the given value may be ignored.
`jmSigmaMinor`	float, scaling factor (like *sigma*)	sigma	This value configures driving imperfection (dawdling) while passing a minor link (ahead of the intersection after having comitted to drive and while still on the intersection).
`jmTimegapMinor`	float s	1	This value defines the minimum time gap when passing ahead of a prioritized vehicle.
`impatience`	float or 'off'	0.0	Willingess of drivers to impede vehicles with higher priority. See below for semantics.

The parameters are set within the `<vType>`:

```
    <vType id="ambulance" jmDriveAfterRedTime="300" jmDriveAfterRedSpeed="5.56"/>
```

Impatience

The impatience of a driver is value between 0 and 1 that grows whenever the driver has to stop unintentionally (i.e. due to a jam or waiting at an intersection). The impatience value is computed as

```
MAX(0, MIN(1.0, baseImpatience + waitingTime / timeToMaxImpatience))
```

Where baseImpatience is configured by setting the vType-attribute *impatience* and timeToMaxImpatience is set using the option **--time-to-impatience** (default 300s). The value of baseImpatience may negative to slow the growth of the dynamically computed impatience. It may also be defined with the value **off** to prevent drivers from becoming impatient.

The impatience value is used to represent a drivers willingness to impede vehicles with higher priority. At a value of 1 or above, the driver will use any gap that is *safe* in the sense of collision-avoidance even if it means that another vehicle has to brake as hard as it can. At a value of 0, the driver will only perform maneuvers that do not force other vehicles to slow down. Intermediate values interpolate smoothly between these extremes.

Default Vehicle Type

If the `type` attribute of a vehicle is not defined it defaults to `"DEFAULT_VEHTYPE"`. By defining a vehicle type with this id (`<vType id="DEFAULT_VEHTYPE" ..../>`) the default parameters for vehicles without an explicititly defined type can be changed. The change of the default vehicle type needs to occur before any reference to the type was made, so basically before any vehicle or vehicle type was defined. So it should always be at the top of the very first route file.

Route and vehicle type distributions

Instead of defining routes and vTypes explicitly for a vehicle SUMO can choose them at runtime from a given distribution. In order to use this feature just define distributions as following:

Vehicle Type Distributions

```
<routes>
    <vTypeDistribution id="typedist1">
        <vType id="type1" accel="0.8" length="5" maxSpeed="70" probability="0.9"/>
        <vType id="type2" accel="1.8" length="15" maxSpeed="50" probability="0.1"/>
    </vTypeDistribution>
</routes>
```

Note:
The python tool createVehTypeDistributions.py can be used to generate large distributions that vary multiple *vType* parameters independently of each other.

Using existing types

```
<routes>
    <vType id="type1" accel="0.8" length="5" maxSpeed="70" probability="0.9"/>
    <vType id="type2" accel="1.8" length="15" maxSpeed="50" probability="0.1"/>
    <vTypeDistribution id="typedist1" vTypes="type1 type2"/>
</routes>
```

Route Distributions

```
<routes>
    <routeDistribution id="routedist1">
        <route id="route0" color="1,1,0" edges="beg middle end rend" probability="0.9"/>
        <route id="route1" color="1,2,0" edges="beg middle end" probability="0.1"/>
    </routeDistribution>
</routes>
```

A distribution has only an id as (mandatory) attribute and needs a probability attribute for each of its child elements. The sum of the probability values needs not to be 1, they are scaled accordingly. Note, that probability defaults to *1.00* when not specified. At the moment the id for the children is mandatory, this is likely to change in future versions.

A distribution can be used just as using individual types and routes:

```
<routes>
   <vehicle id="0" type="typedist1" route="routedist1" depart="0" color="1,0,0"/>
</routes>
```

Caution:
When using DUAROUTER with input files containing distributions, the output files will contain a fixed route and type for each vehicle and the distributions will be gone. This is to ensure that the each vehicles route will fit its sampled `vClass` when using the input files with SUMO

Stops

Vehicles may be forced to stop for a defined time span or wait for persons by using the stop element either as part of a route or a vehicle definition as following:

```
<routes>
    <route id="route0" edges="beg middle end rend">
        <stop lane="middle_0" endPos="50" duration="20"/>
    </route>
    <vehicle id="v0" route="route0" depart="0">
        <stop lane="end_0" endPos="10" until="50"/>
    </vehicle>
</routes>
```

The resulting vehicle will stop twice, once at lane middle_0 because of the stop defined in its route and the second time because of the stop defined in the vehicle itself. The first stop will last 20 seconds the second one until simulation second 50. For a detailed list of attributes to stops see below. For a description on how to use them to simulate public transport see Simulation/Public Transport.

Stops can be childs of vehicles, routes, persons or containers.

Attribute	Type	Range	Default	Remark
busStop	string	valid busStop ids	-	if given, containerStop, chargingStation, edge, lane, startPos and endPos are not allowed
containerStop	string	valid containerStop ids	-	if given, busStop, chargingStation, edge, lane, startPos and endPos are not allowed
chargingStation	string	valid chargingStation ids	-	if given, busStop, containerStop, edge, lane, startPos and endPos are not allowed
lane	string	lane id	-	the lane id takes the form <edge_id>_<lane_index>. the edge has to be part of the corresponding route
endPos	float(m)	-lane.length < x < lane.length (negative values count backwards from the end of the lane)	lane.length	
startPos	float(m)	-lane.length < x < lane.length (negative values count backwards from the end of the lane)	endPos-0.2m	there must be a difference of more than 0.1m between *startPos* and *endPos*
friendlyPos	bool	true,false	false	whether invalid stop positions should be corrected automatically
duration	float(s)	≥0	-	minimum duration for stopping
until	float(s)	≥0	-	the time step at which the route continues
	int,			

index	"end", "fit"	0≤index≤number of stops in the route	"end"	where to insert the stop in the vehicle's list of stops
triggered	bool	true,false	false	whether a person may end the stop
expected	string	list of person IDs		list of persons that must board the vehicle before it may continue (only takes effect for triggered stops)
expectedContainers	string	list of container IDs		list of containers that must be loaded onto the vehicle before it may continue (only takes effect for triggered stops)
parking	bool	true,false	value of triggered	whether the vehicle stops on the road or beside
actType	string	arbitrary	'waiting'	activity displayed for stopped person in GUI and output files (only applies to person simulation)
tripId	string	arbitrary		parameter to be applied to the vehicle to track the trip id within a cyclical public transport route
line	string	arbitrary		new line attribute to be set on the vehicle when reaching this stop (for cyclical public transport route)

- If "duration" *and* "until" are given, the vehicle will stop for at least "duration" seconds.
- If "duration" is 0 the vehicle will decelerate to reach velocity 0 and then start to accelerate again.
- If "until" is given and "duration" is not and the vehicle arrives at the stop at or after the time step defined by "until" it will decelerate to speed 0 and then accelerate again.
- If persons board the vehicle, the stop is extended by the "boardingDuration" of the vehicle or until the "personCapacity" is reached. (or "loadingDuration" and "containerCapacity" for containers).
- If until is defined in the context of a repeated vehicle insertion (flow) it will be incremented by the difference of vehicle creation time and "begin" of the flow.
- If neither "duration" nor "until" are given, "triggered" defaults to true. If "triggered" is set to false explicitly the vehicle will stop forever.
- if "duration" or "until" are given along with "triggered", then the vehicle will stop until the given duration/until is reached **and** a person has boarded

- If "parking" is set to true. The vehicle stops besides the road without blocking other vehicles.

Caution:
If *triggered* is true then *parking* will also be set to true by default. If you then set *parking* to false you may create deadlocks which prevent the simulation from terminating

Note:
Bus stops must have a length of at least 10

Colors

A color is defined as *red,green,blue* or *red,green,blue,alpha* either in a vehicle, route or vType.

```
<route id="r0" color="0,255,255"/>
<type id="t0" color="0,0,255"/>
<vehicle id="v0" color="255,0,0,0"/>
```

In the default visualization settings the vehicle color will be used if define, otherwise the type and finally the route color. These settings can be changed.

By default color components should be given as integers in the range of (0,255) but other definitions are also supported:

```
color="0.5, 0.5, 1.0"
color="#FF0000"
color="red"
```

The transparency value (alpha) only takes effect when also using the vType attribute *imgFile*.

Devices

Vehicle devices are used to model and configure different aspects such as output (device.fcd) or behavior (device.rerouting).

The following device names are supported and can be used for the placeholder `<DEVICENAME>` below

- emission
- battery
- btreiver
- btsender
- rerouting
- ssm
- toc
- driverstate

- fcd
- example

Automatic assignment

Some devices are assigned automatically. Every <trip> that is loaded into the simulation is automatically equipped with a *rerouting* device to perform the initial route computation.

Other devices such as *fcd* are assigned automatically when the option **--fcd-output** is set.

Assignment by global options

Devices can be configured globally for all vehicles in the simulation by setting the option **--device.<DEVICENAME>.probability** (i.e.) --device.fcd.probability 0.25 This will equip about a quarter of the vehicles with an fcd device (each vehicle determines this randomly with 25% probability) To make the assignment exact the additional option **--device.<DEVICENAME>.deterministic** can be set Another option is to pass the list of vehicle ids that shall be equipped using the option **--device.<DEVICENAME>.explicit <ID1,ID2,...IDk>**.

Note:
These options take precedence over automatic assignment by output-option.

Assignment by generic parameters

Another option for assigning devices for vehicle types or individual vehicles is by using generic parameters. This is done by defining them for the vehicle or the vehicle type in the following way:

```
<routes>
    <vehicle id="v0" route="route0" depart="0">
        <param key="has.<DEVICENAME>.device" value="true"/>
    </vehicle>

    <vType id="t1">
        <param key="has.<DEVICENAME>.device" value="true"/>
    </vType>

    <vehicle id="v1" route="route0" depart="0" type="t1"/>
</routes>
```

Note:
The <param> of a vehicle has precedence over the <param> of the vehicle's type. Both have precedence over the assignment by options.

取自“http://sumo.dlr.de/w/index.php?title=Definition_of_Vehicles,_Vehicle_Types,_and_Routes&oldid=12337”

本页面最后编辑于2019年7月2日 (星期二) 07:36。