SiMTraM – Simulation of Mixed Traffic MObility - User Documentation

Tom V Mathew
Ashutosh Bajpai

SiMTraM – Simulation of Mixed Traffic MObility - User Documentation

Tom V Mathew
Ashutosh Bajpai
$Revision: 001 $

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

What is SiMTraM?

SiMTraM is the extention of SUMO for heterogeneous traffic developed by IIT Bombay, India. SUMO is a microscopic road traffic simulation package. In the near future it will be extended to model other transit modes simultaneously with ordinary car traffic.

Why open source?

Two thoughts stood behind the release of the package as open source. At first the fact that every traffic research organisation is forced to implement an own simulation package; some people are interested in traffic light optimisation, other try to find mistakes made during the design of a road network. Both need some kind of a simulation package and have to implement a framework containing input and output functions and other things from scratch. So the first idea was to give them a basic framework containing all needed methods for a simulation - they can put own ideas into. The second idea is to supply a common test bed for models, especially car models, to the community to make them comparable. Due to different architectures of traffic simulations such comparisons on a wide scale are not possible by now.

Contributors & Participants?

Name	Role	Topics/Contribution
Prof. Tom V Mathew	*Research & Development Support*	*Everything*
Prof. Anirudha Sahoo	*Development Support*	*Strip model*
Ronald Caleb	*Research Support*	*SiMTraM models (Car following & Lane changing)*
Omair Md.	*Development*	*Mid-block Section*
Ashish Cherian	*Development*	*SiMTraM-GUI and MOVE-M*
Ashutosh Bajpai	*Research Support & Development*	*Interface, Intersection and Support for everything*
Pulakesh Upadhyay	Development	TraCI-M and Interface
Sagar chordia	Development	E-1 Detector

Features

• Homogeneous and Heterogeneous
• High portability (using standard - C++ and portable libraries only)
• Collision free vehicle movement
• Different vehicle types
• Single-vehicle routing
• Multi-lane streets with lane/strip changing
• Junction-based right-of-way or left-of-way rules
• Hierarchy of junction types

• A fast openGL graphical user interface
• Dynamic routing
• Manages networks with several 10.000 edges (streets)
• Fast execution speed (up to 100.000 vehicle updates/s on a 1GHz machine)
• Supports import of many network formats (OpenStreetMap, Visum, Vissim, ArcView, XML Descriptions)

About this Document

This document describes how to use each of the applications that come with the SiMTraM-package. We Should remark, that this document only covers the usage of the software and some descriptions of the Used models.

Described Applications

Table 1.1. Applications described within this document

Application	**Application Name (Windows)**	**Application name (Unix)**	**Description**	**Described in chapter**
NETCONVERT-M	inetconvert.exe	isumo-netconvert	A Network Convertor/ Importer	
NETGEN-M	netgen.exe	sumo-netgen	A Generator of Abstract Networks	
DFROUTER-M	dfrouter.exe	sumo-dfrouter	A router using detector flows	
DUAROUTER-M	iduarouter.exe	isumo-duarouter	A router fordynamic user assignment	
JTROUTER-M	jtrouter.exe	sumo-jtrouter	A router using junction turning ratios	
SiMTraM	isumo.exe	isumo	The microscopic simulation	
SiMTraM-gui	iguisim.exe	isumo-guisim	The gui-version of the microscopic	
POLYCONVERT-M	polyconvert.exe	sumo-polyconvert	A tool for importing polygons from other formats	
Others				

Please remark that you may also find the applications "MOVE-M" within the source distribution. A separate user document for MOVE-M is available

Notation

This document uses coloring to differ between different type of information. If you encounter something like this:

```
netconvert --visum=MyVisumNet.inp --output-file=MySUMONet.net.xml
```

you should know that this is a call on the command line. There may be also a '\' at the end of a line. This indicates that you have to continue typing without pressing return (ignoring both the '\' and the following newline). The following example means exactly the same as the one above:

```
netconvert --visum=MyVisumNet.inp \
      --output-file=MySUMONet.net.xml
```

Command line option names are normally coloured `this way`. Their values if optional *<LIKE THIS>*. XML-elements and attributes are shown are coloured `like this`. Their values if optional *<LIKE THIS>*. Complete examples of XML-Files are shown like the following:

```
      <MyType>
            <MyElem myAttr1="0" myAttr2="0.0"/>
            <MyElem myAttr1="1" myAttr2="-500.0"/>
      </MyType>
```

You may also find some notations from the EBNF; brackets '[' and ']' indicate that the enclosed information is optional. Brackets '<' and '>' indicate a type - insert your own value in here... All applications are shown like THIS. *<SUMO_DIST>* is the path you have saved your SiMTraM-package into.

Status

This document is still under development and grows with the software. Due to this, you may find it together with the sources within the SiMTraM repository at IITB (http://www.civil.iitb.ac.in/tvm/SiMTraM_Web/html/). It should always describe the current version.

Call for Help

Please let us know when either the document remains at any point unclear or any of the applications does not behave as expected. We would be very happy if you report broken links or misspelled words. We also seek for some participants and further users, not only to share the development tasks, but also to gain some feedback and critics or some usage examples.

To summarize: every help is appreciated. Thank you.

Chapter 2. First Steps

Installing SiMTraM

Refer to the website: http://www.civil.iitb.ac.in/tvm/SiMTraM_Web/html

Running the Examples

Examples are included in the package. In each example there is a sumo.CFG file. You need to open this file in GUI.

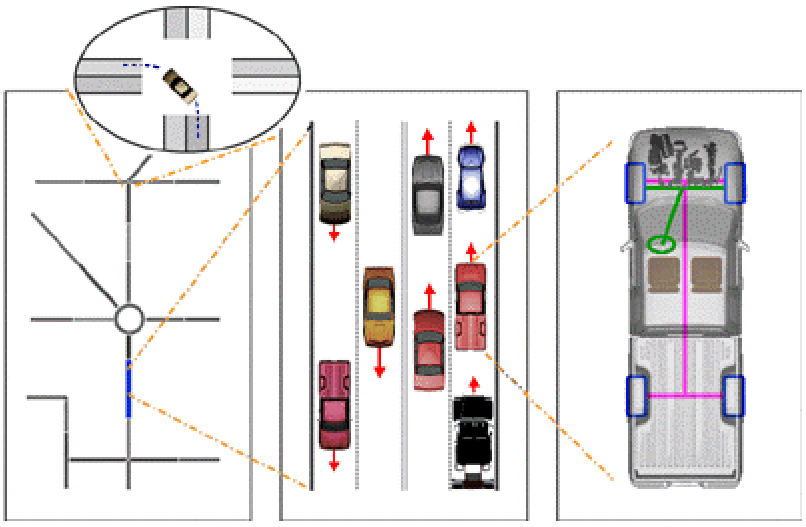
Chapter 3. Traffic Simulations and SiMTraM

A short Introduction to Traffic Simulation Theory

Simulation types

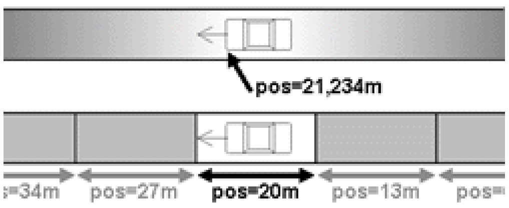
Basic SUMO is a microscopic, space continuous and time discrete traffic simulation. In traffic research four classes of traffic flow models are distinguished according to the level of detail of the simulation. In *macroscopic* models traffic flow is the basic entity. *Microscopic* models simulate the movement of every single vehicle on the street, mostly assuming that the behaviour of the vehicle depends on both, the vehicle's physical abilities to move and the driver's controlling behaviour (see [Chowdhury, Santen, Schadschneider,2000][http://sumo.sourceforge.net/docs/bibliography.shtml#ChowdhurySantenSchadschneider2000]). Within SUMO, the microscopic model developed by Stefan Krauß is used (see [Krauss1998_1][http://sumo.sourceforge.net/docs/bibliography.shtml#Krauss1998_1], [Krauss1998_2][http:// sumo.sourceforge.net/docs/bibliography.shtml#Krauss1998_2]), extended by some further assumptions. *Mesoscopic* simulations are located at the boundary between microscopic and macroscopic simulations. Herein, vehicle movement is mostly simulated using queue approaches and single vehicles are moved between such queues. *Sub-microscopic* models regard single vehicles like microscopic but extend them by dividing them into further substructures, which describe the engine's rotation speed in relation to the vehicle's speed or the driver's preferred gear switching actions, for instance. This allows more detailed computations compared to simple microscopic simulations. However, sub-microscopic models require longer computation times. This restrains the size of the networks to be simulated.

Figure 3.1. The different simulation granularities; from left to right: macroscopic, microscopic, sub-microscopic (within the circle: mesoscopic)



Within a space-continuous simulation each vehicle has a certain position described by a floating-point number. In contrast, space-discrete simulations are a special kind of cellular automata. They use to divide streets into cells and vehicles driving on the simulated streets "jump" from one cell to another.

Figure 3.2. The difference between a space-continuous (top) and a space-discrete (bottom) simulation



Almost every simulation package uses an own model for vehicle movement. Almost all models are so-called "*car-following-models*": the behaviour of the driver is herein meant to be dependent on his distance to the vehicle in front of him and of this leading vehicle's speed. Although SUMO is meant to be a test bed for such vehicle models, only one is implemented by now, an extension to the one developed by Stefan Krauß. Other obstacles, such as traffic lights, are of course considered herein, too. It seems obvious, that each driver is trying to use to shortest path through the network. But when all are trying to do this, some of the roads - mainly the arterial roads - would get congested reducing the benefit of using them. Solutions for this problem are known to traffic research as *dynamic user assignment*. For solving this, several approaches are available and SUMO uses the dynamic user assignment approach developed by Christian Gawron (see [Gawron1998_1] [http://sumo.sourceforge.net/docs/ bibliography.shtml#Gawron1998_1]).

Most of the Model from basic SUMO still works in SiMTram. A slightly changed car-following and lane changing model can be found at the website.

Needed Data

At first, you need the network the traffic to simulate takes place on. As SUMO is meant to work with large networks, we mainly concentrated our work on importing networks and the computation of further needed values. Due to this, no graphical editor for networks is available, yet. Beside information about a network's roads, information about traffic lights is needed. Further, you need information about the traffic demand. While most traffic simulation use a statistical distribution which is laid over the network, each vehicle within SiMTraM knows its route like SUMO . Within this approach, the route is a list of edges to pass. Although this approach is more realistic, it also induces a large amount of data needed to describe the vehicle movements. By now, routes are not compressed within SUMO and so may be several MB large. We will possibly change this in future.

The Workflow of Preparing a Simulation

As shortly described above, you basically have to perform the following steps in order to make your simulation run:

1. Build your network

   Use either own descriptions (described in chapter 4, "Building Networks from own XML-descriptions or if you have some digital networks SUMO can import, convert them (described in chapter 4, "Converting other Input Data)

2. Build the demand

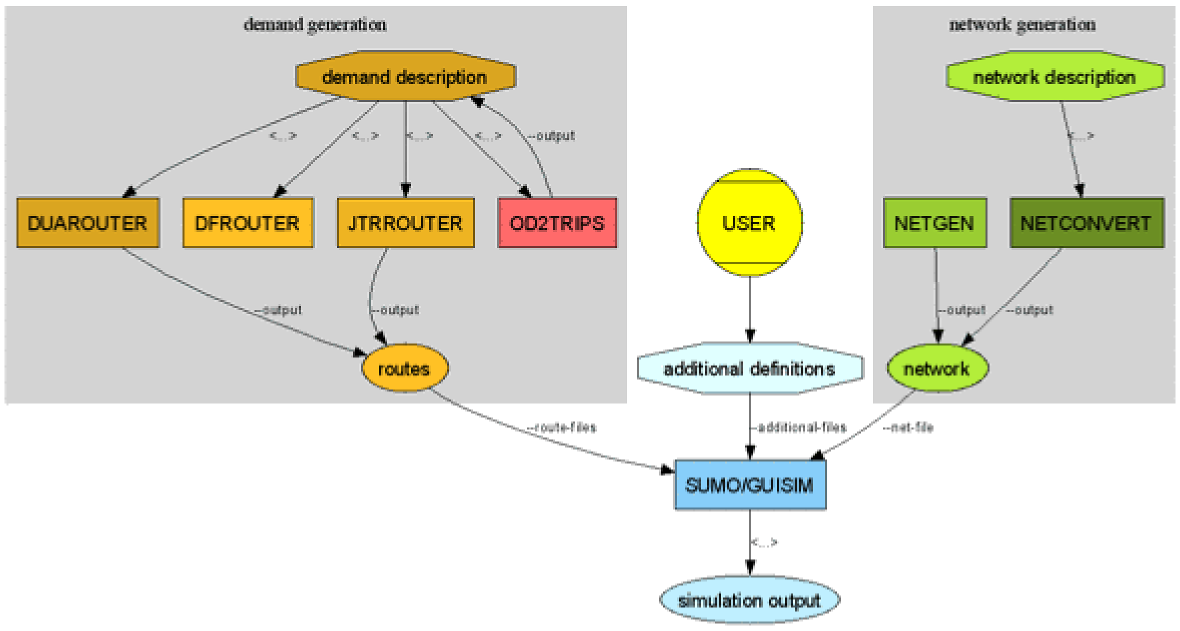
   Build your own movements using either by a) describing explicit vehicle routes (see chapter 5, "Using Trip Definitions"), b) using flows and turning percentages only (see chapter 5, "Using the

Junction Turning Ratio - Router"), c) generating random routes (see chapter 5, "Generating random Routes").

3. If needed, compute the dynamic user assignment (described in chapter 5, "Dynamic User Assignment")

4. Perform the simulation (described in chapter 6, "Performing the Simulation“) to get your desired output This process is also visualised within the next figure.

Figure 3.3. Process of simulation with SiMTraM; (rounded: definite data types; boxes: applications; octagons: abstract data types) (Similar as SUMO)



Please remark, that most of the tools are command-line tools by now. They do nothing if you just double-click them (besides printing errors). Do also notice, that the call parameter desribed in the following chapters may be also stored in so-called "*configuration files*" to allow their reuse. This possibility is described in chapter "Using Configuration Files [http://sumo.sourceforge.net/docs/gen/user_chp08.shtml#user_chp08-configs]".

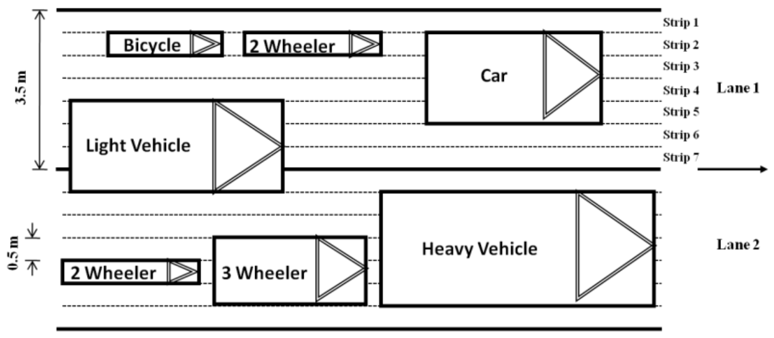
SiMTraM

The traditional car-following models look at the vehicles in the current lane. However, in strip-based movement, a vehicle may occupy more than one strip and so it may have more than one vehicle in front of it (or behind it). So, to take into account this basic difference, a longitudinal movement model was built. The conventional lane-based lane changing model looks at the left and right lane as candidate lanes for changing the lane. However, in strip based model, a vehicle needs to look at and beyond the strips and should be able to calculate the benefit a driver will get due to a strip change/s. These aspects are taken care of by the newly developed lateral movement model. These features of the model are elaborated in the subsequent sections.

Strip-based Representation

The road space is divided into strips, with each lane having an integral number of strips (Figure 1). The strip width is configurable and is related to the lane width. If fine-grained simulation is required, the strip width can be made smaller. In fact, it can even approximate a coordinate-based model with a strip size of 0.1 meter or less. The usual strip size is taken to be the width of the smallest vehicle being simulated.

FIGURE 3.4 various vehicles on a mid-block with a typical lane width



Main Software Paradigms

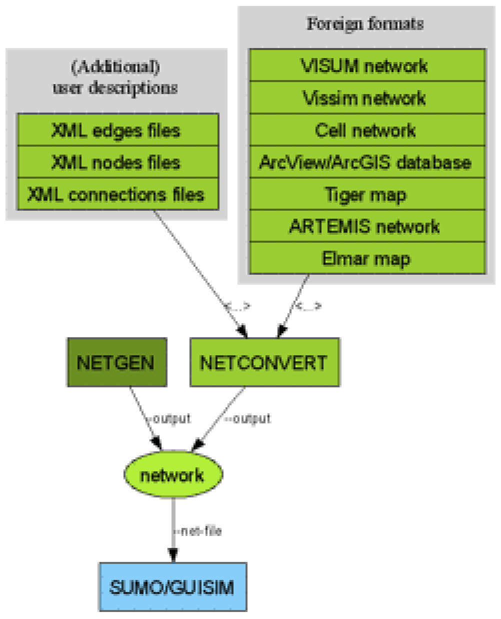
Two basic design goals are approached: the software shall be fast and it shall be portable. Due to this, the very first versions were developed to be run from the command line only - no graphical interface Traffic Simulations and SUMO was supplied at first and all parameter had to be inserted by hand. This should increase the execution speed by leaving off slow visualisation. Also, due to these goals, the software was split into several parts. Each of them has a certain purpose and must be run individually. This is something that makes SUMO different to other simulation packages where the dynamical user assignment is made within the simulation itself, not via an external application like here. This split allows an easier extension of each of the applications within the package because each is smaller than a monolithic application doing everything. Also, it also allows the usage of faster data structures, each adjusted to the current purpose, instead of using complicated and ballast-loaded ones. Still, this makes the usage of SUMO a little bit uncomfortable in comparison to other simulation packages. As there are still other things to do, we are not thinking of a redesign towards an integrated approach by now.

Chapter 4. Network Generation

Introduction

As SiMTraM uses an own road network description, networks must be converted from an existing dataset. Although being readable (xml) by human beings, the format of road networks used by SiMTraM is not meant to be edited by hand and will also not be described herein due to its complexity. SiMTraM networks can be build by either converting an existing map or by using NETGEN-M to generate basic, abstract road maps. The following figure shows the function of NETCONVERT-M and NETGEN-M within the procedure of building and running a simulation.

Figure 4.1. Building a network



Having data describing a road network, you may convert them into a network description readable by SiMTraM using the NETCONVERT-M tool. By now, NETCONVERT-M is capable to parse the following formats:

- *ptv VISUM* (a macroscopic traffic simulation package), see chapter "Importing VISUM-networks [#user_chp04-other_input-visum]"
- *ptv VISSIM* (a microscopic traffic simulation package), see chapter "Importing VISSIM-networks [#user_chp04-other_input-vissim]"
- *ArcView*-data base files, see chapter "Importing ArcView-databases [#user_chp04-other_input]"
- *XML*-descriptions, see chapter "Building Networks from own XML-descriptions [#user_chp04-xml_descriptions]"
- Elmar Brockfelds *unsplitted* and *splitted NavTeq-data*, see chapter "Importing Elmar's converted NavTech-Files [#user_chp04-other_input-elmar]"
- *TIGER* databases, see chapter "Importing TIGER-databases [#user_chp04-other_input-tiger]"

In most of these cases, NETCOVERT-M needs only two parameter: the option named as the source application/format followed by the name of the file to convert and the name of the output file (using the `--output-file` option). So if you want to import a file generated by the VISUM simulation package, simply write the following:

```
Netconvert-M --visum=MyVisumNet.inp -output file=MySUMONet.net.xml
```

The parameter `--output-file` has the default value "`net.net.xml`". That means that both NETCONVERT-M and NETGEN-M will save the generated file as "`net.net.xml`" if the option is not set. Please note, that NETCONVERT-M has to be started from the command line. There is no graphical interface available, yet. The following subchapters will describe more deeply how NETCONVERT-M and NETGEN-M are used, also discussing some problems with each of the import formats NETCONVERT-M supports. Please remind the option to name the output generated by both applications:

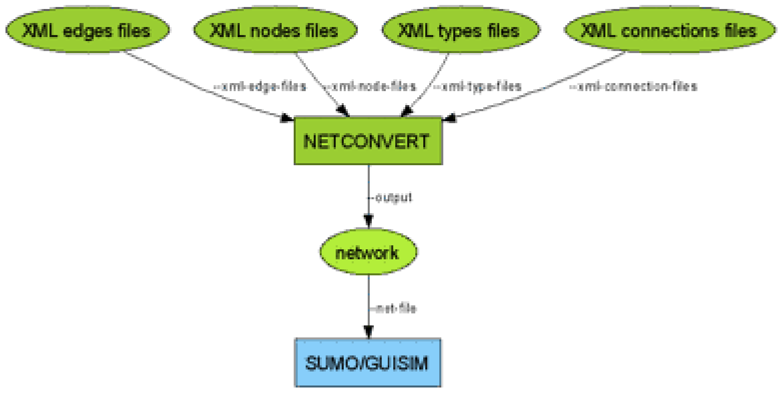
```
(--output-file| --output | -o ) <OUTPUT_FILE>
```

Defines the file to write the computed network into. This file will contain the generated network if the conversion could be accomplished. *Optional (pregiven), type:filename, default: "net.net.xml"*

Building Networks from own XMLdescriptions

All examples within the distribution were made by hand. For doing this, you need at least two files: one file for nodes and another one for the streets between them. Please notice that herein, "*node*" and "*junction*" mean the same as well as "*edge*" and "*street*" do. Besides defining the nodes and edges, you can also join edges by type and set explicit connections between lanes. We will describe how each of these four file types should look like in the following chapters.

Figure 4.2. Building a network from XML-descriptions



Nodes Descriptions

Within the nodes-files, normally having the extension "`.nod.xml`" (see Appendix "Naming Conventions [http://sumo.sourceforge.net/docs/gen/user_apa.shtml]"), every node is described in a Network Generation single line which looks like this: `<node id="`*`<STRING>`*`" x="`*`<FLOAT>`*`" y="`*`<FLOAT>`*`" [type="`*`<TYPE>`*`"]/>` - the straight brackets ('[' and ']') indicate that the parameter is optional. Each of these attributes has a certain meaning and value range:

- `id`: The name of the node; may be any character string
- `x`: The x-position of the node on the plane in meters; must be a floating point number
- `y`: The y-position of the node on the plane in meters; must be a floating point number

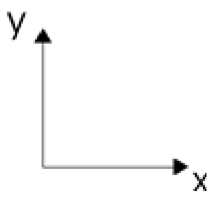
- `type`: An optional type for the node. If you leave out the type of the node, it is automatically guessed by NETCOVERT-M but may not be the one you intentionally thought of. The following types are possible, any other string is counted as an error and will yield in a program stop:
    - `priority`: Vehicles have to wait until vehicles right to them have passed the junction.
    - `traffic_light`: The junction is controlled by a traffic light.

When writing your nodes-file, please do not forget to embed your node definitions into an opening and a closing "tag". A complete file should like the example below, which is the node file `"cross3l.nod.xml"` for the examples `"<SUMO_DIST>/data/examples/netbuild/types/cross_usingtypes/"` and `"<SUMO_DIST>/data/examples/netbuild/types/cross_notypes/"` example.

```
<nodes> <!-- The opening tag -->

<node id="0" x="0.0" y="0.0" type="traffic_light"/> <!-- def. of node
"0" -->
<node id="1" x="-500.0" y="0.0" type="priority"/> <!-- def. of node
"1" -->
<node id="2" x="+500.0" y="0.0" type="priority"/> <!-- def. of node
"2" -->
<node id="3" x="0.0" y="-500.0" type="priority"/> <!-- def. of node
"3" -->
<node id="4" x="0.0" y="+500.0" type="priority"/> <!-- def. of node
"4" -->
<node id="m1" x="-250.0" y="0.0" type="priority"/> <!-- def. of node
"m1" -->
<node id="m2" x="+250.0" y="0.0" type="priority"/> <!-- def. of node
"m2" -->
<node id="m3" x="0.0" y="-250.0" type="priority"/> <!-- def. of node
"m3" -->
<node id="m4" x="0.0" y="+250.0" type="priority"/> <!-- def. of node
"m4" -->

</nodes> <!-- The closing tag -->
```

As you may notice, only the first node named "0", which is the node in the middle of the network, is a traffic light controlled junction. All other nodes are uncontrolled. You may also notice, that each of both ends of a street needs an according node. This is not really necessary as you may see soon, but it eases the understanding of the concept: every edge (street/road) is a connection between two nodes (junctions). You should also know something about the coordinate system: the higher a node on the screen shall be (the nearer to the top of your monitor), the higher his y-value must be. The more to left it shall be, the higher his x-value.

Figure 4.3. Coordinate system used in SiMTraM



Since version 0.9.4 in SUMO you can also give the `x`- and `y`-coordinates using geocoordinates. In this case, the coordinates will be interpreted as long/lat in degrees. Read more on this in "Converting from Geocoordinates".

Edges Descriptions

Edges are described quite the same way as nodes, but posses other parameter. Within the edges file, each description of a single edge looks like this: `<edge id="`*`<STRING>`*`" (fromnode="`*`<NODE_ID>`*`" tonode="`*`<NODE_ID>`*`" | xfrom="`*`<FLOAT>`*`" yfrom="`*`<FLOAT>`*`" xto="`*`<FLOAT>`*`" yto="`*`<FLOAT>`*`") [type="`*`<STRING>`*`" | nolanes="`*`<INT>`*`" speed="`*`<FLOAT>`*`" priority="`*`<UINT>`*`" length="`*`<FLOAT>`*`")] [shape="`*`<2D_POINT>`*` [ `*`<2D_POINT>`*`]*"] [spread_type="center"]/>`.

What does it mean? Every one who knows how XML-files look like should have noticed brackets ('(' and ')') and pipes ('|') within the definition and these characters are not allowed within XML... What we wanted to show which parameter is optional. So for the definition of the origin and the destination node, you can either give their names using `fromnode="`*`<NODE_ID>`*`" tonode="`*`<NODE_ID>`*`"` or you give their positions using `xfrom="`*`<FLOAT>`*`" yfrom="`*`<FLOAT>`*` xto="`*`<FLOAT>`*`" yto="`*`<FLOAT>`*`"`. In the second case, nodes will be build automatically at the given positions. Each edge is unidirectional and starts at the "`from`"-node and ends at the "`to`"-node. If a name of one of the nodes can not be dereferenced (because they have not been defined within the nodes file) an error is generated (see also the documentation on "`--dismiss-loading-errors`" in subchapter "Building the Network").

For each edge, some further attributes should be supplied, being the number of lanes the edge has, the maximum speed allowed on the edge, the length the edge has (in meters). Furthermore, the priority may be defined optionally. All these values - beside the length in fact - may either be given for each edge using according attributes or you can omit them by giving the edge a "`type`". In this case, you should also write a type-file (see subchapter "Types Descriptions [#user_chp04-xml_descriptions-types]"). A type with this name should of course be within the generated type-file, otherwise an error is reported. Even if you supply a type, you can still override the type's values by supplying any of the parameter `nolanes`, `speed` and `priority`. You may also leave the edge parameter completely unset. In this case, default-values will be used and the edge will have a single lane, a default (unset) priority and the maximum allowed speed on this edge will be 13.9m/s being around 50km/h. The length of this edge will be computed as the distance between the starting and the end point.

As an edge may have a more complicated geometry, you may supply the edge's shape within the `shape` tag. If the length of the edge is not given otherwise, the distances of the shape elements will be summed. The information `spread_type="center"` forces NETCONVERT-M to spread lanes to both sides of the connection between the begin node and the end node or from the list of lines making up the shape. If not given, lanes are spread to right, as default.

Let's list an edge's attributes again:

- `id`: The name of the edge; may be any character string
- Origin and destination node descriptions
  Either:
  - `fromnode`: The name of a node within the nodes-file the edge shall start at
  - `tonode`: The name of a node within the nodes-file the edge shall end at

  or:
  - `xfrom`: The x-position of the node the edge shall start at in meters; must be a floating point number
  - `yfrom`: The y-position of the node the edge shall start at in meters; must be a floating point number
  - `xto`: The x-position of the node the edge shall end at in meters; must be a floating point number

  - `yto`: The y-position of the node the edge shall end at in meters; must be a floating point number

- Descriptions of the edge's type and atomic attributes:
  Either:
  - `type`: The name of a type within the types-file

  or/and:
  - `nolanes`: The number of lanes of the edge; must be an integer value
  - `speed`: The maximum speed allowed on the edge in m/s; must be a floating point number (see also "Using Edges' maximum Speed Definitions in km/h" [#user_chp04-further_optionskmh_ speed])
  - `priority`: The priority of the edge; must be a positive integer value
  - `length`: The length of the edge in meter; must be an float value
- The edges shape:
  - `shape`: List of positions; each position is encoded in x,y (do not separate the numbers with a space!) in meters; the start and end node are omitted from the shape definition; an example: `<edge id="e1" fromnode="0" tonode="1" shape="0,0 0,100"/>` describes an edge that after starting at node 0, first visits position 0,0 than goes one hundred meters to the right before finally reaching the position of node 1.
  - `spread_type`: The description of how to spread the lanes; "`center`" spreads lanes to both directions of the shape, any other value will be interpreted as "`right`".

The priority plays a role during the computation of the way-giving rules of a node. Normally, the allowed speed on the edge and the edge's number of lanes are used to compute which edge has a greater priority on a junction. Using the priority attribute, you may increase the priority of the edge making more lanes yielding in it or making vehicles coming from this edge into the junction not wait.

Also the definitions of edges must be embedded into an opening and a closing tag and for the example "*<SUMO_DIST>*`/data/examples/netbuild/types/cross_notypes/`" the whole edges-file looks like this ("`cross3l.edg.xml`"):

```
<edges>
      <edge id="1fi" fromnode="1" tonode="m1" priority="2"
      nolanes="2" speed="11.11"/>
      <edge id="1si" fromnode="m1" tonode="0" priority="3"
      nolanes="3" speed="13.89"/>
      <edge id="1o" fromnode="0" tonode="1" priority="1" nolanes="1"
      speed="11.11"/>
      <edge id="2fi" fromnode="2" tonode="m2" priority="2"
      nolanes="2" speed="11.11"/>
      <edge id="2si" fromnode="m2" tonode="0" priority="3"
      nolanes="3" speed="13.89"/>
      <edge id="2o" fromnode="0" tonode="2" priority="1" nolanes="1"
      speed="11.11"/>
      <edge id="3fi" fromnode="3" tonode="m3" priority="2"
      nolanes="2" speed="11.11"/>
      <edge id="3si" fromnode="m3" tonode="0" priority="3"
      nolanes="3" speed="13.89"/>
      <edge id="3o" fromnode="0" tonode="3" priority="1" nolanes="1"
      speed="11.11"/>
      <edge id="4fi" fromnode="4" tonode="m4" priority="2"
      nolanes="2" speed="11.11"/>
      <edge id="4si" fromnode="m4" tonode="0" priority="3"
      nolanes="3" speed="13.89"/>
      <edge id="4o" fromnode="0" tonode="4" priority="1" nolanes="1"
      speed="11.11"/>
</edges>
```

Within this example, we have used explicit definitions of edges. An example for using types is described in the chapter "Types Descriptions [#user_chp04-xml_descriptions-types]".

Caution

There are some constraints about the streets' ids. They must not contain any of the following characters: '_' (underline - used for lane ids), '[' and ']' (used for enumerations), ' ' (space – used as list divider), '*' (star, used as wildcard), ':' (used as marker for internal lanes).

Recent changes:

- The `function`-tag was added for version 0.9.4 and was revalidated for version 0.9.5
- 11.03.2008: False documentation updated: `--omit-corrupt-edges` is outdated; use `--dismiss-loading-errors` instead
- The `function`-tag was removed for version 0.9.9; a warning is generated when this attribute is used

Defining allowed Vehicle Types

Since version 0.9.5 you may allow/forbid explicite vehicle classes to use a lane. The information which vehicle classes are allowed on a lane may be specified within an edges descriptions file by embedding the list of lanes together with vehicle classes allowed/forbidden on them into these lanes' edge. Assume you want to allow only busses to use the leftmost lane of edge "2si" from the example above. Simply change this edge's definition into:

```
... previous definitions ...
      <edge id="2si" fromnode="m2" tonode="0" priority="3"
      nolanes="3" speed="13.89">
      <lane id="2" allow="bus"/>
      </edge>
... further definitions ...
```

If you would like to disallow passenger cars and taxis, the following snipplet would do it:

```
... previous definitions ...
      <edge id="2si" fromnode="m2" tonode="0" priority="3"
      nolanes="3" speed="13.89">
      <lane id="2" disallow="passenger;taxis"/>
      </edge>
... further definitions ...
```

The definition of a lane contains by now the following attributes:

- `id`: The enumeration id of the lane (0 is the rightmost lane, *`<NUMBER_LANES>`*-1 is the leftmost one)
- `allow`: The list of explicitely allowed vehicle classes
- `disallow`: The list of explicitely disallowed vehicle classes

Both the allowed and the disallowed attributes assume to get a list of vehicle class names devided by a ';'. See "Vehicle Classes [http://sumo.sourceforge.net/docs/gen/user_chp06.shtml#user_chp06-management-vclasses]" for further information about allowed vehicle classes and their usage.

Caution

This is a new feature. Its usage and the way it works will surely change in the future.

Examples: none yet
Recent changes:

- The possibility to define which vehicle classes are allowed on a lane was added in version 0.9.5

Types Descriptions

As mentioned, road types are meant to be used to ease the definition of edges. As described above, the description of an edge should include information about the number of lanes, the maximum speed allowed on this edge and the edge's priority. To avoid the explicit definition of each parameter for every edge, one can use road types, which encapsulate these parameter under a given name. The format of this definition is: `<type id="<STRING>" nolanes="<INT>" speed="<FLOAT>" priority="<UINT>"/> numstrips="<UINT>"`. The attributes of a type are of course exactly the same as for edges themselves:

- `id`: The name of the road type; may be any character string
- `nolanes`: The number of lanes of the referencing must be an integer value
- `speed`: The maximum speed allowed on the referencing edge in m/s; must be a floating point number
- `priority`: The priority of the referencing edge; must be a positive integer value
- `numstrips` : number of strips a lane can have in that edge; must be an integer value

The information about the nodes the edge starts and ends at is not given within the types' descriptions. They can only be set within the edge's attributes. Here's an example on referencing types in edge definitions:

```
<edges>

      <edge id="1fi" fromnode="1" tonode="m1" type="b"/>
      <edge id="1si" fromnode="m1" tonode="0" type="a"/>
      <edge id="1o" fromnode="0" tonode="1" type="c"/>
      <edge id="2fi" fromnode="2" tonode="m2" type="b"/>
      <edge id="2si" fromnode="m2" tonode="0" type="a"/>
      <edge id="2o" fromnode="0" tonode="2" type="c"/>
      <edge id="3fi" fromnode="3" tonode="m3" type="b"/>
      <edge id="3si" fromnode="m3" tonode="0" type="a"/>
      <edge id="3o" fromnode="0" tonode="3" type="c"/>
      <edge id="4fi" fromnode="4" tonode="m4" type="b"/>
      <edge id="4si" fromnode="m4" tonode="0" type="a"/>
      <edge id="4o" fromnode="0" tonode="4" type="c"/>

</edges>
```

The according types file looks like this:

```
<types>

      <type id="a" priority="3" nolanes="3" speed="13.889" numstrips
      =“3”/>
      <type id="b" priority="2" nolanes="2" speed="11.111" numstrips
      = “4”/>
      <type id="c" priority="1" nolanes="1" speed="11.111"/>

</types>
```

As you can see, we have joined the edges into three classes "a", "b", and "c" and have generated a description for each of these classes. Doing this, the generated net is similar to the one generated using the settings described above (example "*<SUMO_DIST>*/data/examples/netbuild/types/cross_notypes/").

Examples:

- The basic usage of types is shown in *<SUMO_DIST>*/data/examples/netbuild/types/cross_notypes/ where the same network is constructed once not using types (subfolder "cross_notypes") and once using them (subfolder "cross_usingtypes").

Recent changes:

- The function-tag was added for version 0.9.5
- The function-tag was removed for version 0.9.9; a warning is generated when this attribute is Used

Connection Descriptions

Explicite setting which Edge / Lane is connected to which

If you have tried the version 0.7 you have possibly missed the possibility to specify the connections between the edges for yourself. This is now possible using a further file, the connections file. The connection file specifies which edges outgoing from a junction may be reached by a certain edge incoming into this junction and optionally also which lanes shall be used on both sides.

If you only want to describe which edges may be reached from a certain edge, this definition could look something like this: <connection from="*<FROM_EDGE_ID>*" to="*<T0_EDGE_ID>*"/>. This tells NETCONVERT not only that vehicles shall be allowed to drive from the edge named *<FROM_EDGE_ID>* to the edge named *<TO_EDGE_ID>*, but also prohibits all movements to other edges from *<FROM_EDGE_ID>*, unless they are specified within this file. Let's repeat the parameters:

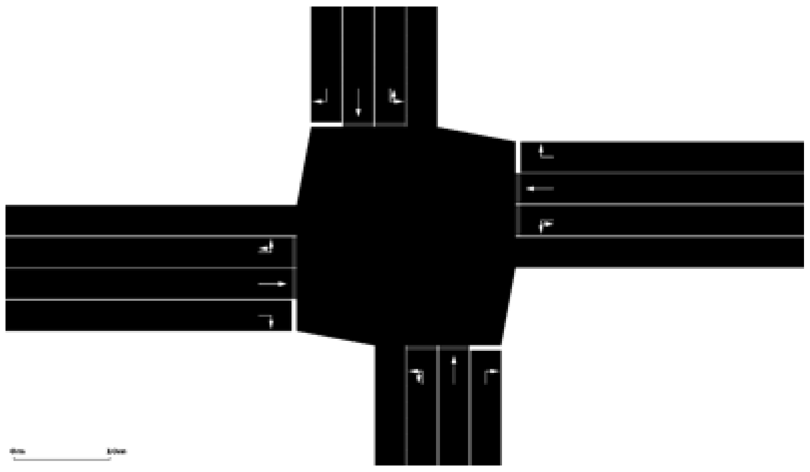
- from: The name of the edge the vehicles leave
- to: The name of the edge the vehicles may reach when leaving "from"

When using this kind of input, NETCONVERT-M will compute which lanes shall be used if any of the connected edges has more than one lane. If you also want to override this computation and set the lanes by hand, use the following: <connection from="<FROM_EDGE_ID>" to="<T0_EDGE_ID>" lane="<INT_1>:<INT_2>"/>. Here, a connection from the edge's "*<FROM_EDGE_ID>*" lane with the number *<INT_1>* is build to the lane *<INT_2>* of the edge "*<TO_EDGE_ID>*". Lanes are counted from the right (outer) to the left (inner) side of the road beginning with 0. Again the parameter:

- from: The name of the edge the vehicles leave
- to: The name of the edge the vehicles may reach when leaving "from"
- lane: the numbers of the connected lanes, separated with ':'; lanes are counter from right to left beginning with 0

There are two examples within the distribution. Both use the nodes and edges descriptions from the example located in "*<SUMO_DIST>*/data/examples/netbuild/types/cross_notypes/". The junction in the center of this example looks like shown within the next figure. We will now call it the "unconstrained network" because all connections and turnarounds are computed using the default values.

Figure 4.4. Unconstrained Network (zoom=2200)

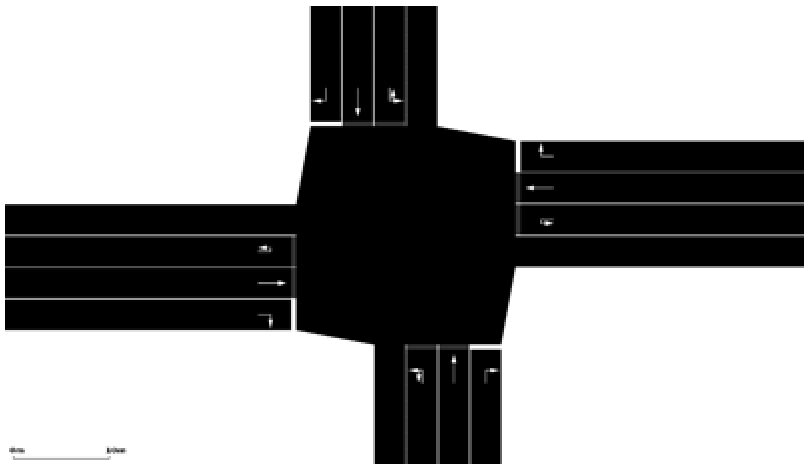


The example <*SUMO_DIST*>`/data/examples/netbuild/connections/cross3l_edge2edge_conns/`" shows what happens when one uses connections to limit the number of reachable edges. To do this we built a connections file where we say that the horizontal edges ("1si" and "2si") have only connections to the edges right to them and the edge in straight direction. The file looks like this:

```
<connections>

      <connection from="1si" to="3o"/>
      <connection from="1si" to="2o"/>
      <connection from="2si" to="4o"/>
      <connection from="2si" to="1o"/>

</connections>
```

As you may see in the next picture, the horizontal edges within the result network contain no leftmoving connections.

Figure 4.5. Network with explicit edge-2-edge connections

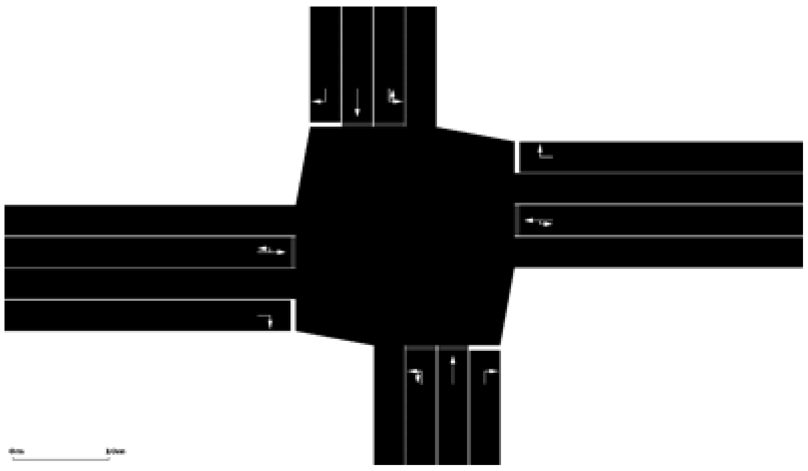


In the second example located in *<SUMO_DIST>*`/data/examples/netbuild/connections/cross3l_laneslane_conns/`" we additionally describe which lanes shall be connected. The according connections file says that the connections going straight shall be start at the second lane of the incoming edges:

```
<connections>

      <connection from="1si" to="3o" lane="0:0"/>
      <connection from="1si" to="2o" lane="2:0"/>
      <connection from="2si" to="4o" lane="0:0"/>
      <connection from="2si" to="1o" lane="2:0"/>

</connections>
```

The built network looks like this:

Figure 4.6. Network with explicit lane-2-lane connections



Warning

Please do not use both types of connection declarations (those with an `lane` attribute and those without) for the same from-edge! The behaviour is not verified and tested for these settings. Examples (compare both to

<SUMO_DIST>/data/examples/netbuild/connections/cross3l_unconstrained/)
:

- *<SUMO_DIST>*/data/examples/netbuild/connections/cross3l_edge2edge_conns/ shows how edge-to-edge connections may be specified
- *<SUMO_DIST>*/data/examples/netbuild/connections/cross3l_lane2lane_conns/ shows how lane-to-lane connections may be specified

Recent Changes:

- A bug which sometimes yielded in a reassignment of connections is patched in version 0.9.3

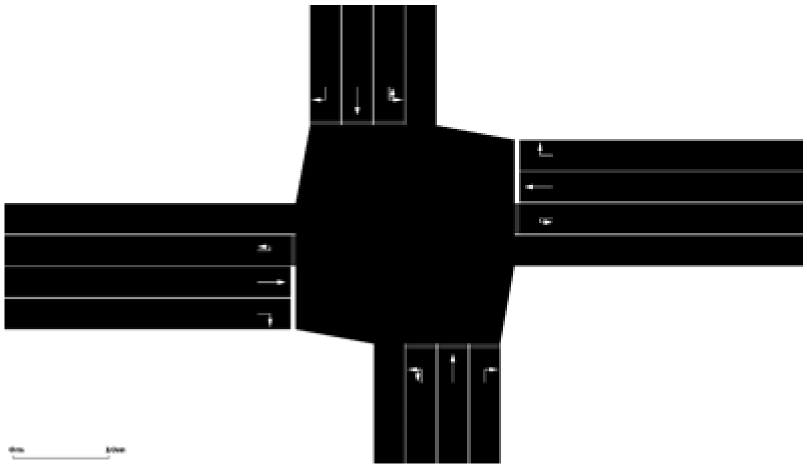
Setting Connection Priorities

Since version 0.9.6 you can also let vehicles passing a connection between two edges wait for another stream. Let's take another look at "Network with explicit edge-2-edge connections" above. Here, all right-moving vehicles may drive. The following definition within the connections file lets vehicles on vertical edges moving right wait for those which move straight on horizontal edges:

```
<connections>

      <!-- The next four connection definitions are same as in
      "Network with explicit edge-2-edge connections" -->

      <connection from="1si" to="3o"/>
      <connection from="1si" to="2o"/>

      <connection from="2si" to="4o"/>
      <connection from="2si" to="1o"/>

      <!-- now, let's prohibit the vertical connections by the
      horizontal -->

      <!-- prohibit moving right from top to left by straight from
      right to left -->
      <prohibition prohibitor="2si->1o" prohibited="4si->1o"/>
      <!-- prohibit moving straight from top to bottom by straight
      from right to left <prohibition prohibitor="2si->1o"
      prohibited="4si->3o"/>
      <!-- prohibit moving left from top to right by straight from
      right to left -->
      <prohibition prohibitor="2si->1o" prohibited="4si->2o"/>
      <!-- prohibit moving right from bottom to right by straight
      from left to right <prohibition prohibitor="1si->2o"
      prohibited="3si->2o"/>
      <!-- prohibit moving straight from bottom to top by straight
      from left to right <prohibition prohibitor="1si->2o"
      prohibited="3si->4o"/>
      <!-- prohibit moving left from bottom to right by straight from
      left to right <prohibition prohibitor="1si->2o"
      prohibited="3si->1o"/>

</connections>
```

As one may see, it was necessary to prohibit all connections from a vertical edge by the counterclockwise straight connection on a horizontal edge because otherwise the vehicles on the horizontal edge want to wait due to right-before-left - rule. The network looks like this:

Figure 4.7. Network with explicite prohibitions



The syntax of a prohibition-tag is: `<prohibition prohibitor="`*`<PROHIBITING_FROM_EDGE_ID>`*`->`*`<PROHIBITING_TO_EDGE_ID>`*`" prohibited="`*`<PROHIBITED_FROM_EDGE_ID>`*`->`*`<PROHIBITED_TO_EDGE_ID>`*`"/>`. This means we define two connections (edge-to-edge), the prohibiting one (`prohibitor`) and the prohibited (`prohibited`). Each connection is defined by a from-edge and a to edge, divided by "->".

Examples (compare to *<SUMO_DIST>*`/data/examples/netbuild/connections/cross3l_unconstrained/`) :

- *<SUMO_DIST>*`/data/examples/netbuild/connections/cross3l_prohibitions/` shows how prohibitions may be specified

Recent Changes:

- The possibility to add prohibitions was implemented for version 0.9.6

Building the Network

After you have generated the files you need being at least the edges and the nodes-files and optionally also a type and/or a connections file you should run NETCONVERT to build the network. The call should look like:

```
Netconvert-M --xml-node-files=MyNodes.nod.xml --xml-edge files = /
MyEdges.edg.xml --output-file=MySUMONet.net.xml
```

if you only use edges and nodes. Types and connections may be given as:

```
netconvert-M --xml-node-files=MyNodes.nod.xml --xml-edge files= /
MyEdges.edg.xml --xml-connection-files=MyConnections.con.xml --xml- /
type- files=MyTypes.typ.xml --output-file=MySUMONet.net.xml
```

Maybe your edge definitions are incomplete or buggy. If you still want to import your network, you can try passing "`--dismiss-loading-errors`" to NETCONVERT-M. In this case, edges which are not defined properly, are omitted, but NETCONVERT-M tries to build the network anyway. You may also flip the network around the horizontal axis. Use option "`--flip-y`" for this. You may also use abbreviations for the option names. These abbreviations and options used when building SUMO-networks from own XML-descriptions are:

```
( --xml-node-files | --xml-nodes | -n ) <NODES_FILE>
```

Uses the given file as the source of specification node positions and types. *Optional, type:filename, default: none*

`( --xml-edge-files | --xml-edges | -e )` *`<EDGES_FILE>`*

Uses the given file as the source of specification of roads connecting nodes. *Optional, type:filename, default: none*

`( --xml-connection-files | --xml-connections | -x )`
`<CONNECTIONS_FILE>`

Uses the given file as the source of specification how roads are connected (which lanes may be reached from which lanes). *Optional, type:filename, default: none*

`( --xml-type-files | --types | -t )` *`<TYPES_FILE>`*

Uses the given file as the source of edge types. *Optional, type:filename, default: none*

`--dismiss-loading-errors`

Continues with parsing although a corrupt edge occurred. This edge is not inserted and a warning is printed. *Optional (pregiven), type:bool, default: false*

`--flip-y`

Flips the y-position of nodes (and edges) along the y=zero-line. *Optional (pregiven), type:bool, default: false*

See also:

- "Setting default Values [#user_chp04-further_options-defaults]"
- "Using Edges' maximum Speed Definitions in km/h" [#user_chp04-further_options-kmh_speed]
- "Importing Networks without Traffic Light Logics [#user_chp04-further_optionsimporting_notls]"
- "Guessing On- and Off-Ramps [#user_chp04-further_options-guessing_ramps]"
- "Adding Turnarounds [#user_chp04-further_options-turnarounds]"
- Converting from Geocoordinates

Examples:

Almost all networks within the *`<SUMO_DIST>`*`/data/` - folder. Additionally some examples that cover the mentioned topics are:

- On using types:
- *`<SUMO_DIST>`*`/data/examples/netbuild/types/cross_notypes/`
- *`<SUMO_DIST>`*`/data/examples/netbuild/types/cross_usingtypes/`
- On using speed definition in km/h
- *`<SUMO_DIST>`*`/data/examples/netbuild/cross_notypes_kmh/`
- *`<SUMO_DIST>`*`/data/examples/netbuild/cross_usingtypes_kmh/`
- On using edge shapes
- *`<SUMO_DIST>`*`/data/examples/netbuild/shapes/hokkaido-japan/`

Recent changes:

- `--xml-type-files` was named `--type-file` in versions earlier than 0.9.2
- In the previous examples the option for nodes inclusion was misspelled (`--xml-nodes-files` is incorrect, `--xml-node-files` is right). Thanks to Leander Verhofstadt to recognize this.
- An error in this documentation has been removed for version 0.9.5

- 11.03.2008: False documentation updated: `--omit-corrupt-edges` is outdated; use `--dismiss-loading-errors` instead

Further NETCONVERT-M Options

NETCONVERT offers some more options to describe how the network shall be imported. The scope of some options does not cover all import types, though a list of valid import types for each option set is given.

Setting default Values

We have mentioned, that edge parameter may be omitted and defaults will be used in this case. You have the possibility to define these defaults using the following options:

`( --type | -T )` *<DEFAULT_TYPE_NAME>*

The name of the default type of edges. *Optional (pregiven), type:string, default: "Unknown"*

`( --lanenumber | -L )` *<DEFAULT_LANE_NUMBER>*

The number of lanes an edge has to use as default. *Optional (pregiven), type:int, default: 1*

`( --speed | -S )` *<DEFAULT_MAX_SPEED>*

The maximum speed allowed on an edge in m/s to use as default. *Optional (pregiven), type:float, default: 13.9*

`( --priority | -P )` *<DEFAULT_PRIORITY>*

The default priority of an edge. *Optional (pregiven), type: positive int, default: -1 (unset)* These options may be used while importing the following formats:

- XML-descriptions

Examples: none yet

Closing Thoughts (so far)

iNETGEN allows to create networks in a very comfortable way. For some small-sized tests of rerouting strategies, tls-signals etc., this is probably the best solution to get a network one can run some simulations at. The clear naming of the streets also eases defining own routes. Still, most examples within the data-section were written by hand for several reasons. At first, the examples are small enough and one may see the effects better than when using NETGEN. Furthermore, defining own networks using XML-data is more flexible. NETGEN is of course useless as soon as you want to simulate the reality.

Our current state-of-the-art approach for building networks is the following:

1. Get a plain (no tls, no link-2-link-connections, etc.) network from our NavTeq database
2. Import it using NETCONVERT-M and write the list of imported edges/nodes using the `–plainoutput` option
3. Build the network from the list of edges/nodes (normally setting the options `--guess-ramps` to true)
4. Load the network into GUISiMTraM, try to determine where tls are located and which connections between edges/lanes are false; A nice possibility to do this is to use Google Earth besides to investigate how the network looks in reality
5. Add `type="traffic_light"` attribute to those nodes in your plain file which were found

to be controlled by a tls
6. Add lane-to-lane connections in a previously generated connections-file
7. Build the network from the modified edges/nodes/connection files
8. Continue with step 4. until the network is as it shoud be

A good idea is to let some vehicles run through the network while investigating it. This will show possible bottleneck that arised from a false modelling of the network.

When using real life networks, we really advice guessing on- and off-ramps The on- off-ramps are guessed quite well, we can not state this for the tls, because we don't have made any comparisons with real life.

Recent Changes

Missing

Chapter 5. Route Generation

Introduction

After having your network converted into the SUMO-format, you could take a look at it using the gui-version of the simulation (see "Simulation-GUI"), but no cars would be driving around. You still need some kind of description about the vehicles. If you are importing data from other simulation packages, they normally bring own route definitions you can use. In case of using ArcView or own data or in other cases where you do not have the vehicle movements at all, you have to generate them by your own. From now on we will use the following nomenclature: A *trip* is a vehicle movement from one place to another defined by the starting edge (street), the destination edge, and the departure time. A *route* is an expanded trip, that means, that a route definition contains not only the first and the last edge, but all edges the vehicle will pass. There are several ways to generate routes for SiMTraM:

- using trip definitions

As described above, each trip consists at least of the starting and the ending edge and the departure time (see Chapter "Using Trip Definitions [#user_chp05-explicite-trips]").

- using flow definitions

This is mostly the same approach as using trip definitions, but you may join several vehicles having the same trips using this method (see Chapter "Using Flow Definitions [#user_chp05-expliciteflows]").

- using flow definitions and turning ratios

You may also leave out the destination edges for flows and use turning ratios at junctions instead (see Chapter "Using the Junction Turning Ratio - Router [#user_chp05-own_routes-jtr]").

- using OD-matrices

OD-matrices have to be converted to trips first (see Chapter "Using OD2TRIPS [#user_chp05-od2trips]"), then from trips to routes (see Chapter "Using Trip Definitions [#user_chp05-explicitetrips]").

- by hand

You can of course generate route files by hand (see Chapter "Building Routes 'by Hand' [#user_chp05-explicite-hand]").

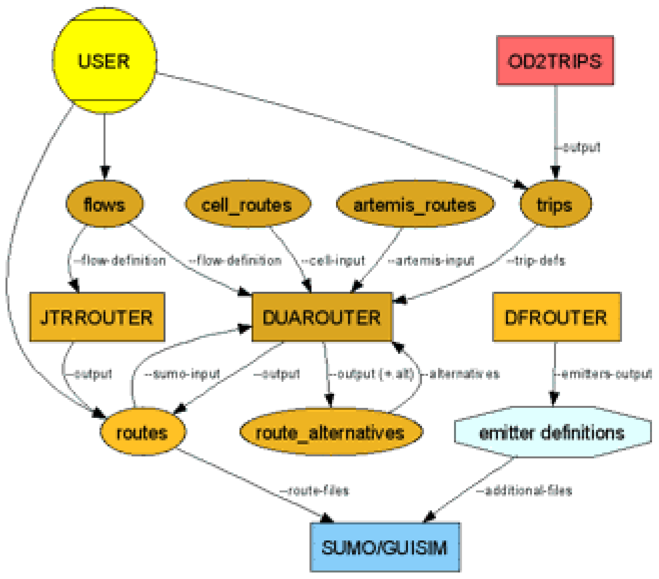
- using random routes

This is fast way to fill the simulation with life, but definitely a very inaccurate one (see Chapter " Generating random Routes [#user_chp05-own_routes-random]").

- by importing available routes (see Chapter "Importing Routes from other Simulations [#user_chp05-import_routes]")

By now, the SiMTraM-package contains four applications for processing routes. DUAROUTER-M is responsible for importing routes from other simulation packages and for computing routes using the shortest-path algorithm by Dijkstra. JTRROUTER-M may be used if you want to model traffic statistically, using flows and turning percentages at junctions. OD2TRIPS helps you to convert ODmatrices (origin/destination-matrices) into trips. A new application, the DFROUTER was added to the suite for version 0.9.5. Within the next chapters, at first the mandatory arguments are described, then we will show how each of the possible methods of generating routes from scratch can be used. In the following, importing routes and additional options are given followed by a small overview.

Figure 5.1. Building routes



Common, mandatory Values

Independent to what you are doing, you always have to supply the network using the `–netfile` (or `--net` or `-n` for short) option when working with either DFROUTER, DUAROUTER-M, JTRROUTER-M, or OD2TRIPS. Additionally, you should let the application know which time interval shall be used. Route/trip/flow definitions will be imported within the interval given by the options `--begin` (`-b`) and `– end` (`-e`). Definitions with departure time earlier than the one specified by `--begin` or later than those specified by `--end` will be discarded. If you do not give a value for the begin / end time step the defaults 0 and 86400 (one day) will be used, respectively.

Common options:
`( --net-file | --net | - n )` *`<SUMO_NET_FILE>`*
The network to route on. *Mandatory, type:filename, default: none*
`( --begin | -b )` *`<TIME>`* Defines the begin time routes shall be generated (in seconds).
Default (pregiven), type:int, default: 0
`( --end | -e )` *`<TIME>`* Defines the end time routes shall be generated (in seconds).
Default (pregiven), type:int, default: 86400

Building Routes from Scratch

You have either the possibility to generate completely random routes or to exactly describe what you want and pass this information to DUAROUTER-M or JTRROUTER-M, which then expand your descriptions to routes. As result, a routes file is normally generated which you may use within your simulation.

Caution

You have to know that each route should consist of at least three edges! On the first, the vehicle will be emitted. As soon as it reaches *the begin* of the last, it will be removed from the network. So to see the vehicle running, you should at least have one edge in between!

Generating own, explicit Routes

There are three possibilities to describe own routes. The most trivial one is to do this by hand. The first way to make more vehicle trips more automatically is the usage of trip definitions, the second one the usage of flow descriptions. Trip definitions describe the movement of a single vehicle giving the departure time, and both the origin and the destination edges via their id. Flow descriptions use these values too, but instead of describing only one vehicle, the description is used for a defined number of vehicles to be emitted within a described interval. Due to this, instead of the departure time, the period's begin and end times must be supplied and the number of vehicles to emit within this interval. We will describe both data types less briefly, now.

Building Routes 'by Hand'

The most simple way to get own routes is to edit a routes file by hand, but only if the number of different routes is not too high. Most of the routes within the examples were written by hand, in fact. Before starting, you must know that a vehicle in SiMTraM 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. In this case, routes need a further information. Assume you want to build a routes file "`routes.rou.xml`". Herein, you can define a vehicle type as following:

```
<routes>
<vtype id="type1" accel="0.8" decel="4.5" sigma="0.5"
stripWidth= "5" length="5" maxspeed="70"/>
</routes>
```

The values used above are the ones most of the examples use. They resemble a standard vehicle as used within the Stefan Krauß' thesis besides that the minimum gap between vehicles is not added to the length. These values have the following meanings:

- `id`: A string holding the id of the vehicle type
- `accel`: The acceleration ability of vehicles of this type (in m/s^2)
- `decel`: The deceleration ability of vehicles of this type (in m/s^2)
- `sigma`: The driver imperfection (between 0 and 1)
- `length`: The vehicle length (in m)
- `maxspeed`: The vehicle's maximum velocity (in m/s)
- `stripWidth`: Number of strips a vehicle can have (vehicle width = number of strip * width of a strip)
- `color`: An optional color of the vehicle type, encoded as three values between 0 and 1 for red, green, and blue, divided by a ','. Please remark that no spaces between the numbers are allowed.

Having this defined, you can build vehicles of type "type1". Let's do this for a vehicle with an completely own route:

```
<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>
```

Ok, now we have a red (color=1,0,0) vehicle of type "type1" named "0" which will start 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. Ok, let's review a vehicle's attributes:

- `id`: A string holding the id of the vehicle
- `type`: The vehicle type to use for this vehicle
- `depart`: The time at which the vehicle shall be emitted into the net
- `color`: An optional color of the vehicle, encoded as three values between 0 and 1 for red, green, and blue, divided by a ','. Please remark that no spaces between the numbers are allowed.

This vehicle has an own, internal route which is not shared with other vehicles. You may also define two vehicles using the same route. In this case you have to "externalize" the route by giving it an id. From SUMO 0.9.7 on it is no longer neccessary to tell SUMO that the route is shared by using the `multi_ref` attribute, all routes defined outside of vehicles are shared. 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>
```

You may have noticed that the route itself also got a color definition, so the attributes of a route are:

- `id`: A string holding the id of the route
- `edges`: A space spearated list of edge ids forming the route (the old style of defining the edges inside route brackets is considered deprecated)
- `color`: An optional color of the vehicle, encoded as three values between 0 and 1 for red, green, and blue, divided by a ','. Please remark that no spaces between the numbers are allowed.

This knowledge should enable you to specify own route definitions by hand or using selfwritten scripts. All routing modules are generating route files that match this routes and vehicles specification.There are a few important things to consider when building your own routes:

- *Routes have to be connected.* At the moment the simulation does not raise an error if the next edge of the current route is not a successor of the current edge. The car will simply stop at the end of the current edge and will possibly be "teleported" to the next edge after a waiting time. This is very likely to change in future versions.
- *Routes have to contain at least two edges.* The simulation stops the car at the start of the last edge, thus a route consisting of a single edge is empty. This is likely to change in future versions of SUMO.
- *The starting edge has to be at least as long as the car starting on it.* At the moment cars can only start at a position which makes them fit on the road completely.
- *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 route with departure time <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 behaviour may be disabled by specifying `--route-steps 0`. The first three conditions can be checked using *<SUMO_DIST>*`/tools/routecheck.py`.
-

Route and vehicle type distributions

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

```
<routes>
<vtypeDistribution id="typedist1">
<vtype id="type1" accel="0.8" decel="4.5" stripWidth = "3"
sigma="0.5" length="5" maxspeed="<vtype id="type2" accel="1.8"
decel="4.5" sigma="0.5" length="15"
maxspeed="</vtypeDistribution>
</routes>
<routes>
<routeDistribution id="routedist1">
<route id="route0" color="1,1,0" edges="beg middle end rend"
probability="<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. At the moment the id for the childs is mandatory, this is likely to change in future versions. Now you can use distribution just as you would use individual types and routes:

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

Using Trip Definitions

Trip definitions that can be laid into the network may be supplied to the router using an XML-file. The syntax of a single trip definition is:

`<tripdef id="`*`<ID>`*`" depart="`*`<TIME>`*`" from="`*`<ORIGIN_EDGE_ID>`*`" to="`*`<DESTINATION_EDGE_ID>`*`" [type="`*`<VEHICLE_TYPE>`*`"] [period="`*`<INT>`*`" repno="`*`<INT>`*`"] [color="`*`<COLOR>`*`"]/>`. You have to supply the edge the trip starts at (origin), the edge the trip ends at (destination) and the departure time at least. If the type is not given, a default ("SUMO_DEFAULT_TYPE") will be used and stored within the routes-file. If the attribute `period` is given, not only one vehicle will use the route, but every n seconds (where n is the number defined in period), a vehicle using this route will be emitted. The number of vehicles to emit using this route may be additionally constrained using `repno`.

Let's review a trip's parameter:

- `id`: A string holding the id of the route (and vehicle)
- `depart`: The time the route starts at
- `from`: The name of the edge the route starts at; the edge must be a part of the used network
- `to`: The name of an the edge the route ends at; the edge must be a part of the used network
- `type`: The name of the type the vehicle has (optional)
- `period`: The time after which another vehicle with the same route shall be emitted (optional)
- `repno`: The number of vehicles to emit which share the same route (optional)
- `color`: Defines the color of the vehicle and the route (optional)

This file is supplied to DUAROUTER using the option "`--trip-defs`" or "`-t`": `duarouter --trip-defs=<TRIP_DEFS> --net=<SUMO_NET> --output-file=MySUMORoutes.rou.xml -b <UINT> -e <UINT>`

Specific options:
`( --trip-defs | --trips | -t ) <TRIP_DEFINITION_FILE>`
Tells DUAROUTER from what file trip definitions shall be read. *Optional, type:filename, default: none*

Using Flow Definitions

Flow amounts share most of the parameter with trip definitions. The syntax is: `<flow id="<ID>" from="<ORIGIN_EDGE_ID>" to="<DESTINATION_EDGE_ID>" begin="<INTERVAL_BEGIN>" end="<INTERVAL_END>" no="<VEHICLES_TO_EMIT>" [type="<VEHICLE_TYPE>"] [color="<COLOR>"]/>`. Notice the following differences: the vehicle does not take a certain departure time as not only one vehicle is described by this parameter, but a set of, given within the attribute "`no`" (short for number). The departure times are spread uniformly within the time interval described by `<INTERVAL_BEGIN>` and `<INTERVAL_END>`. All these three attributes must be integer values. The values "`period`" and "`repno`" are not used herein. Flow definitions can also be embedded into an interval tag. In this case one can (but does not have to) leave the tags "`begin`" and "`end`" out. So the following two snipples mean the same: `<flow id="0" from="edge0" to="edge1" begin="0" end="3600" no="100"/> no="100"/> </interval>`

Let's review flow parameter:

- `id`: A string holding the id of the flow; vehicles and routes will be named "`<id>_<RUNNING>`" where `<RUNNING>` is a number starting at 0 and increased for each vehicle.
- `from`: The name of the edge the routes start at; the edge must be a part of the used network
- `to`: The name of an the edge the routes end at; the edge must be a part of the used network
- `type`: The name of the type the vehicle has
- `begin`: The begin time for the described interval
- `end`: The end time for the interval; must be greater than `<begin>`; vehicles will be emitted between `<begin>` and `<end>`-1
- `no`: The number of vehicles that shall be emitted during this interval
- `color`: Defines the color of the vehicles and their routes (optional)

As we have to read in the flow definitions completely into the memory - something we do not have to do necessarily with trips, an extra parameter (`-f` or `--flows`) is used to make them known by the router:

```
duarouter --flows=<FLOW_DEFS> --net=<SUMO_NET> \
--output-file=MySUMORoutes.rou.xml -b <UINT> -e <UINT>
```

Remind that you can not insert flow descriptions into a trip definitions file. The opposite (some trip definitions within a flow descriptions file) is possible. You also can give both files at the input file, for

example:

```
duarouter --flows=<FLOW_DEFS> --trip-defs=<TRIP_DEFS> --  \
net=<SUMO_NET> --output-file=MySUMORoutes.rou.xml -b <UINT> -e <UINT>
```

Specific options:
`( --flow-definition | --flows | -f ) <FLOW_DEFINITION_FILE>`
Tells DUAROUTER/JTRROUTER from what file flow definitions shall be read. *Optional, type:filename, default: none*

Generating random Routes

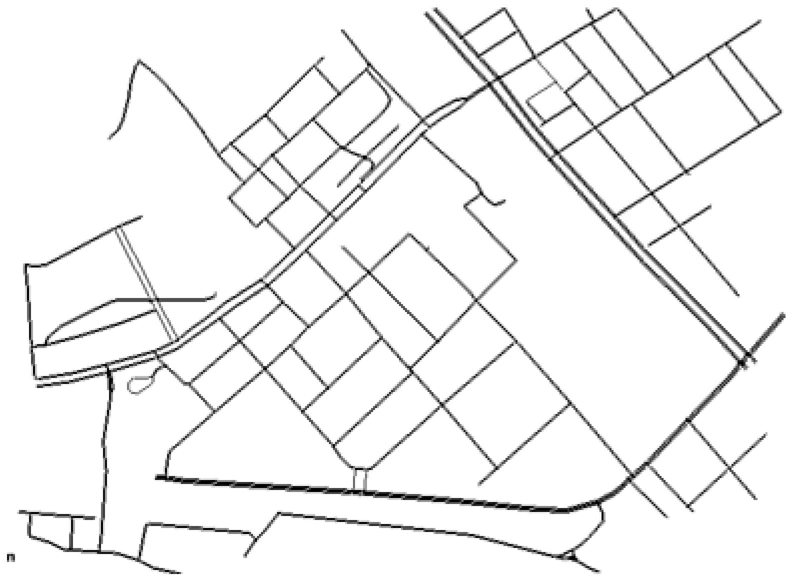
Random routes are the easiest, but also the most inaccurate way to feed your network with vehicle movements. Using the following call to DUAROUTER:

```
duarouter --net=<SUMO_NET> -R <FLOAT> --output- \
file=MySUMORoutes.rou.xml  -b <UINT> -e <UINT>
```

You will generate random routes for the time interval given by `-b(egin)` and `-e(nd)`. In each time step as many vehicles will be emitted into the network as given by the value of `-R` (`--random-persecond`). You can also supply values smaller than one. In this case, a single vehicle will be emitted each 1/*<-R>* step. Example: `-R 0.25` generates a route description, which, when loaded, forces the simulation to emit a single vehicle each fourth time step. It is also possible to use this parameter in combination with other route definitions, for example supplying some fix routes and additionally generate random routes.

Random routes are not the best way to generate routes. Take a look at the network displayed below. This network has two rural and many minor roads. Random routes are by now spread all over the network and each road is chosen to be the starting or the ending without respecting his function. Due to this, the network is filled over with cars, coming from and approaching directions, the normal traffic is not taking - the normal traffic would concentrate on rural roads.

Figure 5.2. A network where the usage of random routes causes an improper behaviour due to the mixture of rural and minor roads



Options: `( --random-per-second | -R )` *<RANDOM_VEHICLES_PER_SECOND>*
Forces DUAROUTER/JTRROUTER to generate random trips. Per second the given number of vehicles will be generated. *Optional, type:float, default: none*

Closing Thoughts (so far)

Recent Changes

Missing

Chapter 6. Performing the Simulation

Having the network description and the routes you have everything to perform a simulation. The fastest way to get results - their different types will be described within the following sub-chapters - is to use the SiMTraM - command line simulation. This command line tool does not generate any graphical output as the SiMTraM-GUI does, but is much faster in execution. To start a simulation, you have to supply the following information:

- The file that contains the network
  Use the `--net-file` (or `--net` or `-n`) *`<FILE>`* option to pass the simulation the name of the network to use. The network must be one build using NETCONVERT-M or NETGEN-M.

- The routes to use
  Use the `--route-files` (or `--routes` or `-r`) *`<FILE>`*`[,`*`<FILE>`*`]*` option to specify which files shall be used to read routes from. In this case, the name is not ambigous – multiple files can be used.

- The simulation time the simulation begins at
  This is the first time step the simulation has to perform. Be aware, that this time should fit to the time your routes start. Pass it to SUMO using `--begin` (or `-b`) *`<INT>`* where *`<INT>`* is the time step in seconds.

- The simulation time the simulation ends at
  This is the last step of the simulation. When this time step is reached, the simulation will end. Pass it to SiMTraM using `--end` (or `-e`) *`<INT>`* where *`<INT>`* is the time step in seconds.

All these values must be given in order to perform a simulation. Still, no output is generated. Generating output is described in the next chapter. Besides this, there are also some other additional structures which may be applied to the simulation scenario and of course there are some more questions to answer about inserting vehicles into the net.

Output Generation

Due to its scientific purpose, SUMO tasks lie beyond simple visualisation of traffic. The results of a simulation must be available and one must be able to process them. In the next subchapters, possibilities to generate output are described.

Detectors

One possibility to generate output is to use so called "detectors". You will find detectors one knows from the real world such as induction loops but also some virtual ones. Basically, the main distinction between detector types SUMO offers is their dimension. The next list shows all available detector types. Their type names "E*" have their origin in the German word "Erfassungsbereich" meaning "detection area".

- E1: Induction loops
  Induction loops have a position only and no areal dimensions. They are meant to be a slice plane through a single lane and measure only the vehicles passing them.

To supply the definitions of these structures to the simulation, we use an additional file and pass it to SiMTraM or GUISiMTraM using the `--additional-files` (`-a`) - option. Each of these files may contain all the definitions about additional structures such as detectors, emitters, etc., in random order.

Caution

Please note that all output have not yet been verified for sub second simulation.

E1-Detectors (Induction Loops)

An induction loop is defined this way within an additional file: `<e1-detector id="<ID>" lane="<LANE_ID>" pos="<POSITION_ON_LANE>" freq="<AGGREGATION_TIME>" file="<OUTPUT_FILE>" [friendly_pos="x"]/>`

The "`id`" is any string by which you can name the detector. The attributes "`lane`" and "`pos`" describe on which lane and at which position on this lane the detector shall lay. The "`freq`"-attribute describes the period over which collected values shalle be aggregated. The "`file`" attribute tells the simulation to which file the detector shall write his results into. The file will be generated, does not have to exist earlier and will be overwritten if existing without any warning. The folder the output file shall be generated in must exist.

The attributes:

- `id`: A string holding the id of the detector
- `lane`: The id of the lane the detector shall be laid on. The lane must be a part of the network used.
- `pos`: The position on the lane the detector shall be laid on in meters. The position must be a value between -1*lane's length and the lane's length. In the case of a negative value, the position will be computed backward from the lane's end (the position the vehicles drive towards).
- `freq`: The aggregation period the values the detector collects shall be summed up.
- `file`: The path to the output file. The path may be relative.
- `friendly_pos`: If set, no error will be reported if the detector is placed behind the lane. Instead, the detector will be placed 0.1 meters from the lane's end or at position 0.1, if the position was negative and larger than the lane's length after multiplication with -1.

A single data line within the output of a simulated e1-detector looks as following (the line is not broken within the output):

```
<interval begin="<BEGIN_TIME>" end="<END_TIME>" id="<DETECTOR_ID>" \
stripnum="<STRIP_ID>" nVehContrib="<MEASURED_VEHICLES>"flow="<FLOW>"\
occupancy="<OCCUPANCY>" speed="<MEAN_SPEED>" length="<MEAN_LENGTH>" \
nVehEntered="<ENTERED_VEHICLES>" totalnum="<ENTERED_VEHICLES in  \
lane>" />
```

The values are described in the following table.

Table 6.1. Definition of values generated by e1-detectors

Name	Measure	Description
`begin`	`(simulation) seconds`	The first time step the values were collected in
`end`	`(simulation) seconds`	The last time step the values were collected in (may be equal to begin)
`id`	`-`	The id of the detector (needed if several detectors share an output file)
`stripnum`	`-`	The id of the strip
`nVehContrib`	`#vehicles`	The number of vehicles that have completely passed the detector within the interval

flow	vehicles/hour	The number of contributing vehicles extrpolated to an hour
occupancy	%	The percentage (0-100%) of the time a vehicle was at the detector.
speed	m/s	The mean velocity of all completely collected vehicles.
length	M	The mean length of all completely collected vehicles.
nVehEntered	#vehicles	All vehicles that have touched the detector. Includes vehicles which have not passed the vehicle completely (and which do not contribute to collected values).
Total_veh	#vehicles	All vehicles that have touched the detector in a lane.

Network State Dump

In the hope that every user wants to know different things and is able to write a tool that parses this information from a not aggregated output, the network dump was the first output capability we've implemented. To force SUMO to build a file that contains the network dump, extend your command line (or configuration) parameter by `--netstate-dump` (or `--ndump` or `-- netstate`) *<FILE>*. *<FILE>* is hereby the name of the file the output will be written to. Any other file with this name will be overwritten, the destination folder must exist. The network dump is a xml-file containing for each time step every edge of the network with every lane of this edge with all vehicles on this lane. For each vehicle, his name, speed and position on his lane are written. A network dump-file looks like this:

```
<sumo-netstate>
      <timestep time="<TIME_STEP>">
            <edge id="<EDGE_ID>">
                  <lane id="<LANE_ID>">
                      <strip id= ="<STRIP_ID>">
                        <vehicle id="<VEHICLE_ID>"
                        pos="<VEH_POSITION>" speed="<VEH_SPEED>"/>
                     ... more vehicles if any on this lane ...
                     </strip>
                  ... More strips if lane possesses more...
                  </lane>
            ... more lanes if the edge possesses more ...
            </edge>
      ... more edges ....
      </timestep>
... the next timestep ...
</sumo-netstate>
```

The values have the following meaning:

- `time`: The time step described by the values within this `timestep`-element
- `id`: The id of the edge/lane/vehicle
- `pos`: The position of the vehicle at the lane within the described time step
- `speed`: The speed of the vehicle within the described time step

As you may imagine, this output is very verbose. His main disadvantage is the size of the generated file. It's very easy to generate files that are several GB large within some minutes. It is of course possible to write some nice tools that parse the file (using a SAX-parser) and generate some meaningful information, but we do not know anyone who has made this. Another problem is that the simulation's execution speed of course breaks down when such an amount of data must be written. Normally, all lanes are written, even if there is no vehicle on them. You can change this behaviour using the boolean switch `--dump-empty-edges`. In this case, only those edges and lanes will be written that contain vehicles.

Traffic Management and Other Structures

SiMTraM holds several additional structures to model speed limits, public transport etc. The structures are normally defined within additional files.

Traffic Lights

Normally, NETCONVERT-M will generate traffic lights and programs for junctions during the computation of the networks. Still, these computed programs differ quite often from those found in reality. To feed the simulation with traffic light programs from the reality, it is possible to load additional programs since version 0.9.4. Furthermore, one can describe when and how a set of traffic lights can switch from one program to another. Both will be discussed in the following subchapters. Handling of traffic lights is not yet very user friendly.

Adding new TLS-Programs

Since version 0.9.4 you may attach a new program to a tls after the network has been loaded. Defining a tls program is not that straightforward, yet. If you are definitely interested in this, we advise you to read the "SUMO - More on... Traffic Lights [http://sumo.sourceforge.net/docs/gen/sumo_moreon_tls.shtml]" document where the format is described. Basically, a tls program definition looks like this:

```
<tl-logic type="static">
      <key>0</key>
      <subkey>0</subkey>
      <phaseno>8</phaseno>
      <offset>0</offset>

      <phase duration="20" phase="0000111100001111"   \
      brake="1111110011111100" yellow="0000000000000000"/>
      <phase duration="4" phase="0000110000001100"    \
      brake="1111111111111111" yellow="0000001100000011"/>
      <phase duration="3" phase="0000110000001100"    \
      brake="1111001111110011" yellow="0000000000000000"/>
      <phase duration="4" phase="0000000000000000"    \
      brake="1111111111111111" yellow="0000110000001100"/>
      <phase duration="20" phase="1111000011110000"   \
      brake="1100111111001111" yellow="0000000000000000"/>
      <phase duration="4" phase="1100000011000000"    \
      brake="1111111111111111" yellow="0011000000110000"/>
      <phase duration="3" phase="1100000011000000"    \
      brake="0011111100111111" yellow="0000000000000000"/>
      <phase duration="4" phase="0000000000000000"    \
      brake="1111111111111111" yellow="1100000011000000"/>
</tl-logic>
```

After you have defined a tls program, you can add it to one of your additional files. You may load several programs for a single tls into the simulation. The program loaded as last will be used (unless not defined using a WAUT description, see below). Please remark, that all subkeys of your programs must differ if they describe the same tls.

Caution

Please keep in mind that this feature is quite new and that du to this some things may not work as suspected and may get changed in the near future.

Defining the switch Times and Procedure

In the reality, a tls often uses different programs during a day and maybe also for weekdays and for the weekend days. Since version 0.9.4 you can define switch times between the programs using a WAUT (I am very sorry, but I do not know the English word for WAUT - this may be a matter of change). Let's assume we would have a tls which knows four programs - two for weekdays and two for weekend days where from 22.00 till 6.00 the night plan shall be used and from 6.00 till 22.00 the day plan. We'll give these programs the names "weekday_night", "weekday_day", "weekend_night", "weekend_day". To describe the switch process, we have to describe the switch at first, assuming our simulation runs from monday 0.00 (second 0) to monday 0.00 (second 604800):

```
<WAUT refTime="0" id="myWAUT" startProg="weekday_night">
 <wautSwitch time="21600" to="weekday_day"/> <!-- monday, 6.00 -->
 <wautSwitch time="79200" to="weekday_night"/> <!-- monday, 22.00 -->
 <wautSwitch time="108000" to="weekday_day"/> <!-- tuesday, 6.00 -->
 ... further weekdays ...
 <wautSwitch time="453600" to="weekend_day"/> <!-- saturday, 6.00 -->
 ... the weekend days ...
</WAUT>
```

The fields in `WAUT` have the following meanings:

- `refTime`: A reference time which is used as offset to the switch times given later (in simulation seconds)
- `id`: The name of the defined WAUT
- `startProg`: The program that will be used at the simulation's begin and the fields in `wautSwitch`:
- `time`: The time the switch will take place
- `to`: The name of the program the assigned tls shall switch to

Of course, programs with the used names must be defined before this definition is read. Also, the time must be sorted. Additionally, we have to define which tls shall be switched by the WAUT. This is done as following:

`<wautJunction wautID="myWAUT" junctionID="RCAS" [procedure="Stretch"] [synchron="`Here, the attributes have the following meaning:

- `wautID`: The id of the WAUT the tls shall be switched by
- `junctionID`: The name of the tls to assign to the WAUT
- `procedure`: The switching algorithm to use; If none is given, the programs will switch immediately (default)
- `synchron`: Additional information whether the switch shall be done synchron (default: false)

You may assign several tls to a single WAUT. YOu may also assign several WAUTs to a single junction in theory, but this is not done in reality. The switching procedures are currently under development.

Vehicle Classes

Since version 0.9.5 SUMO is capable to handle vehicle classes. One can close a road or a lane for certain vehicle classes or explicitely allow certain vehicle classes on a road/lane. This is done by a combination of assigning allowed/disallowed vehicle classes to roads/lanes and additionally giving vehicles a further class attributes. Available vehicle classes as well as using them is described within the next subchapters.

Available Vehicle Classes

A vehicle class is made up of two parts. The first part describes to what kind of an authority the vehicle belongs. The next table shows what kind of authorities are defined currently:

Table 6.2. Allowed vehicle class authority descriptions

Table Name	Description
private	The vehicle belongs to a private person
public_transport	The vehicle is a public transport vehicle
public_emergency	The vehicle is an emergency vehicle
public_authority	The vehicle belongs to a public authority (police)
public_army	The vehicle is an army vehicle
vip	The vehicle is used to transport a vip (very important person)

The second part describes the kind of the vehicle. Currently possible values are shown within the next table:

Table 6.3. Allowed vehicle class vehicle kind descriptions

Table name	Description
passenger	A plain passenger car
hov	A heavy occupied vehicle
taxi	A taxi
bus	A bus
delivery	delivery A small delivery vehicle
transport	A truck
lightrail	A lightrail
cityrail	A cityrail
rail_slow	A slow transport rail
rail_fast	A fast passenger rail
motorcycle	A motorcycle
bicycle	A bicycle
pedestrian	A pedestrian

Please remark that both the authority descriptions and kind descriptions are only names, no model is stored behind them. By defining a vehicle type as "pedestrian" you will not get a person walking within the simulation - currently pedestrian are not modeled anyway. These values simply name possible types of vehicles found on a network to allow closing/opening lanes or edges for them currently.

Closing/Opening Roads/Lanes for certain Vehicle Classes

Roads/lanes are normally marked to allow/disallow a certain vehicle class while building the network using NETCONVERT-M. This process is described in chapter "Defining allowed Vehicle Types [http://sumo.sourceforge.net/docs/gen/user_chp04.shtml#user_chp04-xml_descriptions-edges-vclasses]".

Assigning a Type to a Vehicle

You can assign a vehicle class to a vehicle by extending this vehicle's vehicle type. Assume you want to set a vehicle as being of the class "bus". A vehicle type definition could look like this:

```
<vtype id="BUS" accel="2.6" decel="4.5" sigma="0.5" length="15"
maxspeed="70" color="1,1,0" vclass="public_bus"/>
```

In this case, the vehicle will drive only on lanes/roads where all vehicle classes are allowed or where public busses are not disallowed or where public busses are explicitely allowed.

Using the Files in a correct Way

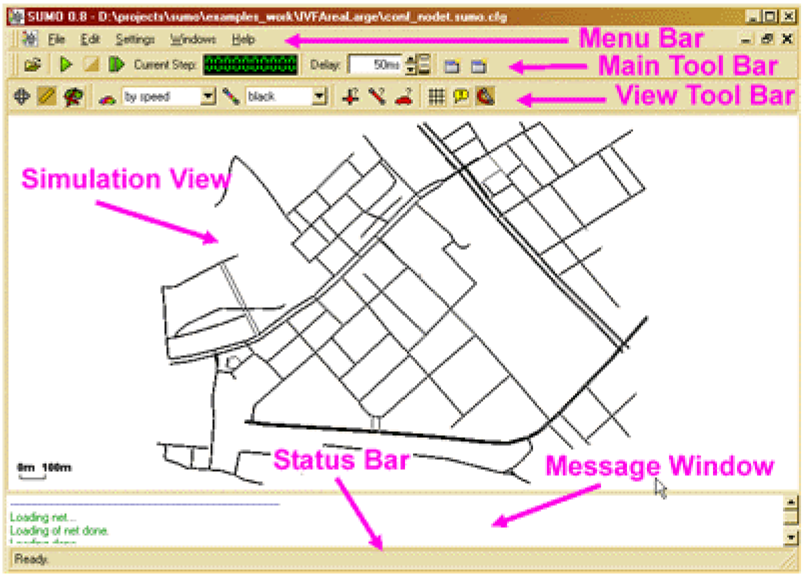
You may have noticed that beside the networks, SUMO additionally reads route files and "additional" files. Most of the structures (detectors, actors, route definitions, vehicle type definitions, tls definitions, etc.) may be placed in both route files and additional files. On the low application level the difference between the two file types is the order of loading them. Normally, when the option `route-steps` is left to be not equal to zero, additional files are parsed first, in the order of their definition. This means if you set the option "`- a file1.add.xml;file2.add.xml`", at first "`file1.add.xml`" will be loaded, then "`file2.ad..xml`". Each file is read completely before the next file is parsed. This means that if you have some global routes and want to reference them by a changing set of vehicles, you should place these routes in a file which is loaded at first. After all additional files have been read, the route files are opened. Still, they are not read immediately but as soon as the simulation starts. Each of these files is read until a vehicle emission occures which is beyond the current time step + time defined in `route-steps`. Here, all route files are parsed in the order they occured within the call, too. The things change a little bit if the option `route-steps` is set to zero. In this case, the route files are parsed as first, BEFORE the simulation starts. They also will be parsed completely before the additional files are parsed. If you need your additional files to be parsed at first, either use a `routesteps` value not equal to zero or place your additional files at the begin of the route-files list.

Other Topics

Chapter 7. Simulation-GUI

The simulation-GUI (graphical user interface) is basically a wrapper around the command line simulation. The normal procedure is to start the gui-version like any other Window-based application (double-click on it) and to load a simulation's description specified using a "normal" configuration-file as used by the simulation's command line version. After loading it - what may dure a longer time if the network is large or the simulation is forced to load many routes at once - the network shall appear. Your application should then look like displayed below (with your own network, of course).

Figure 7.1. The GUI-Window with a loaded simulation (violet: names of the controls as used below)



```
Loading net...
Loading of net done.
```

You can now start your simulation using the "play"-button and/or manoeuvre within the network pressing one of the mouse buttons and moving the mouse. When moving the mouse within the window with the left button pressed, you'll move the network to the direction you move the mouse. When the mouse is moved with the right button pressed, you change the scale the network is displayed in, zooming into and out of the network. We will now discuss the different possibilities to use the graphical user interface more deeply.

Main Window Interface

Menu Bar

File-Menu

- Open Simulation...

  Opens a file dialog that lets you choose a SiMTraM-configuration file that describes a complete simulation. The simulation described within this file will be loaded. Remark that you have to describe the simulation in full - no further extension is possible. You can of course load a simulation if another one is already loaded. In this case, the previous simulation will be closed.
- Reload Simulation

  Reloads the previously opened simulation.
- Close

  Closes the loaded simulation.
- [RECENT FILES]

if you have opened at least one file before, it will be displayed within this list. The list may contain up to ten files read previously.

- Clear Recent Files
  Clears the list of recent files.
- Quit
  Quits the application.

Edit-Menu

- Edit Chosen...
  Opens a dialog that lets you load/save and edit the list of chosen items.
- Edit Additional Weights...
  This menu enables you to edit additional weights for edges. These additional weight descriptions may be saved into a file and read by the DUAROUTER-M and his variants.
- Edit Breakpoints...
  This menu enables you to edit, load and save breakpoints. By now, the simulation will stop at one of the given brekpoints (simulation time steps) and can be then continued by pressing the "play"-button ( ).

Settings-Menu

- Application Settings...
  By now, one can only set whether the application shall be closed automatically when the loaded simulation ends.
- Simulation Settings...
  Displays the settings as read from the configuration file. This item is only accessible if a simulation has been loaded.

Windows-Menu

- Show Status Line
  By pressing this menu item, you can switch the status line off and on.
- Show Message Window
  By pressing this menu item, you can switch the message window off and on.
- Show Tool Bar
  By pressing this menu item, you can switch the toolbar off and on.
- Tile Horizontally
  Reorders the position of windows.
- Tile Vertically
  Reorders the position of windows.
- Cascade
  Reorders the position of windows.
- Close
  Closes the uppermost window.
- Clear Message Window
  Deletes all contents from the message window.

Help-Menu

- About
  Shows a small window with some information about SiMTraM.

Tool Bar

File Operations

- Open Button
  Opens a file dialog that lets you choose a SiMTraM-configuration file that describes a complete simulation. The simulation described within this file will be loaded. Remark that you have to describe the simulation in full - no further extension is possible. You can of course load a simulation if another one is already loaded. In this case, the previous simulation will be closed.
- Reload Button
  Reloads the previously opened simulation.

Simulation Operations

- Play Button
  Starts the simulation. If a loaded simulation was not started before, it will begin with the step described by the b(egin)-parameter within the loaded configuration file. If the simulation was started and stopped, it will continue.

Caution

It is not possible to restart a simulation, you have to reload it.

- Stop Button
  Stops a running application. A stopped application can be continued using the play-button (see above).
- Single Step Button
  Performs a single simulation step.
- Current Step Field
  After the loaded simulation has been started, the information about the current time step is displayed herein.
- Simulation Speed Control

The value you can change using this control is the time the application waits between two simulation steps. The higher the value, the slower the simulation will run.

Window Operations

- New Microscopic View - Button
  Opens a new window which displays the streets and vehicles moving on them.
- New Lane-Aggregated View - Button
  Opens a new window which displays the streets and vehicles moving on them.

Simulation Window Interfaces

SUMO-GUI provides different views on the simulation. The microscopic view shows the vehicles running just the way as the simulation performs his work. Aggregated views show the situation on the streets by coloring lanes by an aggregated value. Vehicles are not shown within the aggregated view.

Common Controls

Tracking Settings

- Locate Junction - Button
  Opens a window that allows to choose a junction name from the set of junctions the network consists of. Pressing ok with a chosen junction zooms the view to this junction.
- Locate Edge - Button
  Opens a window that allows to choose an edge name from the set of junctions the network consists of. Pressing ok with a chosen edge zooms the view to this edge.

View Settings

- Recenter View - Button
  You can use this button to reset the view to show the whole network. After pressing this button, the view will be the same as after loading the simulation: The zoom factor will be reset to a value that lets the window display the whole simulation area and the middle of the loaded network will be place into the middle of the view.

Interacting with Objects

Display an Object's Name

Each view has the possibility to display tool tips. If enabled using the "Show Tool Tips"-Button ( ) the name of an object will pop up in a yellow windows if the cursor is over the object. A second click on the "Show Tool Tips"-Button disables this feature.

Caution

This feature does slow down the visualisation. Use should use this carefully and disable if not needed.

Object Popup Menus

If the cursor is over an object you can press down the right mouse button and after ahlf a second a popup menu will be shown that allows you some further interaction with the object. Normally, the following functions are available:

- Center

Changes the view in a manner that the current object lies within the the view's center. Further, some objects allow an interaction, that means to change some of the object's parameter. You can access this using the command:

- Manipulate

Object Selection

From version 0.8. you are able to add every object that has a name (as shown if turning Tool Tips on) into a list of selected objects. You can select an object by holding the Alt-key and pressing the left mouse button when the mouse is over the object. Doing the same a second time will deselect the object again. You may wonder whether an object is selected or not. Use the lane colouring "by selection" from "Change Lane Colouring Scheme". When this colouring scheme is used, selected lanes are shown blueish, the other black. The menu entry Edit-Edit Chosen... allows you to edit the list of selected objects by deselected ones you don't need. It also allows you to save the list of selected objects. The resulting file contains the names of the selected objects predeccesed by the object's type, one per line

Caution

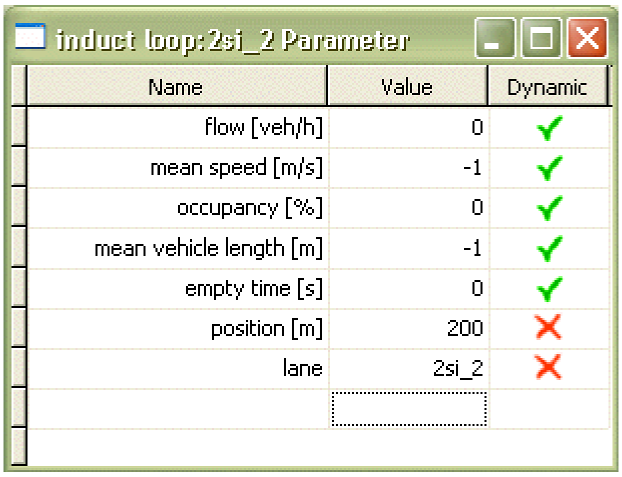
Load is not implemented, yet.

Parameter Windows

If you choose the option "Show Parameter" from an object's popup menu, a window like the one displayed below will appear:

Figure 7.2. A sample Parameter Window (for an induction loop in this case)

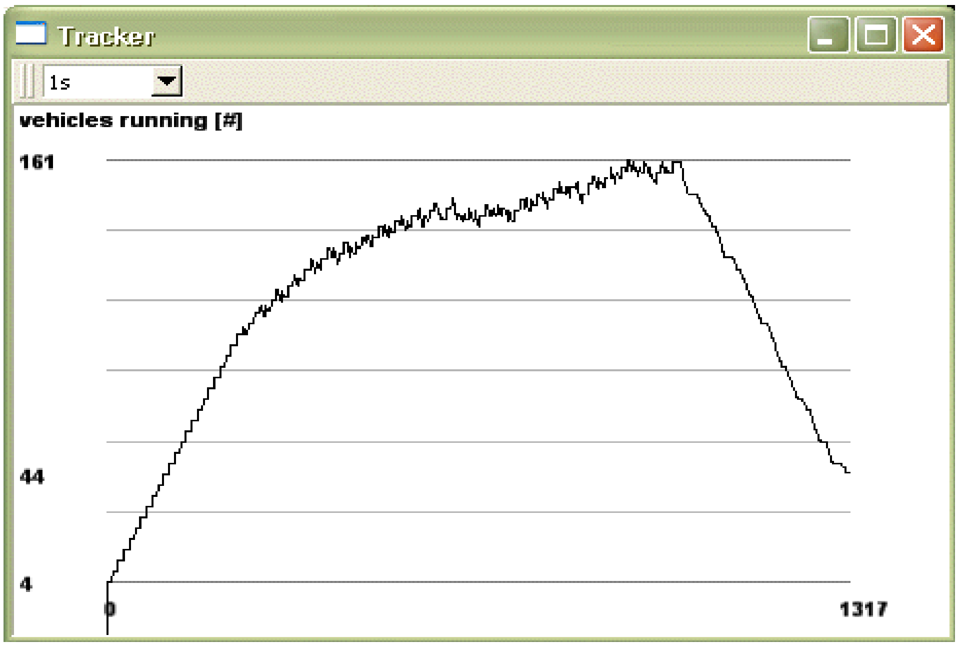
This window conatins some of each object's parameter, including the parameter's name, its current value and the information is static (marked with a) or dynamic (marked with a) within a simulation run. Pressing the right mouse button when being over a line marked as dynamic will show a small popup window with only a single command: "Open in new Tracker". Choosing this option will allow you to open another window where this parameter's values will be shown as a time line over the simulation run.



induct loop:2si_2 Parameter

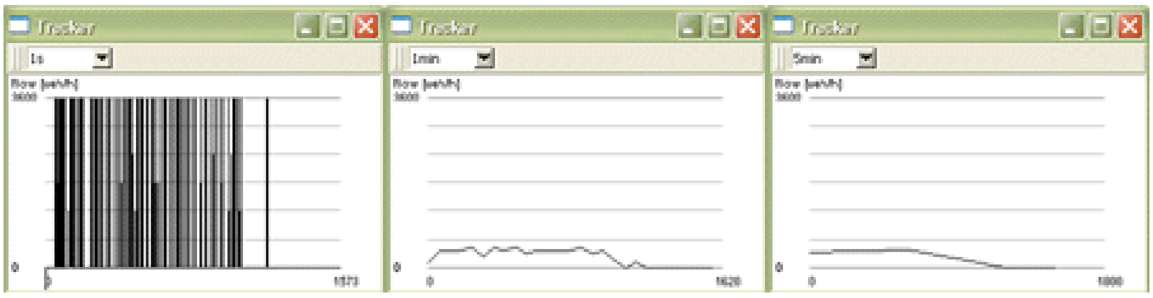
Name	Value	Dynamic
flow [veh/h]	0	✓
mean speed [m/s]	-1	✓
occupancy [%]	0	✓
mean vehicle length [m]	-1	✓
empty time [s]	0	✓
position [m]	200	✗
lane	2si_2	✗

Figure 7.3. A sample Parameter Window (for the number of vehicles within a simulation in this case)



You can change the aggregation time of the tracked values within this window using the combobox in this window's menu.

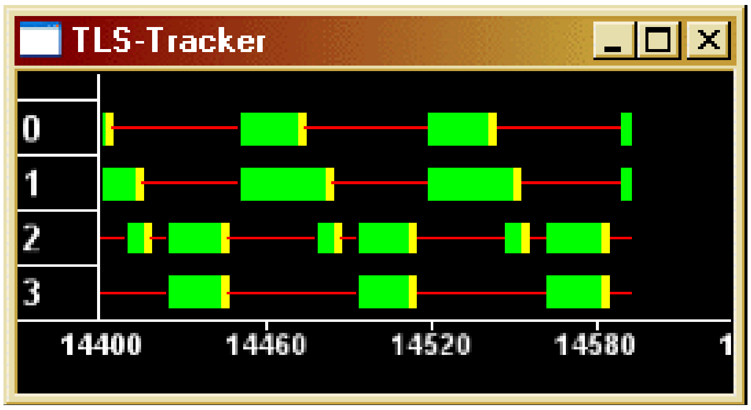
Figure 7.4. A sample usage of the aggregation option (for an induction loop in this case, for aggregation times of 1s, 1min, 5min (from left to right))



TL-Tracker Windows

If you position your mouse over one of the red, green or yellow traffic light-bars that show the state of the traffic light and press the right mouse button for at least one second, the appearing pop-up includes a menu entry "Show Phases". Choosing this menu item will show up a diagram that shows the states of the tl chronologically. Each pixel in x-direction shows the state of the tls of one second. The display contains the tl-states from the time the tracker has been opened, no scrolling aorund is supported.

Figure 7.5. A sample usage of the tls-tracker



Chapter 8. Tips, Tricks and Tools

We want to supply some additional information that did not fit into the descriptions within the previous chapters. The next chapters are possibly the most interesting ones of this document as they describe some possibilities to ease the work.

Appendix A. Naming Conventions

To ease the usage of the supplied files, all of which are within a XML-derivate, we use a naming convention for the file extensions to allow a distinction between the contents with a single look. The list of used extensions is showed below. We of course highly encourage you to use this pattern, but if you have a better idea, let us know.

- Configuration files:
- *.sumo.cfg
  Configuration file for SiMTraM (both command line and GUI-version)
- *.netc.cfg
  Configuration file for NETCONVERT-M
- *.netg.cfg
  Configuration file for NETGEN-M
- *.dua.cfg (sometimes also *.rou.cfg)
  Configuration file for DUAROUTER-M
- *.jtr.cfg
  Configuration file for JTRROUTER-M
- *.od2t.cfg
  Configuration file for OD2TRIPS
- Data files:
- *.net.xml
  SiMTraM - network file
  Contents: the SiMTraM-network including definitions for all streets, lanes and junctions
  Generated by: NETCONVERT-M or NETGEN-M
  Used by: SiMMTraM, GUISiMTraM, DUAROUTER-M, JTRROUTER-M, OD2TRIPS
- *.rou.xml
  sumo - routes file
  Contents: vehicle type definitions, route definitions, vehicle definitions
  Generated by: DUAROUTER-M, JTRROUTER-M or the user
  Used by: SiMTraM, GUISiMTraM, DUAROUTER-M
- *.add.xml
  sumo - additional definitions file
  Contents: The definitions of detectors to build, sources to build etc.
  Generated by: the user
  Used by: SiMTraM, GUISiMTraM
- *.out.xml
  sumo - output file
  Contents: The "raw" output with edges, lanes and vehicles on them
  Generated by: SiMTraM, GUISiMTraM
  Used by: the user
- *.edg.xml
  NETCONVERT-M - edges file
  Contents: definitions of edges to build the network from
  Generated by: the user

Used by: NETCONVERT-M

- *.nod.xml
  NETCONVERT-M - nodes file
  Contents: definitions of nodes to build the network from
  Generated by: the user
  Used by: NETCONVERT-M
- *.con.xml
  NETCONVERT-M- connection file
  Contents: definitions of connections between edges
  Generated by: the user
  Used by: NETCONVERT-M
- *.trips.xml
  trip definitions for DUAROUTER-M
  Contents: A list of trip definitions
  Generated by: the user
  Used by: DUAROUTER-M
- *.flows.xml
  flow definitions for JTRROUTER-M/DUAROUTER-M
  Contents: A list of flow definitions
  Generated by: the user
  Used by: JTRROUTER-M/DUAROUTER-M
- Other used file types
- *.inp
  VISSIM network files
- *.net
  VISUM network files
- *.shp, *.shx, *.dbf
  ArcView-network descriptions (shapes, shape indices, definitions)

Appendix B. Included Data

Configuration File Templates

You can find the templates for each of the package's application's configuration files within the folder `<SUMO_DIST>/data/cfg_templates`. These templates may be filled with your own values. Examples of fille configuration files may be found within the examples-section.

Included Examples