SUMO - Simulation of Urban MObility - User Documentation

Daniel Krajzewicz

Christian Rössel

SUMO - Simulation of Urban MObility - User Documentation

Daniel Krajzewicz
Christian Rössel

$Revision: 3977 $

Table of Contents

1. Introduction 1
What is SUMO? 1
Why open source? 1
Features 1
About this Document 1
Described Applications 1
Notation 2
Status 3
Call for Help 3
2. First Steps 4
Installing SUMO 4
Running the Examples 4
3. Traffic Simulations and SUMO 5
A short Introduction to Traffic Simulation Theory 5
Simulation types 5
Needed Data 6
The Workflow of Preparing a Simulation 6
SUMO 7
Main Software Paradigms 8
4. Network Generation 9
Introduction 9
Building Networks from own XML-descriptions 10
Nodes Descriptions 11
Edges Descriptions 12
Types Descriptions 15
Connection Descriptions 16
Building the Network 21
Converting other Input Data 22
Importing ArcView-databases 22
Importing Artemis-simulation folders 25
Importing Cell/Fastlane-networks 25
Importing VISSIM-networks 26
Importing VISUM-networks 27
Importing Elmar's converted NavTech-Files 30
Importing TIGER-databases 30
Further NETCONVERT Options 31
Setting default Values 31
Adding Turnarounds 31
Removing Geometry Nodes 32
Using Edges' maximum Speed Definitions in km/h 32
Importing Networks without Traffic Light Logics 32
Guessing On- and Off-Ramps 35
Converting from Geocoordinates 35
Adding inner-junction Traffic 36
Constraining the Input 36
Additional Output 37
Automatic Network Generation 38
Grid-like Networks 39
Spider-net-like Networks 40
Random Networks 41
Closing Thoughts (so far) 42
Recent Changes 42
Missing 43
5. Route Generation 44
Introduction 44
Common, mandatory Values 45
Building Routes from Scratch 45

Generating own, explicit Routes 46
Generating random Routes 50
Using the Junction Turning Ratio - Router 51
Using OD2TRIPS 52
Importing Routes from other Simulations 57
Importing Artemis-routes 57
Importing FastLane-routes 57
Importing VISSIM und VISUM-routes 58
Dynamic User Assignment and Alternative Routes 58
Automatic Iteration using 'dua-iterate.pl' 60
Additional Weights 61
Using Detectors and DFROUTER 62
Computing Detector Types 62
Computing Routes 63
Computing Flows 64
Saving Flows and other Values 65
Closing Thoughts (so far) 66
Recent Changes 66
Missing 66
6. Performing the Simulation 67
Output Generation 67
Detectors 67
Network State Dump 73
Aggregated Lane/Edge States (Edge/Lane-Dumps) 74
Net-Wide Vehicle Emission States & Travel Times 76
Vehicle-Oriented Trip Information 77
Vehicle Routes 78
Output coupled to Traffic Lights 79
Vehicles Handling Revisited 82
Emitter 82
Traffic Management and Other Structures 85
Traffic Lights 85
Public Transport 87
Variable Speed Signs (VSS) 89
Rerouter 89
Vehicle Classes 91
Using the Files in a correct Way 95
Other Topics 95
Simulation of Accidents 95
Missing 96
7. Simulation-GUI 97
Main Window Interface 97
Menu Bar 97
Tool Bar 99
Simulation Window Interfaces 100
Common Controls 100
Additional Microscopic View Controls 102
Additional Aggregated View Controls 103
Interacting with Objects 103
Display an Object's Name 103
Object Popup Menus 103
Object Selection 104
Parameter Windows 104
TL-Tracker Windows 106
Additional Geometry Files 107
Polygon Definitions 107
Point-of-interest Definitions 107
8. Tips, Tricks and Tools 109
Using Configuration Files 109
Additional Meta-Information 109
Additional Tools 110
Polygon Conversion 110
Helpers for DUA-Computation 111

Handling Routes and Route Alternatives 112
A. Naming Conventions 114
B. Included Data 117
Configuration File Templates 117
Included Examples 117
SIMPLE_NETS: Basic Examples 117
NETBUILD: Examples for NETCONVERT'S XML-Import 117
ROUTER: Examples for DUAROUTER and JTRROUTER 118
EXTENDED: Examples for using additional SUMO-structures 118

List of Figures

3.1. The different simulation granularities; from left to right: macroscopic, microscopic, sub-microscopic (within the circle: mesoscopic) 5
3.2. The difference between a space-continuous (top) and a space-discrete (bottom) simulation 6
3.3. Process of simulation with SUMO; (rounded: definite data types; boxes: applications; octagons: abstract data types) 7
4.1. Building a network 9
4.2. Building a network from XML-descriptions 10
4.3. Coordinate system used in SUMO 11
4.4. Unconstrained Network (zoom=2200) 17
4.5. Network with explicit edge-2-edge connections 17
4.6. Network with explicit lane-2-lane connections 18
4.7. Network with explicite prohibitions 20
4.8. **netgen --grid-net --grid-number=10 --grid-length=400 --output-file=MySUMOFile.net.xml** 39
4.9. **netgen --grid-net --grid-x-number=20 --grid-y-number=5 --grid-y-length=40 --grid-x-length=200 --output-file=MySUMOFile.net.xml** 39
4.10. **netgen --spider-net --spider-arm-number=10 --spider-circle-number=10 --spider-space-rad=100 --output-file=MySUMOFile.net.xml** 40
4.11. **netgen --spider-net --spider-arm-number=4 --spider-circle-number=3 --spider-space-rad=100 --output-file=MySUMOFile.net.xml** 40
4.12. **netgen --random-net -o MySUMOFile.net.xml --rand-iterations=200 --abs-rand** 41
5.1. Building routes (!!! Netzwerk fehlt) 45
5.2. A network where the usage of random routes causes an improper behaviour due to the mixture of rural and minor roads 50
5.3. Building trips from the OD-matrix 53
5.4. Example DUA-network (from "<SUMO_DIST>/data/examples/dua/dua3s*") 58
5.5. Sketch showing the effects of Christian Gawron dua-approach on route distribution within the network; blue color indicates that an edge is used within the step, red shows jams 59
6.1. Visualization of a bus stop in SUMO (from `<SUMO_DIST>/data/examples/extended/busses1`) 88
7.1. The GUI-Window with a loaded simulation (violet: names of the controls as used below) 97
7.2. The difference between simple (left) and full (right) geometry mode 101
7.3. A sample Parameter Window (for an induct loop in this case) 104
7.4. A sample Parameter Window (for the number of vehicles within a simulation in this case) 105
7.5. A sample usage of the aggregation option (for an induct loop in this case, for aggregation times of 1s, 1min, 5min (from left to right)) 106
7.6. A sample usage of the tls-tracker 106

List of Tables

1.1. Applications described within this document 1
4.1. Entries read by NETCONVERT 22
4.2. Possible entries to override NavTech-information 23
4.3. Imported VISUM-tables 27
6.1. Allowed vehicle class authority descriptions 92
6.2. Allowed vehicle class vehicle kind descriptions 92
B.1. Supported example folders 117

Chapter 1. Introduction

What is SUMO?

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.

Features

- High portability (using standard - c++ and portable libraries only)
- Collision free vehicle movement
- Different vehicle types
- Single-vehicle routing
- Multi-lane streets with lane changing
- Junction-based right-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 (Visum, Vissim, ArcView, XML-Descriptions)

About this Document

This document describes how to use each of the applications that come with the SUMO-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 (Linux/UNIX)	Description	Described in Chapter
NETCONVERT	netconvert.exe	sumo-netconvert	A network converter/importer	Chapter 4 [http://sumo.sourceforge.net/docs/gen/user_chp04.shtml]
NETGEN	netgen.exe	sumo-netgen	A generator of abstract networks	Chapter 4 [http://sumo.sourceforge.net/docs/gen/user_

Application	Application Name (Windows)	Application Name (Linux/UNIX)	Description	Described in Chapter
				chp04.shtml]
DFROUTER	dfrouter.exe	sumo-dfrouter	A router using detector flows	Chapter 5 [http://sumo.sourceforge.net/docs/gen/user_chp05.shtml]
DUAROUTER	duarouter.exe	sumo-durarouter	A router for dynamic user assignment	Chapter 5 [http://sumo.sourceforge.net/docs/gen/user_chp05.shtml]
JTRROUTER	jtrrouter.exe	sumo-jtrrouter	A router using junction turning ratios	Chapter 5 [http://sumo.sourceforge.net/docs/gen/user_chp05.shtml]
SUMO	sumo.exe	sumo	The microscopic simulation	Chapter 6 [http://sumo.sourceforge.net/docs/gen/user_chp06.shtml]
GUISIM	guisim.exe	sumo-guisim	The gui-version of the microscopic simulation	Chapter 7 [http://sumo.sourceforge.net/docs/gen/user_chp07.shtml]
POLYCONVERT	polyconvert.exe	sumo-polyconvert	A tool for importing polygons from other formats	Chapter 8.3.1 [http://sumo.sourceforge.net/docs/gen/user_chp08.shtml#user_chp08-tools-polyconvert]
other	---	---	---	Chapter 8 [http://sumo.sourceforge.net/docs/gen/user_chp08.shtml]

Please remark that you may also find the applications "NETEDIT" and "GIANT" within the source distribution. Both are not supported, not working properly and will be not discussed, herein.

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 SUMO-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 SUMO repository at sourceforge (http://sumo.sourceforge.net/ [http://sumo.sourceforge.net]). 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 SUMO

From version 0.8 on, we want not only supply the sources, but also the compiled binaries for MS Windows. We have abandonned the idea of releasing binaries for Linux due to large variety of the target systems.

If you are a Windows user, you should decide whether you primary want to use the software only or also extend it. In the first case, you should download the binaries. All needed libraries are included. In the latter case, please download the source distribution and compile it for your own. The description of the building process is found within a separate document located here [http://sumo.sourceforge.net/docs/gen/sumo_howto_building.shtml]. If you have built the package on a system not included within our binary distribution, please let us know and send it to us, so that we can include it into the pages.

When using Linux, you probably have to compile SUMO from the sources. A description about how to do this is located here [http://sumo.sourceforge.net/docs/gen/sumo_howto_building.shtml].

The SUMO-package also contains some further scripts which are located within the `tools`-folder. To execute them you'll need to have python [http://www.python.org/] and/or perl [http://www.perl.org/] installed.

Running the Examples

You may find some examples within the folder `<SUMO_DIST>/data` and its subfolders. Please remark that almost all applications are command line tools, what means that no window pops up if you start it, you have to open a shell window first.

Most of the examples come with a configuration-file so that may be directly run from the command line. Read chapter "Using Configuration Files [http://sumo.sourceforge.net/docs/gen/user_chp08.shtml#user_chp08-configs]" for further information on how to use configuration files.

We apologize that a documentation on the examples does not exist (yet). Nonetheless please feel invited to take a look at the tutorial [http://sumo.sourceforge.net/wiki/index.php/Tutorial](s) available at our wiki [http://sumo.sourceforge.net/wiki/index.php/Main_Page].

Chapter 3. Traffic Simulations and SUMO

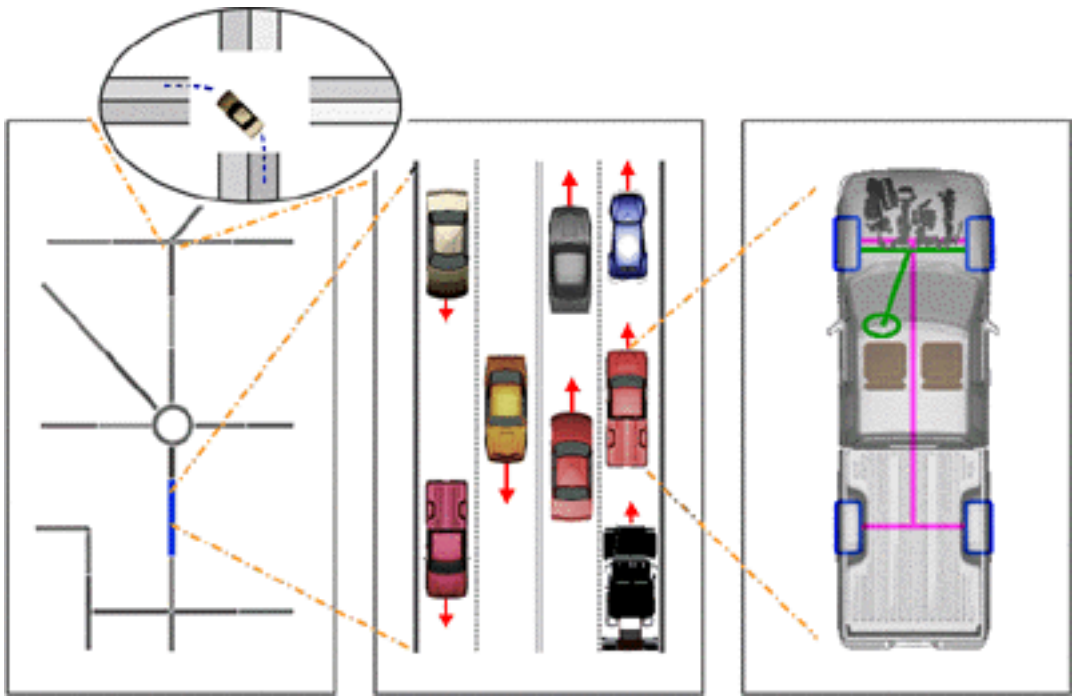
A short Introduction to Traffic Simulation Theory

Simulation types

SUMO is a microscopic, space continuous and time discrete traffic simulation.

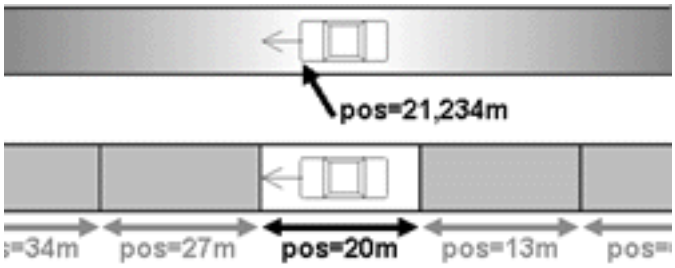
In traffic research four classes of traffic flow models are distinguished according to the level of detail of the simulation. In *macroscopic* models traffic flow is the basic entity. *Microscopic* models simulate the movement of every single vehicle on the street, mostly assuming that the behaviour 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 (see !!!). *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 (see !!! for furhter details). 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]).

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 SUMO knows its route. 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 [http://sumo.sourceforge.net/docs/gen/user_chp04.shtml#user_chp04-xml_descriptions]") or if you have some digital networks SUMO can import, convert them (described in chapter 4, "Converting other Input Data [ht-

tp://sumo.sourceforge.net/docs/gen/user_chp04.shtml#user_chp04-other_input]")

2. Build the demand

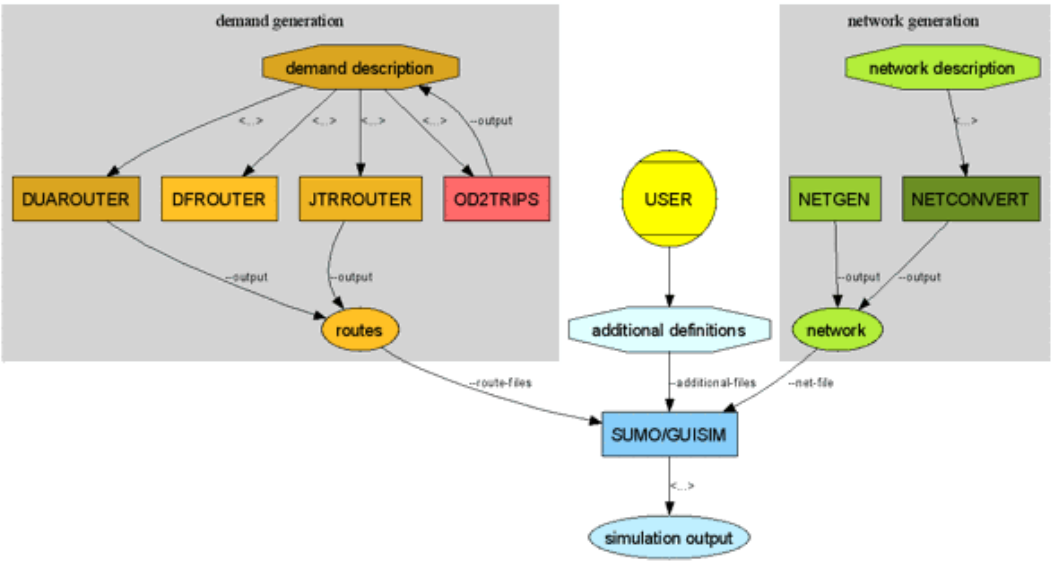
   Build your own movements using either by a) describing explicit vehicle routes (see chapter 5, "Using Trip Definitions [http://sumo.sourceforge.net/docs/gen/user_chp05.shtml#user_chp05-explicit]"), b) using flows and turning percentages only (see chapter 5, "Using the Junction Turning Ratio - Router [http://sumo.sourceforge.net/docs/gen/user_chp05.shtml#user_chp05-own_routes-jtr]"), c) generating random routes (see chapter 5, "Generating random Routes [http://sumo.sourceforge.net/docs/gen/user_chp05.shtml#user_chp05-own_routes-random]"), d) importing OD-matrices (see chapter "Using OD2TRIPS [http://sumo.sourceforge.net/docs/gen/user_chp05.shtml#user_chp05-od2trips]" or "Using Flow Definitions [http://sumo.sourceforge.net/docs/gen/user_chp05.shtml#user_chp05-explicite-flows]"), or e) importing routes you own (see chapter 5, "Importing Routes [http://sumo.sourceforge.net/docs/gen/user_chp05.shtml#user_chp05-import_routes]").

3. If needed, compute the dynamic user assignment (described in chapter 5, "Dynamic User Assignment [http://sumo.sourceforge.net/docs/gen/user_chp05.shtml#user_chp05-dua]")

4. Perform the simulation (described in chapter 6, "Performing the Simulation [http://sumo.sourceforge.net/docs/gen/user_chp06.shtml]") to get your desired output

This process is also visualised within the next figure.

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



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]".

SUMO

Main Software Paradigms

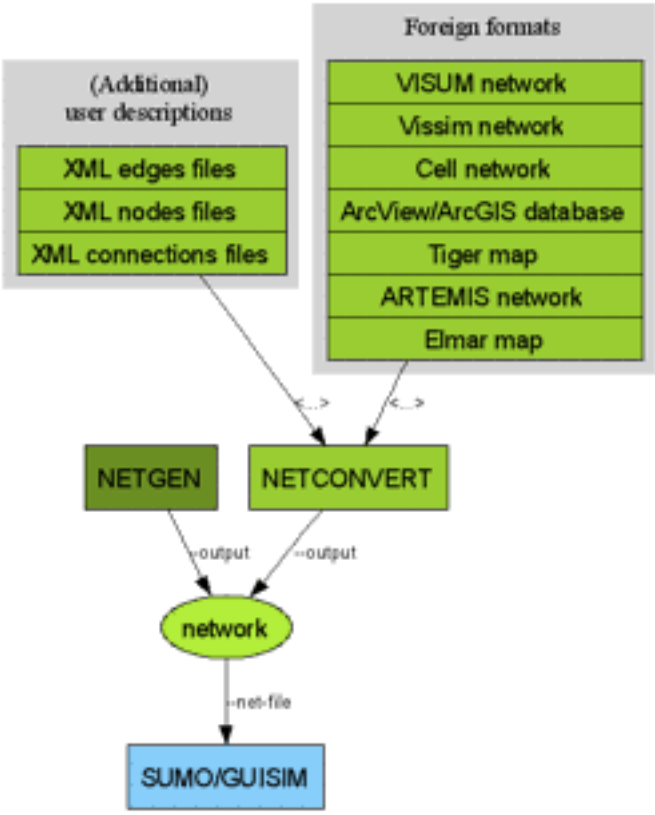
Two basic design goals are approached: the software shall be fast and it shall be portable. Due to this, the very first versions were developed to be run from the command line only - no graphical interface was supplied at first and all parameter had to be inserted by hand. This should increase the execution 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 SUMO 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 SUMO is not meant to be edited by hand and will also not be described herein due to its complexity. SUMO networks can be build by either converting an existing map or by using NETGEN to generate basic, abstract road maps. The following figure shows the function of NETCONVERT and NETGEN 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 SUMO using the NETCONVERT tool. By now, NETCONVERT 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]"
- *ARTEMIS* by Prof. Peter Hidas (a microscopic traffic simulation package), see chapter "Importing Artemis-simulation folders [#user_chp04-other_input-artemis]"
- *FastLane* by Christian Gawron (a mesoscopic traffic simulation package), see chapter "Importing Cell/Fastlane-networks [#user_chp04-other_input-fastlane]"
- *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 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 --visum=MyVisumNet.inp --output-file=MySUMONet.net.xml
```

The parameter `--output-file` has the default value "`net.net.xml`". That means that both NETCONVERT and NETGEN will save the generated file as "`net.net.xml`" if the option is not set. Please note, that NETCONVERT has to be started from the command line. There is no graphical interface available, yet.

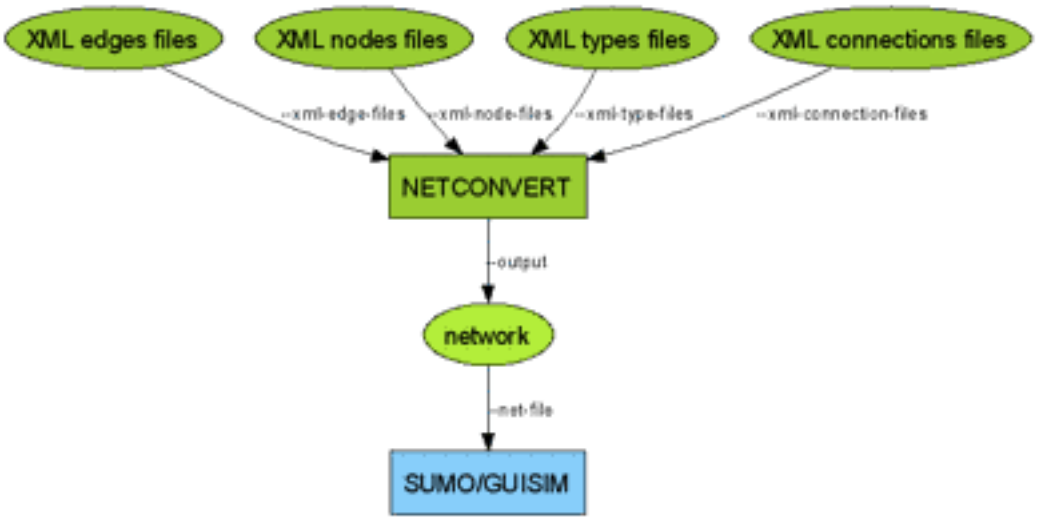
The following subchapters will describe more deeply how NETCONVERT and NETGEN are used, also discussing some problems with each of the import formats NETCONVERT supports. Please remind the option to name the output generated by both applications:

`( --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 XML-descriptions

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 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 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. *(TBD: would further types make sense?)*

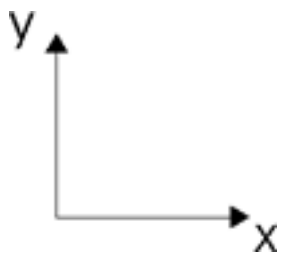
When writing your nodes-file, please do not forget to embed your node definitions into an opening and a closing "tag". A complete file should like the example below, which is the node file "`cross3l.nod.xml`" for the examples "`<SUMO_DIST>/data/examples/netbuild/types/cross_usingtypes/`" and "`<SUMO_DIST>/data/examples/netbuild/types/cross_notypes/`"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



Since version 0.9.4 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>]* <2D_POINT>"] [spread_type="center"] [function=("source" | "sink" | "normal")]/>`.

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 "`--omit-corrupt-edges`" 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 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. Using `function` you can define whether the edge is a "`normal`" edge, a "`source`", or a "`sink`" edge. The default is "`normal`", of course. This information is used for routing purposes (see "Using the Junction Turning Ratio - Router") and for vehicle emission (see "Vehicles Handling Revisited"). Briefly, vehicles may be inserted on normal and source edges (although the insertion procedure changes, see "Vehicles Handling Revisited") and may leave the network on normal and sink edges.

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_options-kmh_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
  - `function`: Information whether the edge is a plain edge, a source edge, or a sink edge. The value must be one of "`normal`", "`sink`", "`source`".
- The edges shape:

  - `shape`: List of positions; each position is encoded in x,y (do not separate the numbers with a space!) in meters; an edge's shape definition must of course be at least two positions long; an example: `shape="0,0 0,100"` describes a vertical edge of one hundred meters.
  - `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="11.1
```

```
   <edge id="1si" fromnode="m1" tonode="0" priority="3" nolanes="3" speed="13.8
   <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="11.1
   <edge id="2si" fromnode="m2" tonode="0" priority="3" nolanes="3" speed="13.8
   <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="11.1
   <edge id="3si" fromnode="m3" tonode="0" priority="3" nolanes="3" speed="13.8
   <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="11.1
   <edge id="4si" fromnode="m4" tonode="0" priority="3" nolanes="3" speed="13.8
   <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

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="13.8
      <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="13.8
      <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 [ht-

tp://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>" [function=( "source" | "sink" | "normal" )]/>`.

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
- `function`: The function of the referencing edge; must be one of "`source`", "`sink`", or "`normal`". "`normal`" is the default.

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"/>
```

```
    <type id="b" priority="2" nolanes="2" speed="11.111"/>
    <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

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:

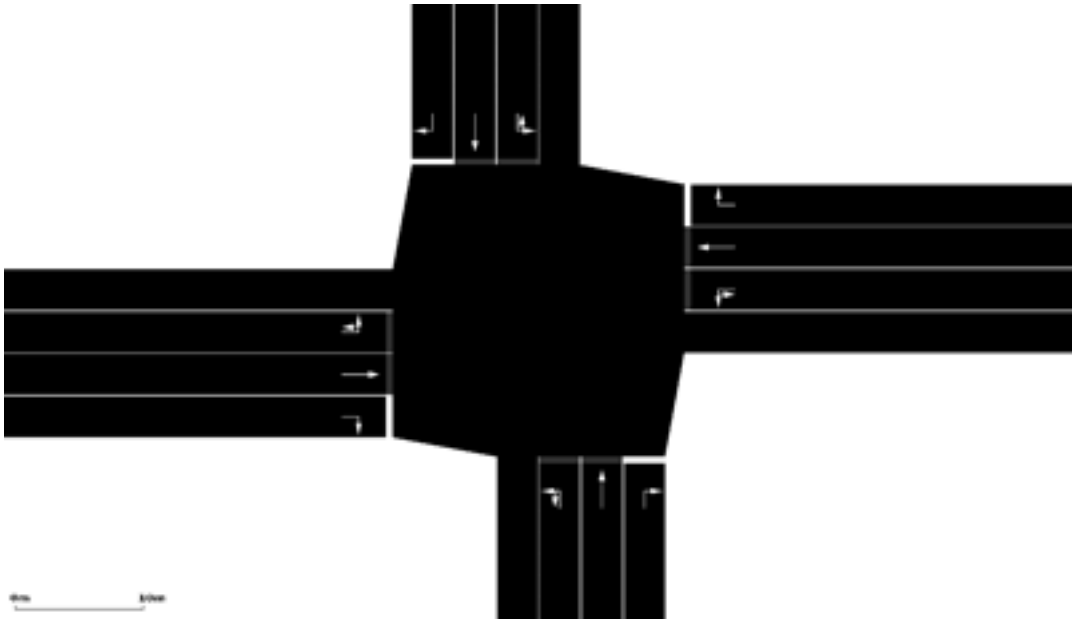
- from: The name of the edge the vehicles leave
- to: The name of the edge the vehicles may reach when leaving "from"

When using this kind of input, NETCONVERT 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)

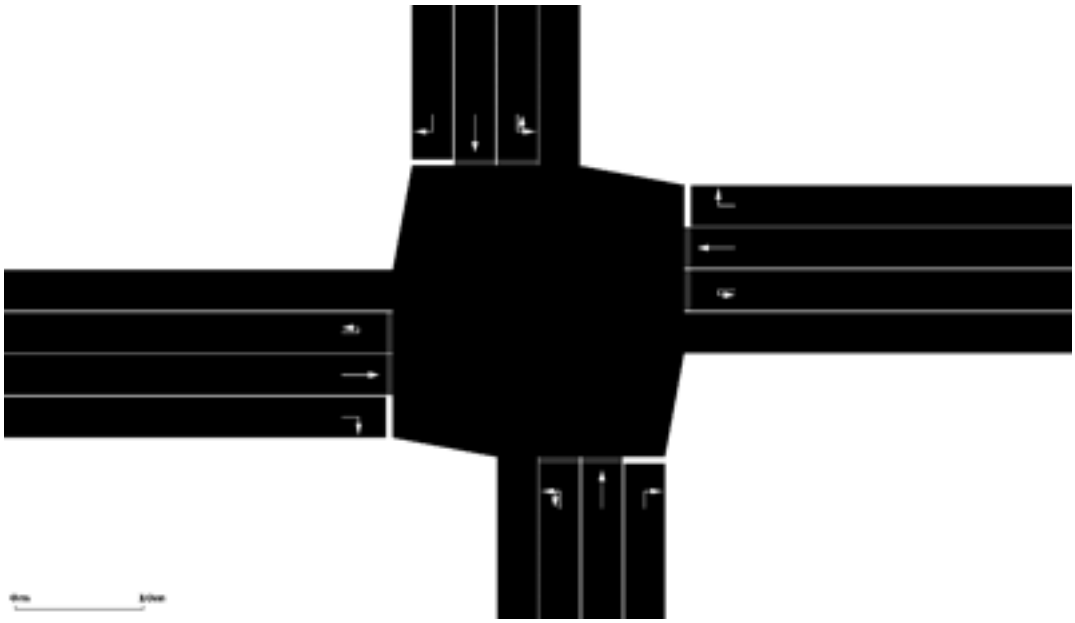


The example *<SUMO_DIST>/*`data/examples/netbuild/connections/cross3l_edge2edge_conns/"` shows what happens when one uses connections to limit the number of reachable edges. To do this we built a connections file where we say that the horizontal edges ("1si" and "2si") have only connections to the edges 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 left-moving connections.

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

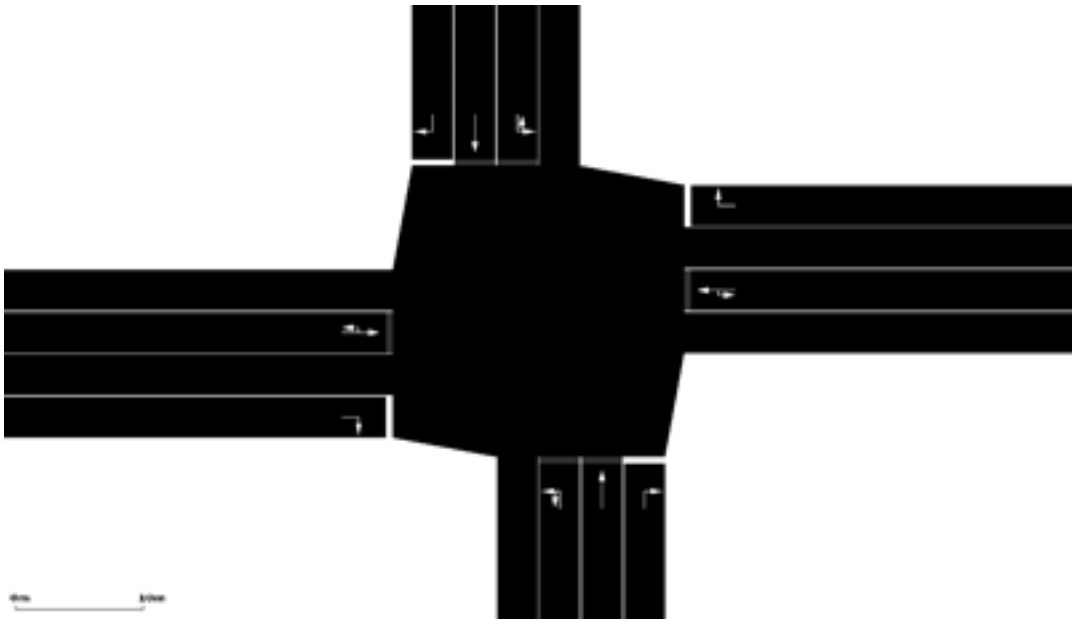


In the second example located in *<SUMO_DIST>/*`data/examples/netbuild/connections/cross3l_laneslane_conns/`" we additionally describe which lanes shall be connected. The according connections file says that the connections going straight shall be start at the second lane of the incoming edges:

```
<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

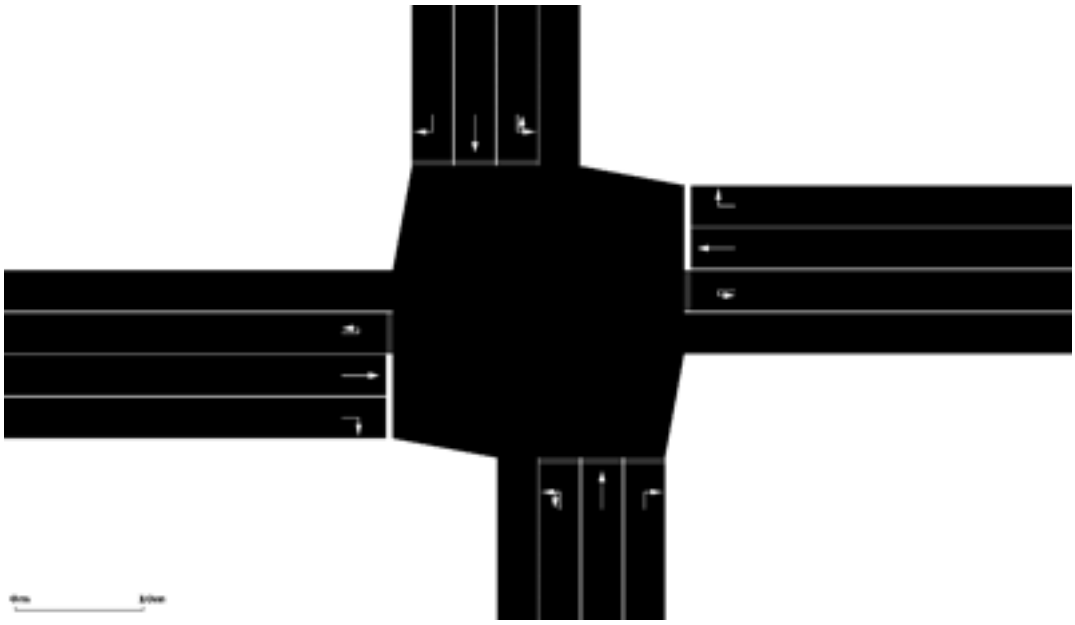
Since version 0.9.6 you can also let vehicles passing a connection between two edges wait for another stream. Let's take another look at "Network with explicit edge-2-edge connections" above. Here, all right-moving vehicles may drive. The following definition within the connections file lets vehicles 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 l
    <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 rig
    <prohibition prohibitor="1si->2o" prohibited="3si->2o"/>
    <!-- prohibit moving straight from bottom to top by straight from left to ri
    <prohibition prohibitor="1si->2o" prohibited="3si->4o"/>
    <!-- prohibit moving left from bottom to right by straight from left to righ
    <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 counter-clockwise 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 --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 --xml-node-files=MyNodes.nod.xml --xml-edge-files=MyEdges.edg.xml \
   --xml-connection-files=MyConnections.con.xml --xml-type-files=MyTypes.typ.xm
   --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 "`--omit-corrupt-edges`" to NETCONVERT. In this case, edges which are not defined properly, are omitted, but NETCONVERT 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*
`--omit-corrupt-edges`	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_options-importing_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

Converting other Input Data

Large maps cannot be written by hand. We use maps from NavTech stored in the ArcView database format and maps from other simulation suppliers such as ptv within our projects and both are too large for this. We will now explain how to convert such data. We will not give any introduction into the formats/simulations themselves or compare their quality, but we will describe what is being imported and what problems may arise during the conversion.

Importing ArcView-databases

NETCONVERT is able to directly read binary NavTech's ArcView databases. To convert such databases, you need at least three files: a file with the extension "`.dbf`", one with the extension ".`shp`" and one with the extension "`.shx`". Additionally, having a projection file with the extension ".proj" is of benefit. Since version 0.9.2 we do not suply the possibility to use different names for the files, so all files should have the same name besides the extension. To build your network from an ArcView-database use the option "`--arcview=<FILENAME_WITHOUT_EXTENSION>`":

```
netconvert --arcview=MyArcViewDB --output-file=MySUMONet.net.xml
```

This call will force NETCONVERT to read the files "`MyArcViewDB.dbf`", "`MyArcViewDB.shx`", and "`MyArcViewDB.shp`" (and possibly "`MyArcViewDB.proj`" and to generate a network named "`MySUMONet.net.xml`". We have been asked which fields are read from ArcView-files. As said before, the reader was build to read ArcView-files containing road networks from NavTech. Due to this the following fields are used as default:

Table 4.1. Entries read by NETCONVERT

Entity Name	Description
LINK_ID	The id of an edge
ST_NAME	The name of an edge (not really used)
REF_IN_ID	The name of the node the edge starts at
NREF_IN_ID	The name of the node the edge ends at
ST_TYP_AFT	The type of the street (not really used)

Entity Name	Description
SPEED_CAT	Speed category
LANE_CAT	Lane category
FUNC_CLASS	Road class, used to determine the priority

The problem is, that not all networks stored as ArcView-databases also use this naming scheme. During some further work with ArcView-networks, some further options got necessary which allow to name the fields the used database contains. The column the street name shall be read from may be specified using `--arcview.street-id <STREET_NAME_COLUMN_NAME>`. You can also name the columns the names of the edges' origin and destination nodes shall be read from using `--arcview.from-id <START_NODE_ID_COLUMN_NAME>` and `--arcview.to-id <END_NODE_ID_COLUMN_NAME>`. If the no information about the starting/ending nodes is given and your database does not contain the columns "REF_IN_ID" and "NREF_IN_ID", nodes will be placed into the network at the positions the streets end.

Since version 0.9.2 we also allow to override the rather "fuzzy" information about an edge's attributes from NavTech using own fields:

Table 4.2. Possible entries to override NavTech-information

Entity Name	Description
SPEED	The speed in m/s (see also "Using Edges' maximum Speed Definitions in km/h" [#user_chp04-further_options-kmh_speed])
NOLANES	The number of lanes
rnol	The number of lanes
PRIORITY	The priority

This idea came from John Michael Calandrino.

Some databases do not contain explicite information about the edges' attributes (number of lanes, priority, allowed speed) at all. Since version 0.9.4 you can use types as described in "Types Descriptions" to describe your edges' attributes. You have to name the column to retrieve the information about a street's type from using `--arcview.type-id <TYPE_ID_COLUMN_NAME>`. Of course, you have then to supply a type-file using `--xml-type-files <TYPES_FILE>` (or `--types` or `-t`). If something fails with the types or the explicite values, you can catch it using `--arcview.use-defaults-on-failure`. Besides this, you can specify your own connections using `--xml-connection-files <CONNECTIONS_FILE>` (or `--xml-connections` or `-x`, see "Connection Descriptions [#user_chp04-xml_descriptions-connections]").

ArcView-networks are (mostly?) encoded using geocoordinates which have to be converted to the cartesian coordinates system used by SUMO. Our current implementation is not yet fully developed, it works for the most cases, but you should not be surprised if it fails with a certain network. Contact us in this case, please. To describe how to convert the coordinates, you should know in which UTM-zone your network is located. Pass this to NETCONVERT using `--arcview.utm <ORIGINAL_UTM_ZONE>`. If the conversion can not be initialised, you may additionally use `--arcview.guess-projection` to let NETCONVERT guess the conversion by him own.

Specific options:

`--arcview <ARCVIEW_PREFIX>`	Loads definitions from "`<ARCVIEW_PREFIX>.shp`", "`<ARCVIEW_PREFIX>.dbf`" and "`<ARCVIEW_PREFIX>.shx`". *Optional, type:filename-prefix, default: none*

`--arcview.street-id <STREET_NAME_COLUMN_NAME>`	This option tells NETCONVERT which of the columns within the ArcView-database to read shall be used as the source of street names. If given, your database must contain this column, and the values must be unique for each street. *Optional, type:string, default: none*
`--arcview.from-id <START_NODE_ID_COLUMN_NAME>`	This option tells NETCONVERT which of the columns within the ArcView-database to read shall be used as the source of the information from which node the street starts. If given, your database must contain this column. *Optional, type:string, default: none*
`--arcview.to-id <END_NODE_ID_COLUMN_NAME>`	This option tells NETCONVERT which of the columns within the ArcView-database to read shall be used as the source of the information at which node the street ends. If given, your database must contain this column. *Optional, type:string, default: none*
`--arcview.type-id <TYPE_ID_COLUMN_NAME>`	This option tells NETCONVERT which of the columns within the ArcView-database to read shall be used as the source of the information about the edge's type. Using this information, you can use type definitions as described in "Types Descriptions" to determine your edges' attributes. If given, your database must contain this column. *Optional, type:string, default: none*
`--arcview.use-defaults-on-failure`	If a type could not be resolved or is invalid or any of the explicite information about an edge was invalid, this option forces NETCONVERT to use the default type values for the current street. If not set, and one of the cases occures, the application's behaviour is not determined. *Optional, type:bool, default: false*
`--arcview.all-bidi`	Forces NETCONVERT to insert all edges bidirectional. *Optional (pregiven), type:bool, default: false*
`--arcview.utm <ORIGINAL_UTM_ZONE>`	This value describes in which UTM-zone your network is located. The default is 32 being a place somwhere in western Germany. You should change this value if importing networks located somewhere else in the world. *Optional (pregiven), type:int, default: 32*
`--arcview.guess-projection`	If building a converter using the given UTM-zone fails, this option tries to guess the project if set. Of course, this may also fail... *Optional (pregiven), type:bool, default: false*

See also:

- "Importing Networks without Traffic Light Logics [#user_chp04-further_options-importing_notls]"
- "Using Edges' maximum Speed Definitions in km/h" [#user_chp04-further_options-kmh_speed]
- "Guessing On- and Off-Ramps [#user_chp04-further_options-guessing_ramps]"
- "Adding Turnarounds [#user_chp04-further_options-turnarounds]"
- "Types Descriptions [#user_chp04-xml_descriptions-types]"
- "Connection Descriptions [#user_chp04-xml_descriptions-connections]"

Examples: none yet

Recent changes:

- versions earlier than 0.9.2 allow to use an explicit filename for both the .dbf and the .shp file us-

ing "`--arcview-dbf`" and "`--arcview-shp`". This was abondonned, because of the need to use .shx-files, also.
- ArcView-import has been completely redesigned for version 0.9.4. All options but "`--arcview <ARCVIEW_PREFIX>`" are not available in versions prior to 0.9.4

Importing Artemis-simulation folders

To import Artemis-network descriptions, start NETCONVERT with the following parameter:

```
netconvert --artemis=<PATH> --output-file=MySUMOFile.net.xml
```

This should build the network "`MySUMOFile.net.xml`" which contains the build network that may be used by SUMO. *<PATH>* is the path to (the name of) the folder that contains the files that make up the description of an ARTEMIS-simulation.

Imported information:

- Nodes (id, position, type)
- Links (origin, destination, id, speed, number of lanes, length)
- Signals
- Signal Groups (node id, group, connection, start & end phase)
- Signal Phases (node id, start, phase, percentage)
- Segments (link, position on link, geometrical position)
- Lanes (lane number, section number, begin and end position)
- HVdests (origin and destination lane)

We have to import the HVdests to know which sources and sinks we have to build.

Known problems:

- The connections between lanes may be not the same as in the input
- The positions of the lanes are not correct
- The times the traffic lights switch at may not be the same as within the input
- Additional source and sink links must be build
- Possible speed changes at segments are not supported

Artemis simulation description also holds definitions of the traffic flows to use. They are not parsed by the NETCONVERT - module, but may be passed to ROUTER to gain the according routes.

Specific options:

`( --artemis-path | --artemis ) <PATH>`	Loads definitions from the given path. *Optional, type:path, default: none*

Known problems:

Caution

The import of ARTEMIS is not longer supplied and seems to be buggy.

Examples: none yet (we do not own a network we could give away for legal reasons)

Importing Cell/Fastlane-networks

FastLane, developed at the ZAIK [http://www.zaik.uni-koeln.de/], is a mesoscopic traffic simulation. The network description consists of a file containing edges and a second one containing nodes. Due to this, you need to supply two values as input parameter and the call looks like this:

```
netconvert --cell-nodes=<CELL_NODE_FILE> --cell-edges=<CELL_EDGE_FILE> \
   --output-file=MySUMOFile.net.xml
```

Of course, both files must belong to the same network. As FastLane is a mesoscopic simulation, sometime the number of an edge's lanes is not given. Instead, FastLane uses the capacity. In such cases, the number of lanes is computed roughly using the edge's capacity. We assume a linear dependency for this, currently, although this may not be the best solution. So, the number of lanes is computed as EDGE_CAPACITY/NORM. You may change the norm from its default of 20000 using the option `--capacity-norm`.

You can also supply a type-file using `--xml-type-files <TYPES_FILE>` (or `--types` or `-t`, see "Types Descriptions [#user_chp04-xml_descriptions-types]") and specify your own connections using `--xml-connection-files <CONNECTIONS_FILE>` (or `--xml-connections` or `-x`, see "Connection Descriptions [#user_chp04-xml_descriptions-connections]").

Specific options:

`( --cell-node-file | --cell-nodes ) <FILE>`	Reads nodes definitions from `<FILE>`. *Optional, type:path, default: none*
`( --cell-edge-file | --cell-edges ) <FILE>`	Reads edges definitions from `<FILE>`. *Optional, type:path, default: none*
`( --capacity-norm | -N ) <FLOAT>`	Sets the capacity norm to the given value. *Optional (pregiven), type:float, default: 20000*

See also:

- "Importing Networks without Traffic Light Logics [#user_chp04-further_options-importing_notls]"
- "Guessing On- and Off-Ramps [#user_chp04-further_options-guessing_ramps]"
- "Adding Turnarounds [#user_chp04-further_options-turnarounds]"
- Converting from Geocoordinates
- "Types Descriptions [#user_chp04-xml_descriptions-types]"
- "Connection Descriptions [#user_chp04-xml_descriptions-connections]"

Examples: none yet (we do not own a network we could give away for legal reasons)

Importing VISSIM-networks

Although Vissim is a microscopic simulation as SUMO is, it follows a completely different concept of modelling traffic. Due to this, the import is quite clumsy and may not work with all networks. Also, we have to insert additional edges into our networks to simulate the Vissim-parking places, originally being nodes, which we do not have. An usage example could be this one:

```
netconvert --vissim=<VISSIM_FILE> --output-file=MySUMOFile.net.xml
```

Vissim-networks do possibly not contain explicit definitions of an edge's speed. We have to propagate once set velocities, but even then some edges' speeds may not be defined. The option "`--vissim.default-speed`" may change the default speed used in the case an edge's speed is not defined. The default value for this parameter is 13.89m/s, being around 50km/h. The second parameter "`--vissim.speed-norm`" describes the factor to multiply a described flows maximum velocity to gain the velocity to use. The default value is 1.

Furthermore, as we have to use geometrical heuristics, a further factor steers the process of converting Vissim-networks: simply spoken, "`--vissim.offset`" holds the information how near two nodes must be (in meters) to be joined.

During import, different actions must be done which may yield in some loss of data and may be watched in part by setting the verbose option. The additional warnings the import of Vissim-networks generates will be described in a further document.

Specific options:

`--vissim <FILE>`	Loads definitions from the given file (should end with ".inp"). *Optional, type:filename, default: none*
`--vissim.speed-norm <SPEED_NORM_FACTOR>`	A factor that is multiplied with a streams maximum velocity to determine the velocity to use for an edge. *Optional (pregiven), type:float, default: 1*
`--vissim.default-speed <DEFAULT_SPEED>`	The default speed to use for a street when no information is available. *Optional (pregiven), type:float, default: 13.89 (m/s)*
`--vissim.offset <MAX_JOIN_DISTANCE>`	This value is used to determine whether two Vissim-structures are near enough to be joined into a node. *Optional (pregiven), type:float, default: 5 (m)*

Known problems:

- Works with German networks only
- All actuated traffic lights are mapped onto the same type of traffic light (MSActuatedTrafficLight)
- Additional source and sink links must be build

Examples: none yet (we do not own a network we could give away for legal reasons)

Recent changes:

- `--vissim.speed-norm`, `--vissim.default-speed`, and `--vissim.offset` were named `--vissim-speed-norm`, `--vissim-default-speed`, and `--vissim-offset` in version prior to 0.9.6.

Importing VISUM-networks

Visum is a macroscopic simulation developed by ptv. NETCONVERT is capable to read VISUM-networks written as `.net` files. An usage import call could be this one:

```
netconvert --visum=<VISUM_FILE> --output-file=MySUMOFile.net.xml
```

As the network description may not contain any information about the number of lanes, we have to generate it from the street's flow. The computation is done by dividing the flow through a fix value, 2000 by default. This yields in a realistic network but fails on 'feeder roads' where vehicles are emitted.

You can also specify your own connections using `--xml-connection-files <CONNECTIONS_FILE>` (or `--xml-connections` or `-x`, see "Connection Descriptions [#user_chp04-xml_descriptions-connections]").

Not all parts of the Visum-Format are read; below you'll find a table which contains the information which Visum-tables are imported.

Table 4.3. Imported VISUM-tables

Table Name	Description	Imported values
VSYS	Traffic modes	VSysCode (CODE)
		VSysMode (TYP)
STRECKENTYP	Edge types	Nr
		v0-IV (V0IV)
		Rang
		Kap-IV (KAPIV)
KNOTEN	Nodes	Nr
		XKoord
		YKoord
BEZIRK	Districts	Nr
		NAME (unused later)
		Proz_Q
		Proz_Z
		XKoord
		YKoord
STRECKE (STRECKEN)	Edges	Nr
		VonKnot (VonKnotNr)
		NachKnot (NachKnotNr)
		Typ (TypNr)
		Einbahn
ANBINDUNG	District connections	BezNr
		KnotNr
		Proz
		t0-IV
		Typ
		Richtung
ABBIEGEBEZIEHUNG (ABBIEGER)	Edge Connections	VonKnot (VonKnotNr)
		UeberKnot (UeberKnotNr)
		NachKnot (NachKnotNr)
		VSysCode (VSYSSET)
STRECKENPOLY	Edge geometries	VonKnot (VonKnotNr)
		NachKnot (NachKnotNr)
		INDEX
		XKoord
		YKoord
FAHRSTREIFEN	Lane descriptions	KNOTNR
		STRNR
		FSNR
		RICHTTYP
		LAENGE
LSA (SIGNALANLAGE)	TLS	Nr
		Umlaufzeit (UMLZEIT)

Table Name	Description	Imported values
		StdZwischenzeit (STDZWZEIT)
		PhasenBasiert
KNOTENZULSA (SIGNALANLAGEZUKNOTEN)	Nodes->TLS	KnotNr
		LsaNr
LSASIGNALGRUPPE (SIGNALGRUPPE)	Signal groups	Nr
		LsaNr
		GzStart (GRUENANF)
		GzEnd (GRUENENDE)
ABBZULSASIGNALGRUPPE (SIGNALGRUPPEZUABBIEGER)	Edge connections->TLS	SGNR (SIGNALGRUPPENNR)
		LsaNr
		VonKnot / VONSTRNR
		NachKnot / NACHSTRNR
		UeberKnot (UeberKnotNr)
		LsaNr
LSAPHASE (PHASE)	Signal phases	Nr
		LsaNr
		GzStart (GRUENANF)
		GzEnd (GRUENENDE)
LSASIGNALGRUPPEZULS-APHASE	Signal groups->phases	PsNr
		LsaNr
		SGNR
FAHRSTREIFENABBIEGER	Lane-to-lane descriptions	KNOT (KNOTNR)
		VONSTR (VONSTRNR)
		NACHSTR (NACHSTRNR)
		VONFSNR
		NACHFSNR

Specific options:

`--visum <FILE>` Loads definitions from the given file (should end with ".net"). *Optional, type:filename, default: none*

See also:

- "Importing Networks without Traffic Light Logics [#user_chp04-further_options-importing_notls]"
- "Guessing On- and Off-Ramps [#user_chp04-further_options-guessing_ramps]"
- "Adding Turnarounds [#user_chp04-further_options-turnarounds]"
- Converting from Geocoordinates
- "Connection Descriptions [#user_chp04-xml_descriptions-connections]"

Examples: none yet (we do not own a network we could give away for legal reasons)

Recent changes:

- Since version 0.9.4, NETCONVERT can also convert VISION-networks (VISUM version 0.9.3).

Please remark for this case, that we do not have a VISION-network which contains TLS-definitions and due to this the import of these may fail.

Importing Elmar's converted NavTech-Files

You can convert both the splitted and the unsplitted version of files generated by Elmar from NavTech-files. There seems to be no difference between the results in the networks' topologies, but the names of junctions and roads change. The option `--elmar` loads the splitted definitions, `--elmar2` the unsplitted. Both options await the prefix as generated by Elmar's converter, an optional path is allowed. Example:

```
netconvert --elmar=berlin_ --output-file=MySUMOFile.net.xml
```

Imports the descriptions of nodes from "`berlin_nodes.txt`" and the edges from "`berlin_links.txt`".

You can also specify your own connections using `--xml-connection-files <CONNECTIONS_FILE>` (or `--xml-connections` or `-x`, see "Connection Descriptions [#user_chp04-xml_descriptions-connections]").

Specific options:

`--elmar <NET_PREFIX>`	Loads the splitted versions of the files behind `<NET_PREFIX>`. *Optional, type:filename-prefix, default: none*
`--elmar2 <NET_PREFIX>`	Loads the unsplitted versions of the files behind `<NET_PREFIX>`. *Optional, type:filename-prefix, default: none*

See also:

- "Importing Networks without Traffic Light Logics [#user_chp04-further_options-importing_notls]"
- "Guessing On- and Off-Ramps [#user_chp04-further_options-guessing_ramps]"
- "Adding Turnarounds [#user_chp04-further_options-turnarounds]"
- Converting from Geocoordinates
- "Connection Descriptions [#user_chp04-xml_descriptions-connections]"

Examples: none yet (we do not own a network we could give away for legal reasons)

Importing TIGER-databases

This import function is in a rather experimental state. We need someone who owns a network she/he knows and who could give us an advice whether the import work as expected. You still may try it out using the option `--tiger=<FILE_PREFIX>`.

You can also specify your own connections using `--xml-connection-files <CONNECTIONS_FILE>` (or `--xml-connections` or `-x`, see "Connection Descriptions [#user_chp04-xml_descriptions-connections]").

See also:

- "Importing Networks without Traffic Light Logics [#user_chp04-further_options-importing_notls]"
- "Guessing On- and Off-Ramps [#user_chp04-further_options-guessing_ramps]"
- "Adding Turnarounds [#user_chp04-further_options-turnarounds]"
- Converting from Geocoordinates

- "Connection Descriptions [#user_chp04-xml_descriptions-connections]"

Examples: none yet

Further NETCONVERT 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

Adding Turnarounds

Normally, turnarounds are added as a possible edge continuations and play an important role during network building (see [Krajzewicz_et_al2005_2] [http://sumo.sourceforge.net/docs/bibliography.shtml#Krajzewicz2005_2]). Still, one may want not to add them. In this cases, it is possible to disallow their appending using option "`--no-turnarounds`".

Specific option:

--no-turnarounds *Optional (pregiven), type:bool, default: false*

This options may be used while importing the following formats:

- FastLane/CELL
- ARCVIEW-data base files
- XML-descriptions

Recent changes:

- in versions earlier than 0.9.3, turnarounds were not added per default. Instead the option "-

`-append-turnarounds`" has forced NETCONVERT to add them.

Removing Geometry Nodes

In most input networks one may find nodes where one street comes in and one with the same attributes goes out or where two parallel edges come in and two (with the same attributes) come out. Such nodes have mostly no meaning (maybe besides the additional possibility to make a U-turn) and may be removed. The removal of such nodes increases the simulation speed due to a smaller number of edges to process during each time step. To remove such nodes and join the incoming and outgoing edges use "`--remove-geometry`". The removal of nodes preserves the geometry of edges by adding a further geometry point at the removed node's position.

Specific option:

(--remove-geometry | -R) *Optional (pregiven), type:bool, default: false*

This options may be used in conjunction with all import formats.

Recent changes:

- in versions earlier than 0.9.3, geometry nodes were removed by default. One could change this behaviour using the "`--no-node-removal`" option.

Using Edges' maximum Speed Definitions in km/h

Some people do not like to use speed definitions in m/s. If you want to define the speeds allowed on your edges in km/h instead, you should pass the following option to NETCONVERT:

--speed-in-kmh *Optional (pregiven), type:bool, default: false*

This option may be used while importing the following formats:

- FastLane/CELL
- ARCView-databases
- XML-descriptions

Examples:

- `<SUMO_DIST>\data\examples\netbuild\cross_notypes_kmh` in comparison to `<SUMO_DIST>\data\examples\netbuild\cross_notypes`
- `<SUMO_DIST>\data\examples\netbuild\cross_usingtypes_kmh` in comparison to `<SUMO_DIST>\data\examples\netbuild\cross_usingtypes`

Recent changes:

- `--speed-in-kmh` was named `--speed-in-km` in versions earlier than 0.9.2

Importing Networks without Traffic Light Logics

Some of the supported network formats - Visum, Vissim and Artemis, supply information about the logic of the traffic lights, other do not. Due to this, we have to compute the traffic lights by our own. Doing this, we do not only have to compute the plans, but of course also, on which junction traffic

lights are positioned. There are several options steering this procedure. At first, you have to tell NETCONVERT/NETGEN that you wish him to guess positions of traffic lights. This is done using the "`--guess-tls`"-option. Then, you have the possibility to describe the junctions at which you think a tls shall be placed using description of incoming and outgoing edges: "`--tls-guess.no-incoming-min`", "`--tls-guess.no-incoming-max`", "`--tls-guess.no-outgoing-min`" and "`--tls-guess.no-outgoing-max`" constraint the building of a tls by the number of the lanes incoming/outgoing edges have. All these four options require an int as parameter. Furthermore, you may constraint the junctions by giving the minimum/maximum of allowed speed on edges that participate: "`--tls-guess.min-incoming-speed`", "`--tls-guess.max-incoming-speed`", "`--tls-guess.min-outgoing-speed`", and "`--tls-guess.max-outgoing-speed`".

Caution

No, we do not have a validated set of these option's settings, yet.

You may also set junctions as tls-controlled using "`--explicite-tls`" or as uncontrolled using "`--explicite-no-tls`". Both options assume to get a list of node names divided by ';' as parameter. The behaviour when a node is in both lists is undefined.

If you want to know where traffic lights have been inserted and your network is too large to evaluate this by hand, you can force NETCONVERT to write a list of POIs where each POI is placed on a tls-controlled junction. This allows you to tae a look at all the positions tls have been inserted at. The option for doing this is `--tls-poi-output <FILENAME>` where `<FILENAME>` is the filename to write the POIs into.

Normally, only one traffic lights logic (phases definition) is computed per a traffic lights controlled junction, but the algorithm we use is able to compute several logics. To force the computation of all possible logics, use "`--all-logics`". Remind, that all logics will be written to the network file and that we have no tools for further procesing of these logics.

During the computation of tls-logics among other information we have to guess the duration of the phases. The options "`--traffic-light-green`" and "`--traffic-light-yellow`" allow you to give the durations of green and yellow lights. Both options assume the duration in s as an int as parameter. The duration of having red is dependant to the number of other phases and their green and yellow phase durations. The green phase length has a default of 20s, yellow lights are - if no value is set for this option - computed using the "`--min-decel`" - value described below.

One has to remind one thing: dead times are necessary to avoid collisions of vehicles which do not manage to break as they are too near to the traffic light when it switches to red. This time may be computed, and is, but depends on the maximum deceleration possibility of the vehicles used. As this parameter is not known to the network builder at all - the vehicle types are supported to the simulation only - the option "`--min-decel`" (or `-D` for short) is used to set the minimum deceleration of vehicles. The default is 3.0 in m/s^2.

There is no possibility to compute or estimate green light districts, yet. You have only the options to shift the computed phases by half of their duration or by a quarter of their duration. The options for this are: "`--tl-logics.half-offset`" and "`--tl-logics.quarter-offset`". Both options assume to get a list of node names divided by ';' as parameter. The behaviour when a node is in both lists or if the node is not meant to be controlled by a tls is undefined.

Specific options:

`--guess-tls`	Forces NETCONVERT/NETGEN to guess whether a junction is controlled by a tls or not. *Optional, type:bool, default: false*

`--tls-guess.max-incoming-speed <SPEED>`	Sets the minimum and the maximum of the velocity a junction's incoming edges may have if the junction shall be tls-controlled in m/s. *Optional (pregiven), type:float, defaults: --tls-guess.min-incoming-speed: 40/3.6, --tls-guess.min-incoming-speed: 69/3.6*
`--tls-guess.min-outgoing-speed <SPEED>, --tls-guess.max-outgoing-speed <SPEED>`	Sets the minimum and the maximum of the velocity a junction's outgoing edges may have if the junction shall be tls-controlled in m/s. *Optional (pregiven), type:float, defaults: --tls-guess.min-outgoing-speed: 40/3.6, --tls-guess.max-outgoing-speed: 69/3.6*
`--tls-guess.no-incoming-min <LANE_NUMBER>, --tls-guess.no-incoming-max <LANE_NUMBER>`	Sets the minimum and the maximum number of a junction's incoming edges to allow the junction to be controlled by a tls. *Optional (pregiven), type:int, defaults: --tls-guess.no-incoming-min: 2, --tls-guess.no-incoming-max: 5*
`--tls-guess.no-outgoing-min <LANE_NUMBER>, --tls-guess.no-outgoing-max <LANE_NUMBER>`	Sets the minimum and the maximum number of a junction's outgoing edges to allow the junction to be controlled by a tls. *Optional (pregiven), type:int, defaults: --tls-guess.no-outgoing-min: 1, --tls-guess.no-outgoing-max: 5*
`--explicite-tls <JUNCTION <JUNCTION_ID>[;_ID>]*`	Informs the network builder that the given junctions shall be treated as being controlled by traffic lights, even if they do not match the tls-guess rules. *Optional, type:list of strings, default: none*
`--explicite-no-tls <JUNCTION <JUNCTION_ID>[;_ID>]*`	Informs the network builder that the given junctions shall be treated as NOT being controlled by traffic lights, even if they match the tls-guess rules. *Optional, type:list of strings, default: none*
`--all-logics`	Computes and saves all possible cliques and phases for a traffic light instead of the fastest one only. *Optional, type:bool, default: false*

Caution

This process may be very time consuming.

`( --min-decel | -D ) <FLOAT>`	he minimum deceleration value for vehicles in m/s^2. *Optional (pregiven), type:float, default: 3*
`--traffic-light-green <DURATION>`	The duration of green lights. *Optional, type:int, default: none (20s internally)*
`--traffic-light-yellow <DURATION>`	The duration of yellow lights; overrides `--min-decel`. *Optional, type:int, default: none*
`--tls-poi-output <FILENAME>`	If given, NETCONVERT will write position of nodes equipped with tls into `<FILENAME>`. *Optional, type:filename, default: none*

These options may be used while importing the following formats:

- FastLane/CELL
- ARCVIEW-data base files
- XML-descriptions
- Elmar-files
- VISUM-networks

Examples: none yet

Guessing On- and Off-Ramps

Most of the imported network descriptions do not have information about highway on- and off-ramps. You can force NETCONVERT to guess where on- and off-ramps shall be build. To enable this, use the option "`--guess-ramps`". The algorithm assumes that an on-ramp shall be build on highway junctions with one incoming and one outgoing highway edge and one incoming minor edge and that an off-ramp shall be build on highway junctions with one incoming and one outgoing highway edge and one outgoing minor edge. You can constrain what a highway is by giving its minimum speed of this edge using "`--ramp-guess.min-highway-speed`" and what a minor edge is by giving its maximum speed using "`--ramp-guess.max-ramp-speed`". Both options assume a float parameter being the speed. Furthermore, "`--ramp-guess.ramp-length`" tells NETCONVERT how long the added ramp shall be in meters.

Note

Normally, we keep `--ramp-guess.ramp-length` unset and let the geometry computation do the rest.

Specific options:

`--guess-ramps`	Forces NETCONVERT/NETGEN to guess whether a junction is controlled by a tls or not. *Optional, type:bool, default: false*
`--ramp-guess.max-ramp-speed` *<SPEED>*	Defines the maximum speed an edge may have in order to be a ramp in m/s. The default of -1 tells NETCONVERT that all edges may be potential ramps. *Optional (pregiven), type:float, default: -1*
`--ramp-guess.min-highway-speed` *<SPEED>*	Defines the minimum speed an edge may have in order to be a highway in m/s. *Optional (pregiven), type:float, default: 80/3.6*
`--ramp-guess.ramp-length` *<LENGTH>*	Sets the length of the ramps to build in meters. *Optional (pregiven), type:float, defaults: 100*

These options may be used in conjunction with all import formats.

Examples: none yet

Converting from Geocoordinates

Most professional networks are not stored using cartesian, but geo-coordinates. Since version 0.9.4 NETCONVERT is able to deal with such positions. NETCONVERT uses herefor the projection library "PROJ.4". This is important for you as a user, because you may have to describe the original projection of your file and when doing this, you have to describe it using the options offered by PROJ.4. In order to enable the reprojection use the option `--use-projection`. You can then add parameter for the projection using `--proj` *<STRING>*. The default for the projection is "+proj=utm +zone=33 +ellps=bessel +units=m". That means, that NETCONVERT assumes the network is a part of the UTM zone 33 and is described using the Bessel ellipsoid. Please remark, that when giving own description, you should embed it into "" for passing all the arguments to PROJ.4.

Specific options:

`--use-projection`	Enables reprojection of the network's coordinates. *Optional, type:bool, default: false*

`--proj <STRING>`	Defines projection. *Optional (pregiven), type:string, default: "+proj=utm +zone=33 +ellps=bessel +units=m"*

These options may be used in conjunction with the following import formats:

- Elmar & Elmar2
- ArcView
- Tiger
- Visum
- XML

Caution

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

Examples: none yet

Adding inner-junction Traffic

If you already know SUMO or if you have taken a look at some of the examples you may have noticed that vehicles "jump" over a junction instead of driving over them. This behaviour was quite appropriate for simulating large scenarios as in these cases the simulation error could be neglected (at least we have neglected it). Since version 0.9.5 SUMO is capable to simulate traffic over the junctions in a way you know it from reality. Because inserting inner lanes into a network dramatically increases the network's size, junction-internal lanes are not build by default. You can allow their building using the option `--add-internal-links` (or `-I` for short).

Note

Please note that you also have to enable the usage of internal lanes within the simulation.

Specific Options:

(--add-internal-links | -I)	Enables building of junction-internal lanes. *Optional, type:bool, default: false*

Examples:

Meanwhile all of the examples included in the distribution are build with this option set.

Recent changes:

- Inner-junction traffic was revalidated for version 0.9.5

Constraining the Input

NETCONVERT offers you some possibillities to constrain the read edges what is quite needful if one has a large street network but only wants to simulate a part of it or only the major roads. The first possibility to constrain the input is to name all the edges you want to keep. You can either do this on the command line/within your configuration directly using `--keep-edges <EDGE_ID>[;<EDGE_ID>]+` where each `<EDGE_ID>` represents the id of an edge you want to keep or you can save this list into a file where each id is stored in a seperate line and then let NETCONVERT read this file using `--keep-edges.input-file <FILENAME>`. In the case you are joining edges using `--remove-geometry` (see "Removing Geometry Nodes"), you may also be interested in the option `--keep-edges.postload` which forces NETCONVERT to join the edges first and remove the unwished afterwards.

It is also possible to constrain the imported edges by giving a minimum velocity that is allowed on an edge in order to include this edge into the generated network. Use `--edges-min-speed <MIN_SPEED>` for this where `<MIN_SPEED>` is the minimum velocity an edge must allow in order to be included in the output in m/s.

Specific options:

`--keep-edges <EDGE_ID>[;<EDGE_ID>]+`	Forces NETCONVERT to remove all edges not within the given list from the network. *Optional, type:list of edge ids, default: none*
`--keep-edges.input-file <FILENAME>`	Forces NETCONVERT to read the list of edge ids to keep from <FILENAME>. *Optional, type:filename, default: none*
`--keep-edges.postload`	Forces NETCONVERT to read all edges first and remove the unwished after joining. *Optional (pregiven), type:bool, default: false*
`--edges-min-speed <MIN_SPEED>`	Forces NETCONVERT to remove all edges from the input which allow a velocity below the given. *Optional, type:float, default: none*

Examples: none yet

Recent changes:

- The edge file generated using `plain-output` contains the information about the shape, the lane spread and the basic type if differing from the defaults since version 0.9.5.

Additional Output

NETCONVERT and NETGEN allow to generate additional output files beside writing the network file. We will present the possibilities in the following subchapters.

Plain Network Output

Parsed node and edge definitions may be saved into a XML-files which have the same formats as the ones used for importing XML-networks (as described in "Nodes Descriptions" and "Edges Descriptions"). This shall ease processing of networks read from other formats than XML. The option `--plain-output <FILENAME_PREFIX>` forces NETCONVERT and NETGEN to generate a file named "`<FILENAME_PREFIX>.nod.xml`" which contains the previously imported nodes and a file named "`<FILENAME_PREFIX>.edg.xml`" which contains the previously imported edges. The edge file will contain the list of previously read edges and each edge will have the information about the edge's id, the allowed velocity, the number of lanes, and the from/to - nodes stored. Geometry information is stored only if the imported edge has a shape, meaning that it is not only a straight connection between the from/to-nodes. The lane spread type and the basic edge type are only saved if differing from defaults ("right" and "normal", respectively). Additionally, if one of the lanes prohibits/allows vehicle classes, this information is saved, too (see also "Defining allowed Vehicle Types").

Specific options:

`--plain-output <PLAIN_OUTPUT_PREFIX>`	Writes the files `<PLAIN_OUTPUT_PREFIX>.nod.xml` and `<PLAIN_OUTPUT_PREFIX>.edg.xml` that contain the descriptions about the nodes and the edges a network is made of. These files may be reread into the netgener using the XML-import capabilities. *Optional, type:filename-prefix, default: none*

Examples: none yet

Recent changes:

- The edge file generated using `--plain-output` contains the information about the shape, the lane spread and the basic type if differing from the defaults since version 0.9.5.

Information about Geometry Removal

The option `--map-output` (or `-M` for short) generates a file which contains the information about which edges have been joined (see chapter "Removing Geometry Nodes [#user_chp04-further_options-turnarounds]").

The format is a little bit strange and should be reworked in the next time. At the begin of each line of the generated file, you will find the id of an edge from the generated network. Then, divided by tabs, you will find the list of edge ids together with the corresponding edges' lengths, the edge consists of. The id is divided from the length by a ':'. This means if that an edge that was joined from the edges 'edge1', 'edge2', 'edge3', each having the length 10, 20, and 30m, respectively, it would appear in the file encoded as following:

```
edge1<TAB>edge1:10<TAB>edge2:20<TAB>edge3:30
```

If the edge was not build by joining other edges, the list of edge ids/length will have only one value, of course:

```
edge<TAB>edge:100
```

Specific options:

`( --map-output | -M )` *`<MAP_OUTPUT_FILE>`*	Writes the file *`<MAP_OUTPUT_FILE>`* which holds the list of edges that were joined to a single edge for each edge. *Optional, type:filename, default: none*

Examples: none yet

Node Geometries Dump and Printing Node Positions

The option `--node-geometry-dump` is meant to be used when debugging the geometry computation. It generates a list of points of interest as readable by guisim (see chapter "Additional Geometry Files") on the positions that were used to compute the imported nodes' geometries. The option "`--print-node-positions`" forces NETCONVERT and NETGEN to print the positions of the imported/build nodes on the command line.

Specific options:

`--node-geometry-dump` *`<DUMP_FILENAME>`*	Writes the positions that were used during geometry computation into *`<DUMP_FILENAME>`*. *Optional, type:filename, default: none*
`--print-node-positions`	Prints the node positions during building. *Optional (pregiven), type:bool, default: false*

Examples: none yet

Automatic Network Generation

NETGEN allows builds abstract networks. Three types of networks can be built. All of them are very easy, but may be used as examples. You are also greatly invited to extend the concepts. Available types are: grid-networks, spider-networks and random-networks. You always have to supply the name of the network to generate using `--output <FILENAME>` or `-o <FILENAME>` for short and the type of network you want to create. So, exactly one of the following switches must be supported: `--grid-net`, `--spider-net` or `--random-net`.

While the type-dependent options are described within the next chapters, all types share some command line options. As all networks may possess junctions, you are able to set the default type of junctions to build using the `--default-junction-type`-option (or `-j` for short). The following junction types are allowed in accordance to the junction types currently known by the simulation: `priority`, `traffic_light`, `actuated`, `agentbased`.

Caution

`traffic_light` will be mapped to "static" within the generated network.

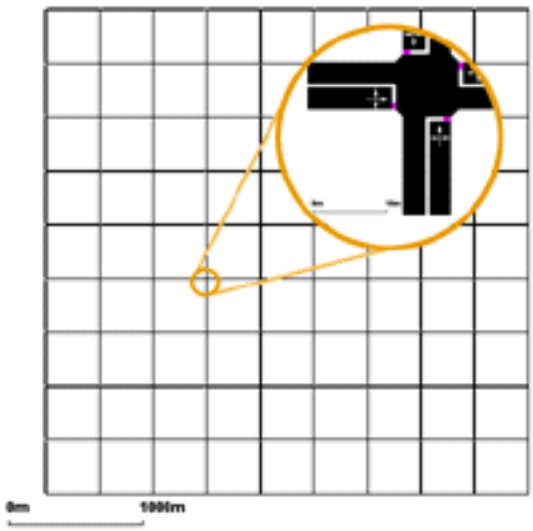
Further, you can specify the default street type by using the same options as in the netconvert-application.

Grid-like Networks

You are able to describe how many junctions in x- and in y-direction you want to be build and how far from each other they should be. The parameter for the number of junctions are `--grid-x-number` and `--grid-y-number`, the ones for the distance between the junctions `--grid-x-length` and `--grid-y-length`. If you want to build networks which have the same values for both axes, use `--grid-number` and `--grid-length`. The lengths are given in meters.

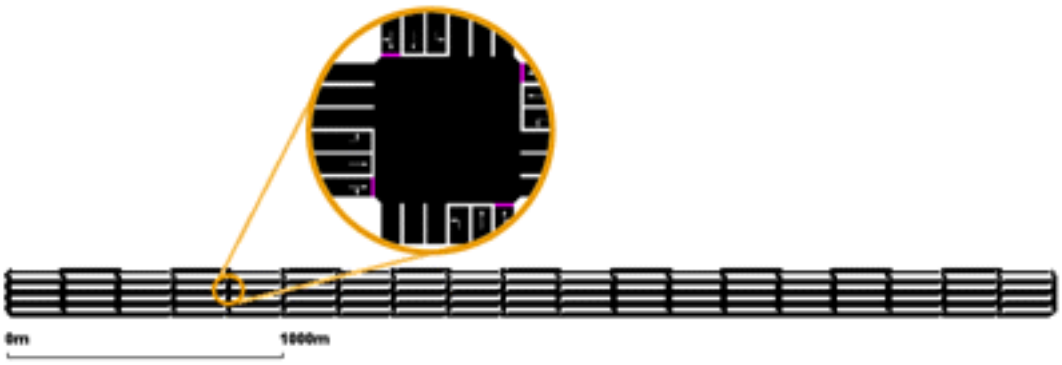
An example usage for building could be:

Figure 4.8. netgen --grid-net --grid-number=10 --grid-length=400 --output-file=MySUMOFile.net.xml



Another one:

Figure 4.9. netgen --grid-net --grid-x-number=20 --grid-y-number=5 --grid-y-length=40 --grid-x-length=200 --output-file=MySUMOFile.net.xml



Spider-net-like Networks

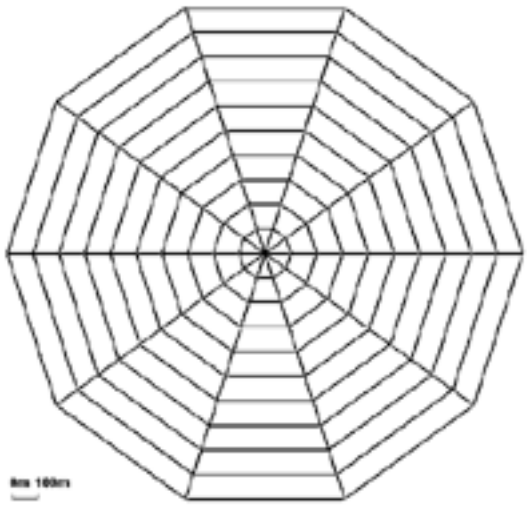
Spider-net networks are defined by the number of axes dividing them (parameter `--spider-arm-number` or `--arms`), the number of the circles they are made of (`--spider-circle-number` or `--circles`) and the distance between the circles (`--spider-space-rad` or `--radius`).

Caution

As the number of edges within the middle of the spider net may be quite large, it is often not possible to build a traffic light junction here. Due to this, this junction is always a right-of-way-junction.

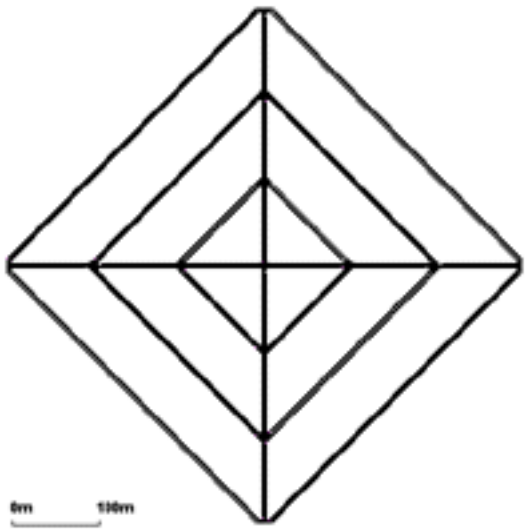
Two examples of usage:

Figure 4.10. netgen --spider-net --spider-arm-number=10 --spider-circle-number=10 --spider-space-rad=100 --output-file=MySUMOFile.net.xml



and:

Figure 4.11. netgen --spider-net --spider-arm-number=4 --spider-circle-number=3 --spider-space-rad=100 --output-file=MySUMOFile.net.xml



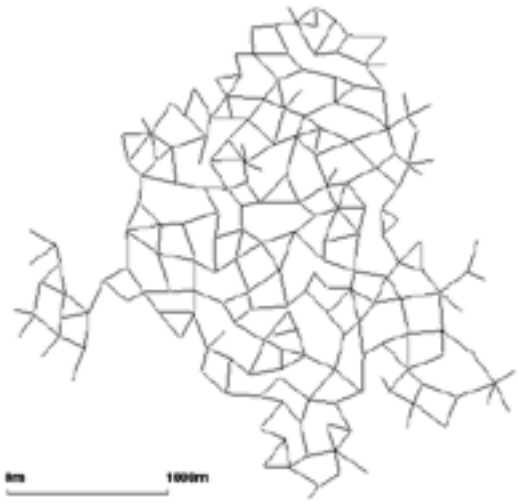
Random Networks

The random network generator does just what his name says, it builds random networks... Several settings may be changed:

- `--rand-max-distance <FLOAT>`: the maximum edge length
- `--rand-min-distance <FLOAT>`: the minimum edge length
- `--rand-min-angle <FLOAT>`: the minimum angle between two edges
- `--rand-num-tries <FLOAT>`:
- `--rand-connectivity <FLOAT>`:
- `--rand-neighbor-dist1 <FLOAT>`:
- `--rand-neighbor-dist2 <FLOAT>`:
- `--rand-neighbor-dist3 <FLOAT>`:
- `--rand-neighbor-dist4 <FLOAT>`:
- `--rand-neighbor-dist5 <FLOAT>`:
- `--rand-neighbor-dist6 <FLOAT>`:

An example:

Figure 4.12. netgen --random-net -o MySUMOFile.net.xml --rand-iterations=200 --abs-rand



Closing Thoughts (so far)

NETGEN 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 and write the list of imported edges/nodes using the `--plain-output` option
3. Build the network from the list of edges/nodes (normally setting the options `--guess-ramps` to true)
4. Load the network into GUISIM, 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

The following list contains recent changes in the naming or meaning of options. It has been started during the work on version 0.9.2, so earlier changes are not contained, herein. The changes list contains the version where the change occured or will occure. The changes should be valid for the next stable release.

- Changes in version 0.9.2
  - `--xml-type-files` was named `--type-file` in versions earlier than 0.9.2; *Reason:* the naming does not fit into the naming scheme of other XML-input files.
  - versions earlier than 0.9.2 allow to use a filename for each of the .dbf and the .shp file using "`--arcview-dbf`" and "`--arcview-shp`"; *Reason:* This was abondonned, because of the need to use .shx-files, also.
  - `--speed-in-kmh` was named `--speed-in-km` in versions earlier than 0.9.2; *Reason*: false physical measure

- Changes in version 0.9.3
  - in versions earlier than 0.9.3, turnarounds were not added per default. Instead the option "`--append-turnarounds`" has forced NETCONVERT to add them; *Reason:* turnarounds semm to be wished for the wider set of applications. Only sometimes it is unwished to add them.
  - in versions earlier than 0.9.3, geometry nodes were removed by default. One could change this behaviour using the "`--no-node-removal`" option; *Reason:* the same as for "`--append-turnarounds`"
- Changes in version 0.9.4
  - The `function`-tag for edges was added for version 0.9.4
- Changes in version 0.9.5
  - The `function`-tag within edges was revalidated for version 0.9.5
  - The `function`-tag for edge types was added for version 0.9.5
  - The edge file generated using `plain-output` contains the information about the shape, the lane spread and the basic type if differing from the defaults since version 0.9.5.
  - Inner-junction traffic was revalidated for version 0.9.5
  - The possibility to define which vehicle classes are allowed on a lane was added in version 0.9.5
- Changes in version 0.9.6
  - Described our current procedure of importing networks

Missing

There are some further options which were not yet described. Use at own risc.

```
--x-offset-to-apply, --y-offset-to-apply, --rotation-to-apply

--keep-unregulated,           --keep-unregulated.nodes,           -
-keep-unregulated.district-nodes

--guess-obscure-ramps,          --obscure-ramps.add-ramp,          -
-obscure-ramps.min-highway-speed
```

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 SUMO:

- 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-explicite-flows]").

- 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-explicite-trips]").

- by hand

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

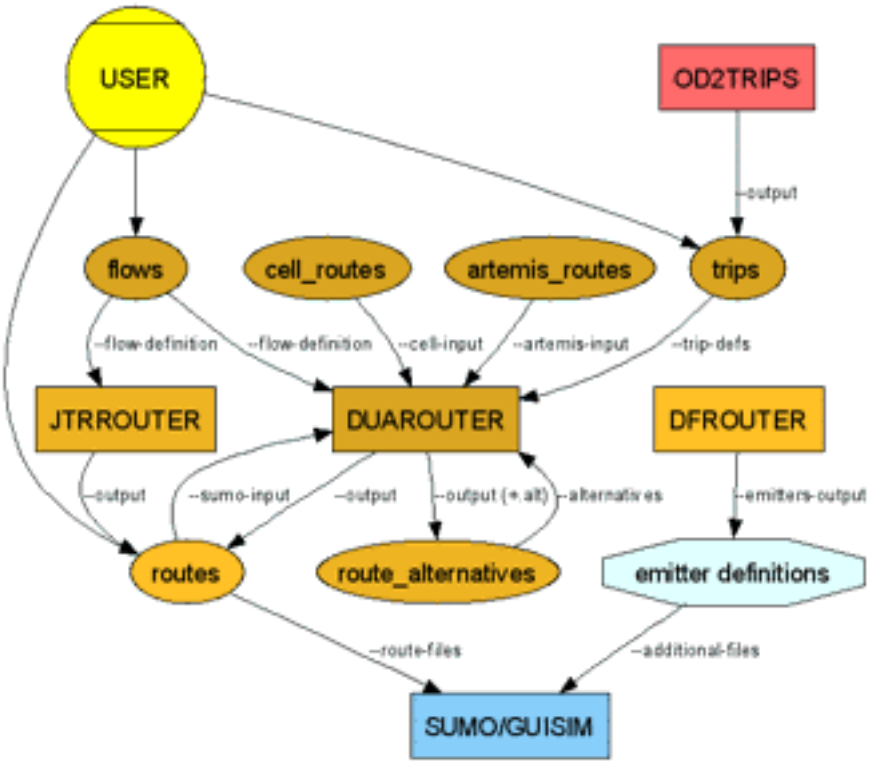
- using random routes

  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 SUMO-package contains four applications for processing routes. DUAROUTER is responsible for importing routes from other simulation packages and for computing routes using the shortest-path algorithm by Dijkstra. JTRROUTER may be used if you want to model traffic statistically, using flows and turning percentages at junctions. OD2TRIPS helps you to convert OD-matrices (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 (!!! Netzwerk fehlt)



Common, mandatory Values

Independent to what you are doing, you always have to supply the network using the `--net-file` (or `--net` or `-n` for short) option when working with either DFROUTER, DUAROUTER, JTRROUTER, 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 or JTRROUTER, 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 SUMO consists of three parts: a vehicle type which describes the vehicle's physical properties, a route the vehicle shall take, and the vehicle itself. Both routes and vehicle types can be shared by several vehicles. 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" length="5" maxspeed="7
</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)
- `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="7

   <vehicle id="0" type="type1" depart="0" color="1,0,0">
      <route>beg middle end rend</route>
   </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 and make SUMO know that it is shared by more than one vehicle using the `multi_ref` attribute. The vehicles using it 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="7

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

   <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
- `multi_ref`: If set to 'x' or 't' then the route is assumed to be shared by several vehicles and not removed until the simulation has ended
- `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 self-written 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.

- *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`.

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`. Otherwise, it is unlimited. !!! test !!!

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*

Examples:

Almost all networks within the `<SUMO_DIST>/data/` - folder. Additionally some examples that deal with trips may be found in `<SUMO_DIST>/data/examples/router`.

- `<SUMO_DIST>/data/examples/router/trips2routes/` shows the basic usage of trips; This example is quiet trivial - 100 same vehicles are emitted
- `<SUMO_DIST>/data/examples/router/trips2routes_repetition/` does exactly the same, but not by defining each of the 100 vehicles, but letting one vehicle be duplicated using `period` and `repno`

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"/>
```

and

```
<interval begin="0" end="3600">
   <flow id="0" from="edge0" to="edge1" 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 !!! test!!!!
- `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 `--flow-defs`) is used to make them known by the router:

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

Specific options:

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

Examples:

- *<SUMO_DIST>*`/data/examples/router/flows2routes/` shows the basic usage of flows; This example generates 100 vehicles just like *<SUMO_DIST>*`/data/examples/router/flows2routes/` but it uses flow definitions instead of trips for this.
- *<SUMO_DIST>*`/data/examples/router/flows2routes_100s_interval/` is almost the same, but vehicles are departing over a time of 100s.
- *<SUMO_DIST>*`/data/examples/router/flows2routes_200s_interval/` is almost the same, but vehicles are departing over a time of 200s.
- *<SUMO_DIST>*`/data/examples/router/flows2routes_100s_interval_ext/` shows the second possibility of defining intervals in flow-defs.

Recent changes:

- There was a bug on using flow in prior versions; the end time step was also used making the behaviour not as good predictable. This has been now changed so that the vehicles are emitted in steps starting at *<begin>* and ending at *<end>*-1.

Reason: The prior behaviour was not correct

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 ro DUAROUTER:

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

or the same for the JTRROUTER:

```
jtrrouter --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-per-second`). 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

There is also another problem with random routes: Often, the route starts in the opposite direction of the destination - on the wrong side of the road. This yields in a large amount of vehicles that want to turn back using the first possibility. This is of course also not common in the real world, especially not in areas as the one shown above. To avoid this, we remove the first and the last edge from the route per default when generating random routes. !!! revalidate this!!!

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*

Using the Junction Turning Ratio - Router

The JTRROUTER is a routing applications which uses flows and turning percentages at junctions as input. The following parameter must be supplied: the network to route the vehicles through, the description of the turning ratios for the junctions (defaults may be used for this, too), and the descriptions of the flows.

A call may look like this:

```
jtrrouter --flow-defs=<FLOW_DEFS> --turns=<TURN_DEFINITIONS> --net=<SUMO_NET> \
   --output-file=MySUMORoutes.rou.xml -b <UINT> -e <UINT>
```

The definitions of the flow look like the ones for the DUAROUTER with just a single difference: as it is not known where the vehicle will leave the network as the route it uses is randomly computed, the destination parameter has no meaning for jtr-routing and so may be left off. A vehicle leaves the network as soon as it comes to a sink edge. As not all networks have sink edges set, one can support a list of edges to be declared as sinks using `--sinks <EDGE_ID>[;<EDGE_ID>]*`.

To describe the turn definitions, one has to build a further file. Within this file, for each interval and each edge the list of percentages to use a certain follower is to be given. An example:

```
<turn-defs>
   <interval begin="0" end="3600">
      <fromedge id="myEdge0">
         <toedge id="myEdge1" probability="0.2"/>
         <toedge id="myEdge2" probability="0.7"/>
         <toedge id="myEdge3" probability="0.1"/>
      </fromedge>

      ... any other edges ...

   </interval>

   ... some further intervals ...

</turn-defs>
```

The snippet defines that vehicles coming at the end of edge "myEdge0" within the time interval between 0s and 3600s will choose the edge "myEdge1" with a probability of 20%, "myEdge2" with a probability of 70% and "myEdge3" with a probability of 10%. The specification of such information is of course quite extensive and so the JTRROUTER is also able to read the turning ratios from a cvs file. The same information as above may be coded the following way:

```
begin;end;from;to;split
0;3600;myEdge0;myEdge1;0.2
0;3600;myEdge0;myEdge2;0.7
0;3600;myEdge0;myEdge3;0.1
```

Caution

Do not forget to give the order of the attributes in the first line of the file.

Another possibility to save time on preparing the description is to use default values. The parameter `--turn-defaults (-T) <TURN_DEFAULTS>` can be used to describe the default ratios that will be used for all junctions for all time steps. `<TURN_DEFAULTS>` is a list of doubles, separated by a ';'. To achieve the same behaviour as in the example above, use `--turn-defaults=20;70;10`. The values will be applied to an edge's following edges beginning at the right edge (20%) and ending at the leftmost edge (10%). As the number of possible followers changes for different edges, the values are resampled for edges which number of following edges differs from the number of given turning probability defaults. Given `--turn-defaults=20;70;10` a vehicle using an edge that has two followers would use the follower to the right with 55% probability, the one to the left with 45%.

As theoretically a route may get infinitely long when a vehicle is forced to take always the same direction, it is possible to limit the route's size using `max-edges-factor`. This factor, multiplied with the number of the used network's edges is the maximum number of edges a route may have. With the default of 2.0, a route may contain twice as many edges as the network has. Any route longer than this size will be marked as invalid. We assume that for each network this number has to be chosen again.

The following options are accepted by JTRROUTER:

`( --net-file | --net | -n ) <SUMO_NET>`	Uses the named network to route vehicles on. *Mandatory, type:filename, default: none*
`( --output-file | --output | -o ) <FILENAME>`	Set `<FILENAME>` as the filename to write computed routes into. Additionally a file named "`<FILENAME>`.alt" will be generated which contains the route alternatives. *Mandatory, type: filename, default: none*
`(--begin | -b ) <INT>`	The first time step for which routes shall be build. *Optional (pregiven), type: int, default: 0*
`(--end | -e ) <INT>`	The last time step (+1) for which routes shall be build. *Optional (pregiven), type: int, default: 86400*
`( --alternatives | -a ) <ALTERNATIVES_FILE>`	Forces JTRROUTER to use the previously generated `<ALTERNATIVES_FILE>` as input. *Optional, type:filename, default: none*

Recent changes:

- The attribute "`probability`" within turn definitions was named "`perc`" in versions lower than 0.9.4 The reason for this change is that "`probability`" is more common throughout the package and the values had to be in the range between 0 and 1, what is no percentage information at all.

Examples:

Several examples may be found in `<SUMO_DIST>/data/examples/jtrrouter/`.

Using OD2TRIPS

OD2TRIPS computes trips tables from O/D (origin/destination) matrices. OD2TRIPS assumes the matrix/matrices to be coded as amounts of vehicles that drive from one district to another within a certain time period. Because the generated trips must start and end at edges, OD2TRIPS requires a

mapping of districts to edges. During conversion of VISUM networks - the base of the OD2TRIPS development - districts stored in the VISUM input file are parsed and stored within the outputted SUMO network file. If you do not use VISUM as input, you must build a districts file by your own. The format is given in "Describing the Districts", one of the next subchapters. You have to pass the file containing the district definitions to OD2TRIPS using the `--net-file` (`--net` or `-n` for short) option.

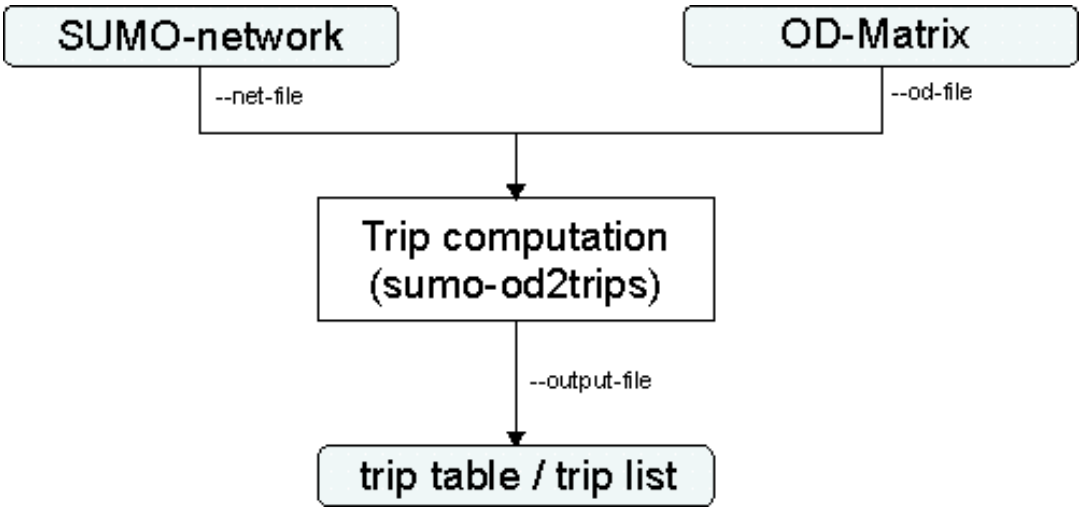
Because OD2TRIPS was used only to import data stored in VISUM/VISION formats, it assumes O/D to be stored in one of the VISUM formats. Not all VISUM formats are supported, by now only two, namely the "VMR" and the "OR" format. If you do not own matrices stored in these formats, you still have three possibilities: a) convert them into one of the supported formats, b) write your own reader for OD2TRIPS, or c) convert them into flow definitions and then give them to DUAROUTER (see Chapter "Using Flow Definitions [#user_chp05-explicite-flows]"). Both supported formats are described in "Describing the Matrix Cells", one of the next subchapters. You may either give a list of matrices to OD2TRIPS using the `--od-files` (`--od` or `-d` for short) option followed by the list of files separated using a ';', or - when importing routes from a Vissim simulation - using `--vissim <VISSIM_SIMULATION_FILE>` to read the matrices referred in this simulation file.

OD2TRIPS reads all matrices and generates trip definitions as described in "Using Trip Definitions". The generated trip definitions are numbered starting at zero. You can also add a prefix to the generated trip definition names using `--prefix <STRING>`. As usual, they are written to the output file named using the `--output-file` option (`--output` or `-o` for short). OD2TRIPS colors the trip definitions. To remove this feature, use `--no-color`. You can specify a vehicle type to be added to the trip definitions using `--vtype` followed by the type name. Please remark that vehicles will have no type unless not given in the O/D-matrices or defined using this option. This option overrides type names given in the O/D-matrices. The type itself will not be generated. Vehicles will be generated for the time period between `--begin` (`-b`) and `--end` (`-e`), having 0 and 86400 as default values, respectively. The meaning is the simulation step in seconds, as usual.

Because each O/D-matrix cell describes the amount of vehicles to emit within a certain time period, OD2TRIPS has to compute the vehicle's explicite departure times. Normally, this is done by using a random time within the time interval the O/D-matrix cell is defined for. It still is possible to emit a cell's vehicles with an uniform time between their emissions. Use the option `--spread.uniform` to enable this.

You can scale the amounts stored in the O/D-matrices using the `--scale` option which assumes a float as parameter. All read flows will be multiplied with this values, the default is 1. When importing O/D-matrices that cover a whole day, you maybe want to apply a curve which resembles the spread of the trip begins found in reality. Please read the subchapter "Splitting large matrices" on this.

Figure 5.3. Building trips from the OD-matrix



Specific options:

`( --net-file | --net | -n ) <DISTRICT_MAP>`	Tells OD2TRIPS to use the districts stored in `<DISTRICT_MAP>`. *Mandatory, type:filename, default: none*
`( --od-files | --od | -d ) <OD_FILE>[;<OD_FILE>]*`	Tells OD2TRIPS to use the given O/D matrices. *Optional, type:(list of) filename(s), default: none*
`--vissim <VISSIM_SIMULATION_FILE>`	Forces OD2TRIPS to read the list of matrices to process from the given Vissim file. *Optional, type: filename, default: none*
`( --output-file | --output | -o ) <FILENAME>`	Forces OD2TRIPS to write the generated vehicle trip definitions into `<FILENAME>`. *Mandatory, type: filename, default: none*
`(--begin | -b ) <INT>`	The first time step for which trip definitions shall be build. *Optional (pregiven), type: int, default: 0*
`(--end | -e ) <INT>`	The last time step (+1) for which trip definitions shall be build. *Optional (pregiven), type: int, default: 86400*
`( --scale | -s ) <FLOAT>`	A value with which read flow amount are multiplied. *Optional (pregiven), type: float, default: 1*
`--no-color`	Disables trip coloring. *Optional (pre-given), type: bool, default: false*
`--vtype <VEHICLE_TYPE_NAME>`	Adds a vehicle type to the trip definitions. *Optional, type: string, default: none*
`--prefix <VEHICLE_NAME_PREFIX>`	Adds a prefix to the vehicle names. *Optional, type: string, default: none*
`--spread.uniform`	Forces OD2TRIPS to spread the vehicle departure time uniformly for each cell. *Optional (pregiven), type: bool, default: false*
`--timeline.day-in-hours`	Says OD2TRIPS that the timeline is a list of amount per hour. *Optional (pregiven), type: bool, default: false*
`--timeline <TIMELINE_DEFINITION>`	If --timeline.day-in-hours is set, the string should contain 24 floating point values, divided by ';', each describing the probability with which a vehicle is emitted into the network dur-

ing the according hour. Otherwise, the string should contain a list of time/probability pairs, divided by ';', each describing the begin time and the probability for the interval defined by this and the next entry. Time and probability are divided by ','. *Optional, type: string, default: none*

Examples: None yet.

Recent changes:

- The whole application was rewritten for the 0.9.5. version. Most things still should work as well with older versions, but the options `--vtype`, `--prefix`, `--timeline`, and `--timeline.day-in-hours` were not available.
- The usage of `--od-files` and `--vissim` differs from prior versions.
- `--od-files` was named `--od-file` in versions prior to 0.9.5.

Describing the Districts

OD2TRIPS assumes the matrix/matrices to be coded as amount of vehicles that drive from one district to another within a certain time period. Because the generated trips must start and end at edges, OD2TRIPS requires a mapping of districts to edges. During conversion of VISUM networks which conversion was the base of the OD2TRIPS development, districts stored in the VISUM input file are parsed and stored within the network file. If you have to generate districts by your own, you must build a districts file by your own. Such a file looks as following:

```
<districts>
   <district id="<DISTRICT_ID>">
      <dsource id="<EDGE_ID>" weight="<PROBABILITY_TO_USE>"/>
      ... further source edges ...

      <dsink id="<EDGE_ID>" weight="<PROBABILITY_TO_USE>"/>
      ... further destination edges ...
   </district>

   ... further districts ...

</districts>
```

This means that a district is described by its `id`, being a simple name, and lists of source and destination edges. A district should have at least one source and one destination edge each described by its `id` and use probability called `weight` herein. These edges are used to emit and remove vehicles into/from the network respectively. The probability sums of each the source and the destination lists are normalized after loading.

Describing the Matrix Cells

To understand how an O/D-matrix is stored, we should remind the meanings of the values stored herein. Each matrix describes a certain time period. The indices within the matrix are names of the origin/destination districts (normally they are equivalent, both lists are the same). The values stored within the matrix are amounts of vehicles driving from the according origin district to the according destination district within the described time period.

The formats used by ptv are described in the VISUM-documentation more detailed. All start with a line where the type of the O/D-matrix is given, appended to a '$'. The first following character tells in which format the table is stored. Then, further characters follow which describe which values are supplied additionally within the matrix. For further information we ask you to consult the documentation supported by ptv. Herein, only the supported variants are described.

The V-format stores the O/D matrix by giving the number of districts first and then naming them. After this, for each of the named districts, a list of vehicle amounts that leave this district is given, sorted by the destination district names as given in the district name list. An example may look like this:

```
$VMR
* vehicle type
4
* From-Time  To-Time
7.00 8.00
* Factor
1.00
*
* some
* additional
* comments
* District number
3
* names:
        1          2          3
*
* District 1 Sum = 6
        1          2          3
* District 1 Sum = 15
        4          5          6
* District 1 Sum = 24
        7          8          9
```

The 'M' in the type name indicates that a vehicle type is used, the "R" that the values shall be rounded randomly. The second information is not processed by OD2TRIPS what means that you can parse both V-, VR-, VMR, and VM-matrices. Please remark that both the names list and the lists containing the amounts are written in a way that no more than 10 fields are stored in the same line. Each of the entries they contain seems to be left-alogned to a boundary of 11 characters (possibly 10 for the name and one space character). Both constraints are not mandatory for the importer used in OD2TRIPS.

The O-format instead simply lists each origin and each destination together with the amount in one line (please remark that we currently ignore the string after the ';' that occures after the type identifier "$OR" in the first line):

```
$OR;D2
* From-Time  To-Time
7.00 8.00
* Factor
1.00
* some
* additional
* comments
        1          1       1.00
        1          2       2.00
        1          3       3.00
        2          1       4.00
        2          2       5.00
        2          3       6.00
        3          1       7.00
        3          2       8.00
        3          3       9.00
```

Splitting large matrices

Since version 0.9.5 OD2TRIPS allows splitting matrices defining a long tie period into parts that dure less long. There are two ways of defining the amounts the matrix shall be split into. The first possibility is to use the option `--timeline` directly. In this case, it should be followed by a list of times and probabilities, separated by ';'. Each time and probability field is made up of two values, an

integer time being the simulation time in seconds and a floating point number describing the probability. These two values are separated using a ','. At least two values must be supplied making the definition of a timeline in this case being decribeable by the following BNF-formula:

<TIME>,<AMOUNT>[;<TIME>,<AMOUNT>]+

In this case, the matrix will be split into (fields-1) parts and each part will have the amount described by the integral within the field.

The second case is rather common in transportation science. It allows to split the matrix into 24 subparts - this means the number of dields is fixed to 24 - allowing to spread an O/D-matrix over a day describing it by hours. To use this, give additionally the option `--timeline.day-in-hours` to OD2TRIPS. It the assumes the values from the `--timeline` - option being a list of 24 floats, divided by ';', each describing the probability of emitting a vehicle within the according hour.

In both cases, the probabilities are automatically normed.

Importing Routes from other Simulations

Importing Artemis-routes

Although originally ARTEMIS-routes are stored within an OD-matrix, the import within SUMO is done using the DUAROUTER program.

This is basically done this way:

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

Remind that you have to specify the begin and the end of the simulation. This is quite inconsequent, as our mechanisms would allow import of repeatable routes, too. This is just not yet implemented.

Known problems:

- Only the a0-Flow is being imported
- Only one time-period is imported

Importing FastLane-routes

One of the main concepts of FastLane is the Dynamic User Assignment by Christian Gawron. As this mechanism is based on single vehicle routes, those are given for a FastLane simulation. The router module has only to convert routes from the FastLane- into the SUMO-representation. Do this using the following command:

```
duarouter --cell=<CELL_FILE_NAME(NO_EXTENSION)> --net=<SUMO_NET> \
   --output-file=MySUMORoutes.rou.xml -b <UINT> -e <UINT>
```

If you are familiar with FastLane, you may have noticed that you have only to supply one file name while FastLane uses two files to describe the routes: one "*.driver" and one "*.rinfo" file. As both files have the same name and only the extension differs, you have to give the name without the extension. So if your files have the names "`foo.driver`" and "`foo.rinfo`", write `--cell=foo`.

FastLane stores routes within a very large file what makes the process quite slow. If you want to speed it up for the future, you can tell SUMO-ROUTER to save some temporary files using `--save-cell-rindex`. A further file named "*<CELL_FILE_NAME>*.rindex" will be generated. It speeds up the processing by a factor of ten or so.

As FastLane is portable, you may encounter files generated on a Windows-machine. Those files give some strange warnings, something that should not happen if you supply the right net. To avoid them, use the `--intel-cell` switch to tell DUAROUTER it has to turn the byte order.

You can also decide whether the best, not the last route a driver took shall be used: set the switch `--no-last-cell` for this.

Importing VISSIM und VISUM-routes

See "Using OD2TRIPS" for information about how to import O/D-matrices in ptv format.

Dynamic User Assignment and Alternative Routes

!!! Weights fehlen!!!

Dynamic Assignment is used to find out which routes the simulated driver really would take. One could assume that everyone is traing to use the fastest route. But what is the fastest route? Look at the network shown in the next picture. It is abvious, that the fastest route is the one in the middle of the network, even when considering the tls at the end of this road. But as soon as we insert many vehicles into the network, all trying to use this route, the edge located at the center soon gets full and vehicles need much more time than estimated to pass it. In fact, they need longer to pass this edge than if they would try to use one of the outer connections.

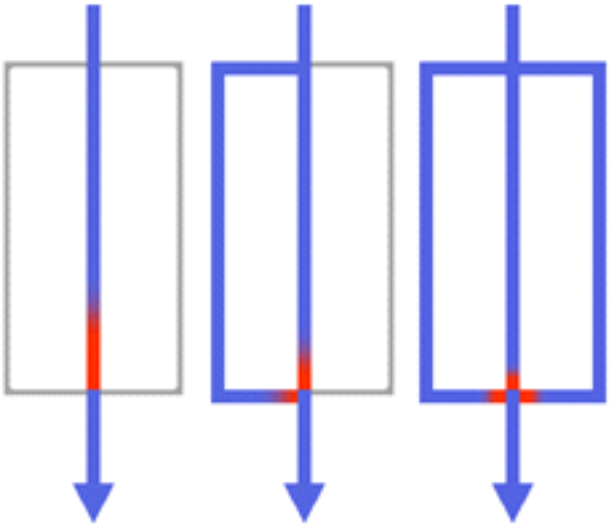
Figure 5.4. Example DUA-network (from "<SUMO_DIST>/data/examples/dua/dua3s*")

Within Christian Gawron's approach (see [Gawron1998_1] [http://sumo.sourceforge.net/docs/bibliography.shtml#Gawron1998_1]), which we use, each driver has a list of routes. At the beginning, we build a single route for each driver being the shortest route within an empty network - not knowing how many vehicles will use this route. You may wonder that two files are built. The file you have named as output file and a further file having the additional extension ".alt". Within this second file route alternatives - the set of all routes the driver knows - are stored.

After having generated the routes, we let the simulation run forcing it to generate edge-based dumps (see chapter 6, "Aggregated Lane/Edge States (Edge/Lane-Dumps) [http://sumo.sourceforge.net/docs/gen/user_chp06.shtml#user_chp06-output-edgelanedump]"). The router is In fact, edge-based dumps are the better choice, because the route is not able to use lane-based dumps at all. Now, we have the information about the state of the network, meaning the real speeds that were driven within it, when all vehicles are using the same route. Now we can pass this information to a second call to DUAROUTER and now - instead of the previous input file - we give

the route alternatives file as input using the option `--alternatives` (or `-a` for short). Now, DUAROUTER is capable to compute the new shortest routes, based on the real speeds within our simulation. In order to avoid that again all vehicles will use the currently shortest route only some of the drivers will get the new shortest route. Again, two files are generated, a route file which contains the current routes and a new route alternatives file. The entries within the new alternatives file stay the same for all those drivers who keep their old route. Those who got a new route assigned, will get this route stored additionally to their previous route within the file. Whether a route is replaced by a new, better route is depending on how much better the new route is and how fast a driver shall forget his old one. There are two parameters determining this: `--gBeta` and `--gA`. Please see (see [Gawron1998_1] [http://sumo.sourceforge.net/docs/bibliography.shtml#Gawron1998_1]) for further information.

Figure 5.5. Sketch showing the effects of Christian Gawron dua-approach on route distribution within the network; blue color indicates that an edge is used within the step, red shows jams



Still, as the picture above shows, there may be other faster routes when again running the simulation, so in order to get a balanced assignment, you have to iterate this process several time.

Here, you may again find the procedure to generate a dynamic user assignment:

1. Generate the network (see chapter 4 [http://sumo.sourceforge.net/docs/gen/user_chp04.shtml])
2. Import your routes (see chapter 5, "Building own Routes from Scratch [#user_chp05-own_routes]" or "Importing Routes from other Simulations [#user_chp05-import_routes]"), generating a routes file and an alternatives file
3. Simulate using the network and the routes (see chapter 6 [http://sumo.sourceforge.net/docs/gen/user_chp06.shtml]) forcing SUMO to write edge-based dumps (see chapter 6, "Aggregated Lane/Edge States (Edge/Lane-Dumps) [http://sumo.sourceforge.net/docs/gen/user_chp06.shtml#user_chp06-output-edgelanedump]")
4. Compute new routes and alternatives using the previously generated alternatives and the edge-based dumps generated within the last step
5. Continue with step 3 until the DUA is completed

How do we know when the dua is completed? In fact. we don't know any possibility to determine

this, yet. Within very large networks we are running mostly about 20 iterations, but this is just a hint. One may say, that the dua is finished if there is no further change in the aggregated vehicles trip duration. DUA is definitely still a research topic. If you are interested in more details than you maybe should take a look at "More On... Dynamic User Assignment [http://sumo.sourceforge.net/docs/gen/sumo_moreon_dua.shtml]".

Specific options:

`( --alternatives | -a ) <ALTERNATIVES_FILE>`	Forces DUAROUTER to use the previously generated `<ALTERNATIVES_FILE>` as input. *Optional, type:filename, default: none*
`--gBeta <FLOAT>`	Defines Christian Gawron's beta. *Optional (pregiven), type:float, default: 0.3*
`--gA <FLOAT>`	Defines Christian Gawron's alpha. *Optional (pregiven), type:float, defaults: 0.05*

Examples:

- `<SUMO_DIST>/data/examples/dua/due2s_singlestep/` contains configuration files for the first three steps of a dynamic user assignment in a network with two possible ways.
- `<SUMO_DIST>/data/examples/dua/due2s_automatic/` uses the same input as `<SUMO_DIST>/data/examples/dua/due2s_singlestep/`, but the iterations are done using a script.
- `<SUMO_DIST>/data/examples/dua/due3s_singlestep/` contains configuration files for the first three steps of a dynamic user assignment in a network with three possible ways.
- `<SUMO_DIST>/data/examples/dua/due3s_automatic/` uses the same input as `<SUMO_DIST>/data/examples/dua/due3s_singlestep/`, but the iterations are done using a script.

Automatic Iteration using 'dua-iterate.pl'

"**dua-iterate.pl**" helps you to perform the computation of a dynamic user assignment. The script needs at least two parameter: the path to the folder where your SUMO-binaries are located and the number of iteration steps to perform. When started with these two options, the script computes the given number of dua-steps. As input two files have to be located in the same folder as the script: a SUMO-network named "`net.net.xml`" and a set of trip definitions named "`trips.trips.xml`". if you want to start with another routes definition, you have to change this within the script's source code.

Within each iteration step, the script generates a configuration file for the DUAROUTER and starts DUAROUTER with this configuration file. Then, a configuration file for SUMO is built and SUMO ist started. Both configuration files are competely defined within the script itself. As default, for each time step, SUMO will generate three dump files with edge-dumps aggregated over 150, 300, and 900s, an emissions and a trip information output. The names of these outputs are numbered over the iteration steps. If you want to change the outputs, you also have to take a look into the script, but you should not disable the edge-based dump for aggregation over 900s, because this is read by the DUAROUTER in the next iteration steps in order to compute the DUA.

When started with a third parameter, the second parameter given will be interpreted as the start iteration step and the third as the end iteration step.

This useful script is located in `<SUMO_DIST>/tools/dua_tools/`.

Synopsis:

```
dua-iterate.pl <PATH_TO_SUMO_BINARIES> [[<BEGIN_ITERATION_STEP>]] <END_ITERATIO
```

Tracking the progress online using "printProgress.pl"

printProgress.pl is a small script which, started in the folder the dua-iterate.pl - iteration takes place - shows how many vehicles were running at the end of the simulation. This helps to see how good the current assignment is (see also "More On... Dynamic User Assignment [http://sumo.sourceforge.net/docs/gen/sumo_moreon_dua.shtml]").

This script is located in `<SUMO_DIST>/tools/dua_tools/`.

Synopsis:

```
printProgress.pl
```

Recent changes:

- **printProgress.pl** is a part of the distribution since version 0.9.6.

Additional Weights

From version 0.8.0.1 on, SUMO is capable to load additional weights for edges. The idea behind this is to virtually increase the costs for an edge in order to make it less attractive to pass. Additional weights are used by DUAROUTER only and are supplied using the `--supplementary-weights <FILE>` (or `--add <FILE>` or `-S <FILE>`) - option. A file containing additional weights looks like this:

```
<supplementary-weights>
   <interval begin="60" end="119">
      <edge id="1fi" absolute="42" factor="2.5" summand="-21"/>
      <edge id="2o" factor="13"/>

      ... further weights ...

   </interval>
   <interval begin="120" end="179">
      <edge id="1fi" absolute="48" factor="2.5"/>
      <edge id="2o" summand="7"/>

      ... further weights ...

   </interval>

   ... further intervals ...

</supplementary-weights>
```

When additional weights are supplied, the DUA-Router first looks whether an additional, absolute value has been provide for the currently watched edge (value of the "`absolute`"-attribute for the current edge and time). If so, this value will be used. If not, either the edge's loaded weight (if available for the current time step) or the default value (length/vallowed) will be changed first by applying the loaded factor (value of the "`factor`"-attribute for the current edge and time) and than by the loaded addend (value of the "`summand`"-attribute for the current edge and time).

You can also use GUISIM to generate an additional weights file using the interface which appears when choosing "Edit->Edit Additional Weights...".

Recent Changes:

- The format has changed in version 0.9.6

Using Detectors and DFROUTER

Since version 0.9.5, the SUMO-package contains a further routing module named DFROUTER. The idea behind this router is that nowadays, most highways are well equipped with induct loops, measuring each of the highways' entering and leaving flows. Given this information one may assume that the flows on the highway are completely known. DFROUTER uses directly the information collected from induct loops to rebuild the vehicle amounts and routes. This is done in several steps, being mainly:

1. Computing (and optionally saving) the detector types in the means that each induction is set to be a source detector, a sink detector or an in-between detector
2. Computing (and optionally saving) the routes between the detectors
3. Computing the flow amounts between the detectors
4. Saving the flow amounts and further control structures

In the following we will describe the steps more deeply, giving the command line options that steer the process.

Caution

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

Examples: none yet

Computing Detector Types

The idea behind the DFROUTER assumes that a network is completely covered by detectors, meaning that all off- and on-ramps have an induction loop placed on them. Such an information whether an induction loop is a pure source or sink or whether it is placed between such is but not given initially. It must be computed. To do this, the DFROUTER needs the underlying network as well as a list of detector definitions where each describes the position of an induction loop. The network, being a previously build SUMO-network, is supplied to the DFROUTER as usually using the `( --net-file | --net | -n ) <SUMO_NET_FILE>` - option, the list of induction loops using `--detector-files` (or `--detectors` or `-d` for short) `<DETECTOR_FILE>[;<DETECTOR_FILE>]+`. A detector file should look as following:

```
<detectors>
   <detector_definition id="<DETECTOR_ID>" lane="<LANE_ID>" pos="<POS>"/>
... further detectors ...
</detectors>
```

This means that each detector is initially described using its id, a lane it is placed on, and a position on the lane. To be exact:

- `id`: A string holding the id of the detector
- `lane`: The id of the lane the detector lies on. Must be a lane within the network.
- `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).

Given a network and the list of detectors, DFROUTER assigns types to detectors *(TBD: add a reference)* and saves the so extended list into a file if the option `--detectors-output <DETECTOR_OUTPUT_FILE>` is given. This list looks like the input described above except that an aditional attribute is given for each detector, "`type`", which may have one of the following val-

ues: "`source`", "`sink`", "`between`", and "`discarded`". You can also generate a list of points of interests (POIs) which can be read by GUISIM where each POI represents a detector and is colored by the detector type: green for source detectors, red for sink detectors, blue for in-between detectors, and black for discarded detectors. To force DFROUTER to do this, use `--detectors-poi-output <POI_FILENAME>`.

When wished, if for example other parameters chage, the extended `<DETECTOR_OUTPUT_FILE>` can be fed back again into DFROUTER instead of the previous `<DETECTOR_FILE>`. In this case the detector types do not have to be computed again. To force DFROUTER to recompute the types though, use `--revalidate-detectors`.

Specific options:

`( --detector-files | --detectors | -d ) <DETECTOR_FILE>[;<DETECTOR_FILE>]*`	The list of descriptions of detectors to use. *Mandatory, type:filename, default: none*
`--detectors-output <DETECTOR_OUTPUT_FILE>`	The file to write extended detector descriptions into. *Optional, type:filename, default: none*
`--detectors-poi-output <POI_FILENAME>`	Writes a list of points of interest into <> where each poi represents a detector and is colored by the detector type. *Optional, type:filename, default: none*
`--revalidate-detectors`	Forces DFROUTER to recompute the detector types even if they were given in `<DETECTOR_FILE>`. *Optional (pregiven), type:bool, default: false*

Computing Routes

Now that we do know where vehicles enter and where they leave the network, we may compute routes for each of the pairs. The DFROUTER is told to build and save routes using `--routes-output <ROUTE_OUTPUT_FILE>` where `<ROUTE_OUTPUT_FILE>` is the name of the file the computed routes shall be written to. The generated file only contains routes, no vehicle type definitions and no vehicles. In later runs, you can omit the routes computation by supplying previously generated routes using `--routes-input` (or `-r`) `<ROUTE_FILE>`. Again, as during the computation of the detector types, you can force DFROUTER to recompute the routes even if suppling them using `--revalidate-routes`.

Normally, only routes starting at source detectors and ending at sink detectors are computed. Using the option `--routes-for-all` you can force DFROUTER to also build routes that start at in-between detectors. The option `--all-end-follower` will make the routes not end at the edge the source detector is placed on, but on all edges that follow this edge *(TBD: recheck this)*. `--keep-unfound-ends` will also keep those routes where a sink detector could not be found for what may be the case if the network is not completely covered with induction loops.

Specific options:

`--routes-output <ROUTE_OUTPUT_FILE>`	Names the file to save computed routes into. *Optional, type:filename, default: none*
`( --routes-input | -r) <ROUTE_FILE>`	Uses routes from `<ROUTE_FILE>`. Omits route computation unless `--revalidate-routes` is set. *Optional, type:filename, default: none*
`--revalidate-routes`	Forces DFROUTER to recompute routes even if some where supplied. *Optional (pregiven), type:bool, default: false*
`--routes-for-all`	Forces DFROUTER also to save routes starting at in-between detectors. *Optional (pregiven), type:bool, default: false*
`--all-end-follower`	Forces DFROUTER to end the routes at edges beyond the

sink detectors. *Optional (pregiven), type:bool, default: false*

`--keep-unfound-ends` Forces DFROUTER to keep routes which last detector was an in-between detector. *Optional (pregiven), type:bool, default: false*

Computing Flows

The next step is to use the computed routes and flow amounts from the real-world detectors to compute flows across the modelled network. The flows are given to DFROUTER using `--detector-flow-files` (or `--detflows`, `-f` for short) *`<DETECTOR_FLOWS>[;<DETECTOR_FLOWS>]+`*. They are assumed to be stored in CSV-format using ';' as dividing character. The file should look as following:

```
Detector;Time;qPKW;qLKW;vPKW;vLKW
myDet1;0;10;2;100;80
... further entries ...
```

This means the first time has to name the entries (columns). Their order is not of importance, but at least the following columns must be included:

- Detector: A string holding the id of the detector this line describes; should be one of the ids used in <DETECTOR_FILE>
- Time: The time period begin this entry describes
- qPKW: The number of passenger cars that drove over the detector within this time period
- qLKW: The number of transport vehicles that drove over the detector within this time period
- vPKW: The average speed of passenger cars that drove over the detector within this time period in km/h
- vLKW: The average speed of transport vehicles that drove over the detector within this time period in km/h

These are not quite the values to be found in induction loop output. We had to constrain the *`<DETECTOR_FLOWS>`* files this way because DFROUTER is meant to read very many of such definitions and to do this as fast as possible. Because even this scheme was too slow for some purposes, flow amounts can also be marked as more valid using the option `--fast-flows`. If set, DFROUTER assumes that there is no header line and that the values are separated using spaces. In this case, the columns must be in the order time, detector name, qPKW, qLKW, vPKW, and vLKW.

Because in some cases one reads detector flow definitions starting at a certain time but wants his simulation begin at another, it is possible to add a time offset using `--time-offset` *`<TIME_OFFSET>`* which is subtracted from the read times. *`<TIME_OFFSET>`* is meant to be an int representing seconds.

(TBD: verify: how long is the time period, are speeds given in m/s, is qPKW/qLKW right, does qPKW/qLKW describe the amount within this period or within one hour)

Specific options:

`( --detector-flow-files | --detflows | -f)` *`<DETECTOR_FLOWS>[;<DETECTOR_FLOWS>]`* Describes the files to read detector flows from. *Optional, type:filename, default: none*

`--fast-flows` Says that the flows are stored in a more constrained, faster to read format *Optional (pregiven), type:bool, default: false*

`--time-offset` *`<TIME_OFFSET>`* Gives a time offset to be subtracted from the times read from detector flows. *Optional (pregiven), type:int, default: 0*

Recent changes

- Changes in version 0.9.6
  - vPKW and vLKW should now be supplied in km/h, not in m/s

Saving Flows and other Values

If flow definitions were supplied, we can let the DFROUTER save the computed vehicles together with their routes. Because vehicles will be emitted at the source detectors which are placed at certain positions of the networks' lanes, emitters (see "Emitter") are used to insert those vehicles into the network. You can force the DFROUTER to generate such emitters using `--emitters-output <EMITTER_OUTPUT_FILE>`. This file will contain emitter declarations for each of the source detectors. If no value is given, emitters will not be written. Accompanying, there will be emitter definitions written named "`emitter_<DETECTOR_ID>.def.xml`" where `<DETECTOR_ID>` is the id of the according source detector. These definitions are called within the `<EMITTER_OUTPUT_FILE>` and contain vehicles which depart the emitter in accordance to the read flows and have routes computed using the flows.

As some approaches use a speed limit to avoid open-end boundary problems *(TBD add a reference)*, the DFROUTER can generate a list of speed triggers (see "Variable Speed Signs (VSS)") placed on the positions of sink detectors. The name to save the declaration of these speed triggers into is given using the option `--speed-trigger-output <VSS_OUTPUT_FILE>`. The according speed trigger definitions will be written into files named "`vss_<DETECTOR_ID>.def.xml`" where `<DETECTOR_ID>` is the name of the according sink detector.

In order not to end vehicle routes on off-ramps, it is possible to place rerouters (see "Rerouter") at the positions of the sink detectors, too. Giving the option `--end-reroute-output <REROUTER_OUTPUT_FILE>` will generate a list of rerouter declarations. Please remark that in this case, no rerouter definitions are written, because the DFROUTER has no further information about possible routes beyond the area covered by the detectors.

It's quite nice to have the possibility to check whether the simulation does what one wants. To validate whether the same flows are found within the simulation as within the reality, the option `--validation-output <SUMO_DETECTORS_OUTPUT>` may be helpful. It generates a list of detector definitions (E1/induct loops, see "E1-Detectors (Induct Loops)") placed at the positions of sink and in-between detectors. Their output will be saved into files named "`validation_det_<DETECTOR_ID>.xml`" and should be easily comparable to the detector flows previously fed to the router. The option `--validation-output.add-sources` will let DFROUTER also build E1-detectors for source detectors which are place 1m behind the real-life detector's position.

(TBD: check whether everything is placed into the proper directory; check what's about PKWs/LKWs)

`--emitters-output <EMITTER_OUTPUT_FILE>`	Forces DFROUTER to write emitter declarations for source detectors into `<EMITTER_OUTPUT_FILE>`. Additionally, emitter definitions are written. *Optional, type:filename, default: none*
`--speed-trigger-output <VSS_OUTPUT_FILE>`	Forces DFROUTER to write speed trigger declarations for sink detectors into `<VSS_OUTPUT_FILE>`. Additionally, speed trigger definitions are written. *Optional, type:filename, default: none*
`--end-reroute-output <REROUTER_OUTPUT_FILE>`	Forces DFROUTER to write rerouter declarations for sink detectors into `<REROUTER_OUTPUT_FILE>`. *Optional, type:filename, default: none*
`--validation-output <SUMO_DETECTORS_OUTPUT>`	Forces DFROUTER to write validation detector definitions for sink and in-betweed detectors into `<SUMO_DETECTORS_OUTPUT>`. *Optional, type:filename,*

	default: none
`--validation-output.add-sources`	Forces DFROUTER to include source detectors in the validation detector output. *Optional (pregiven), type:bool, default: false*

Closing Thoughts (so far)

If you are want to simulate small scenarios only, with a few vehicles, you probably should try to work with routes where each vehicle is duplicated. This needs only a small amount of editing and lets you fill the simulation fast. This has been done for most of the examples within the example data. Trying to generate own OD-matrices is not to be adviced, because handling of such is not really well supported by the package. Using flows generated by hand may be a possibility, but for small scenarios, flows seem more clumsy than routes.

In some cases, flows at each junction are counted and how many vehicles are driving in which direction. This is exactly what JTRROUTER resembles.

Recent Changes

The following list contains recent changes in the naming or meaning of options. It has been started during the work on version 0.9.2, so earlier changes are not contained, herein. The changes list contains the version where the change occured or will occure. The changes should be valid for the next stable release.

- Changes in version 0.9.3
  - There was a bug on using flow in prior versions; the end time step was also used making the behaviour not as good predictable. This has been now changed so that the vehicles are emitted in steps starting at `<begin>` and ending at `<end>`-1.
  - Types given in trips were always replaced by "KRAUSS_DEFAULT"!!!. This has been patched. Now the type you specify will be used and only if no type information was given, "KRAUSS_DEFAULT" will be used.
- Changes in version 0.9.5
  - OD2TRIPS was rewritten for the 0.9.5. version. Most things still should work as well with older versions, but the options `--vtype`, `--prefix`, `--timeline`, and `--timeline.day-in-hours` were not available.
  - The usage of `--od-files` and `--vissim` within OD2TRIPS differs from prior versions.
  - `--od-files` was named `--od-file` in OD2TRIPS versions prior to 0.9.5.
- Changes in version 0.9.6
  - **printProgress.pl** is a part of the distribution since version 0.9.6.

Missing

`--lane-weights` is not tested and described

`--scheme` is not implemented and described

`--max-alternatives` is not tested and described

`--prune-random` is not tested and described

spread uniformly of DFROUTER is not described

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 SUMO - command line simulation. This command line tool does not generate any graphical output as the SUMO-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 or NETGEN.

- 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 SUMO 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 and furthermore possibilities to influence the simulation are necessary to make it more variable in use. Detectors, artifacts used to gain more or less processed results from the simulation and some further structures used are described in the next subchapters.

To supply the definitions of these structures to the simulation, we use an additional file normally and pass it to "SUMO" 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.

Detectors

The results are obtained from the simulation using simulated detectors. You will find detectors one knows from the real world such as induct loops, but also some virtual ones that allow gaining values one can work with more easily.

Basically, the main distinction between detectors SUMO offers is their dimension. The next list shows all available detector types, some of which are still under development. The type names "E*" have their origin in the German word "Erfassungsbereich" meaning "detection area".

- E1: Induct loops

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

- E2: Areal, lane-based detectors

  These detectors describe a part of a lane or alternatively a part of the network made up of consecutive lanes (a begin lane and his predecessors). The measured values are derived from the movements over the whole part of the network the detector is lying at.

- E23: Route-dependent Origin/Destination detectors

  ... not yet implemented ...

- E3: Multi-Origin/Multi-Destination detectors

  E3-detectors measure vehicles passing a set of entry and an according set of exit points. Each of these points is a position on a lane. Measured are values that may be derived from the movements of vehicles between any of the entry and any of the exit points.

- E41: Destination based detectors

  ... not yet implemented ...

- E42: Edgebased detectors

  ... not yet implemented ...

We will not replicate the exact computation of the detector's values. A document describing this should be found on our pages within the documentation [http://sumo.sourceforge.net/docs/documentation.html] part. The next subchapters hold the information about how to set detectors onto a network only.

To ease the usage and for backward compatibility, all detectors may be defined in two ways. The first one is by using the following notation: `<detector id="<ID>" type="<TYPE>" ...further attributes.../>`. Herein, the detector type is determined by the type-attribute which is "induct_loop" as default. The second possibility is: : `<XX-detector id="<ID>" ...further attributes.../>` where the tag name already defines the detector to build. Possible values for *XX* within the tag name and attributes needed to describe each detector are described in the following subchapters.

E1-Detectors (Induct Loops)

An induct loop is defined either this way:

```
<detector id="<ID>" type="(induct_loop|E1)" lane="<LANE_ID>" pos="<POSITION_ON_
   freq="<AGGREGATION_TIME>" [style="xml"] file="<OUTPUT_FILE>" [friendly_pos="
```

or this way:

```
<e1-detector id="<ID>" lane="<LANE_ID>" pos="<POSITION_ON_LANE>"
   freq="<AGGREGATION_TIME>" [style="xml"] file="<OUTPUT_FILE>" [friendly_pos="
```

The id is any string that let's you know which detector is meant. The type indicates that a induct loop shall be build, here. The attributes "`lane`" and "`pos`" describe on which lane and at which position on him the detector shall lay. As induct loop detectors may aggregate the values they collect, the freq-attribute describes this period. The style-parameter is obsolete by now as the earlier possibility to use either "xml" or "cvs"-output is now not supported, the values are stored in xml-files only. 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 warn-

ing.

Caution

The folder the output file shall be generated in must exist.

Let's review the attributes:

- `id`: A string holding the id of the detector
- `type`: Always "induct_loop" or "E1" for this type of detectors ("induct_loop" is the default value)
- `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.
- `style`: Obsolete/deprecated; Always "xml" by now
- `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 place 0.1 meters from the lane's end.

Recent changes:

- The attribute `friendly_pos` is available since version 0.9.4

E2-Detectors (Areal, lane-based Detectors)

An induct loop is defined the following way:

```
<detector id="<ID>" type="(areal|lane_based|E2)" lane="<LANE_ID>"
   pos="<POSITION_ON_LANE>" length="<DETECTOR_LENGTH>" freq="<AGGREGATION_TIME>
   [style="xml"] file="<OUTPUT_FILE>" [measures="<MEASURES>"] [time_treshold="<
   [speed_treshold="<FLOAT>"] [jam_treshold="<FLOAT>"] [keep_for="<FLOAT>"]/>
```

or:

```
<e2-detector id="<ID>" lane="<LANE_ID>" pos="<POSITION_ON_LANE>"
   length="<DETECTOR_LENGTH>" freq="<AGGREGATION_TIME>" [style="xml"]
   file="<OUTPUT_FILE>" [measures="<MEASURES>"] [time_treshold="<FLOAT>"]
   [speed_treshold="<FLOAT>"] [jam_treshold="<FLOAT>"] [keep_for="<FLOAT>"]/>
```

Most of the attributes have the same meaning as for induct loops. As an areal detector has a certain length, this length must be supplied as a further parameter. It may also be a negative number which lets the detector be extended upstream to the given beginning position. The type must be set to either "`areal`", "`lane_based`" or "`E2`" to let the simulation know what's desired to build. The optional parameter "cont" let's the detector continue over the current lane onto this lane's predecessors when the detector's length plus his position is larger than the place available on the lane. The attribute "`measures`" describes which values the detector shall compute. The optional values are described below.

Caution

The folder the output file shall be generated in must exist.

Caution

For detectors that span over more than a single edge, only the attribute QUEUE_LENGTH_AHEAD_OF_TRAFFIC_LIGHTS_IN_VEHICLES is defined all other may return strange values.

But there is also a further possibility to use E2-detectors. If you place them in front of a traffic light, you can use the traffic light to describe the intervals (aggregation) time instead of giving a fixed aggregation time. In this case, output will be generated every time the traffic light switches. To use this feature, simply replace the freq-attribute within the description of your detector by the id of the traffic light that should steer it (use the attribute "`tl`" to specify the id) :

```
<detector id="<ID>" type="[(areal|lane_based|E2)" lane="<LANE_ID>" \
   pos="<POSITION_ON_LANE>" length="<DETECTOR_LENGTH>" tl="<TL-ID>"
   freq="<AGGREGATION_TIME>" [style="xml"] file="<OUTPUT_FILE>"
   [measures="<MEASURES>"] [time_treshold="<FLOAT>"]
   [speed_treshold="<FLOAT>"] [jam_treshold="<FLOAT>"] [keep_for="<FLOAT>"]/>
```

or:

```
<e2-detector id="<ID>" lane="<LANE_ID>" pos="<POSITION_ON_LANE>"
   length="<DETECTOR_LENGTH>"  tl="<TL-ID>" freq="<AGGREGATION_TIME>" [style="x
   file="<OUTPUT_FILE>" [measures="<MEASURES>"] [time_treshold="<FLOAT>"]
   [speed_treshold="<FLOAT>"] [jam_treshold="<FLOAT>"] [keep_for="<FLOAT>"]/>
```

A further feature allows you to output values not for all switches of the traffic light the detector is attached to, but only when the light turns red for the assigned link (connection between the incoming and the outgoing lane). This should allow you to measure the maximum jam length in front of a red traffic light for this link. To switch on this feature, you have to add the name of the following lane: `to="<LANE_ID>"`. The incoming lane is already given by the "`lane`"-attribute.

E2-detectors may compute many different measures and the user has the possibility to describe which measures he actually wants to be generated. The "measures"-attribute must contain the measures divided by a ' ' (blank) in the case he does not want to compute all parameters. Computing all parameters is the default case but may also be set using '`measures="ALL"`' . The available measures are:

- DENSITY: The density on the detector in vehicles/hour averaged over the requested interval.
- MAX_JAM_LENGTH_IN_VEHICLES: Every timestep, the maximum number of consecutivly jamming vehicles is detected. These values are averaged over the requested interval.
- MAX_JAM_LENGTH_IN_METERS: Every timestep, the maximum length demand of consecutivly jamming vehicles in meters is detected. These values are averaged over the requested interval.
- JAM_LENGTH_SUM_IN_VEHICLES: Every timestep, the sum of the lengths of all jams on the detector is measured (in vehicles). These values are averaged over the requested interval.
- JAM_LENGTH_SUM_IN_METERS: Every timestep, the sum of the lengths of all jams on the detector is measured (in meters). These values are averaged over the requested interval.
- QUEUE_LENGTH_AHEAD_OF_TRAFFIC_LIGHTS_IN_VEHICLES: This detector uses a MAX_JAM_LENGTH_IN_VEHICLES one as a helper. Every timestep, the "maximum-jam-length" (in vehicles) from MAX_JAM_LENGTH_IN_VEHICLES will be compared to the maximum of "maximum-jam-lengths" that occured since the last reset. If the new value is larger, the maximum of "maximum-jam-lengths" is updated. Between two resets, this detector records a monoton growing set of "maximum-jam-lengths". These values are averaged over the requested interval. The reset is performed by a traffic light.
- QUEUE_LENGTH_AHEAD_OF_TRAFFIC_LIGHTS_IN_METERS: As QUEUE_LENGTH_AHEAD_OF_TRAFFIC_LIGHTS_IN_VEHICLES, but in meters, not in vehicles.
- N_VEHICLES: Every timestep, the number of vehicles that populate the detector is recorded. These values are averaged over the requested interval.
- OCCUPANCY_DEGREE: Every timestep the length of the vehicles populating the detector is summed up. We divide this length by the detectorlength to get a value out of [0,1]. These values are averaged over the requested interval.
- SPACE_MEAN_SPEED: Every timestep, the mean-speed of the vehicles on the detector is calcu-

lated. These values are averaged over the requested interval.
- CURRENT_HALTING_DURATION_SUM_PER_VEHICLE: Every timestep, the halting-time of the vehicles on the detector is summed up and then averaged over the number of vehicles. These values are averaged over the requested interval.
- N_STARTED_HALTS: A vehicle on the detector that just started halting, will report the time when this event took place to the detector. All events during the requested interval are summed up.
- HALTING_DURATION_SUM: A vehicle that starts moving after a halt will report it's halting-duration (in seconds) and the time when this event took place to the detector. The halting-durations of all events during the requested interval are summed up.
- HALTING_DURATION_MEAN: Every vehicle sums up it's halting-durations (in seconds) during it's stay on the detector. When a vehicle leaves the detector, it's halting-duration-sum is stored by the detector. These values are averaged over the requested interval.Only vehicles that moved through the entire detector contribute.
- APPROACHING_VEHICLES_STATES: This detector is a special kind of E2 detector. It doesn't return a single value but a container of vehicle states. Here, a vehicle state is a tuple consisting of the distance from the vehicle front to the detector end and the vehicle's speed. There is no averaging or summing up but current output is provided. This detector is intended for internal use, e.g. as input to traffic-light-controls.

Again, the explicit list of available attributes:

- `id`: A string holding the id of the detector
- `type`: Always "lane_based" or "E2" for this type of detectors ("induct_loop" is the default value)
- `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. See information about the same attribute within the detector loop description for further information.
- `length`: The length of the detector in meters. If the detector grows over the lane's end (begin in fact), it is either cut off at the lane's length if the "`cont`"-attribute is false or not given or is continued on the predeceding lanes in the case the "`cont`"-attribute is set to true.
- `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.
- `measures`: Should contain the list of measures to compute (see above) or "ALL" to compute all measures.

And the optional ones:

- `style`: Obsolete/deprecated; Always xml by now
- cont: Holds the information whether detectors longer than a lane shall be cut off or continued (set it to true for the second case) default: false (detector lies on one lane only).
- `time_treshold`: The time-based threshold that describes how much time has to pass until a vehicle is recognized as halting (in s, default: 1s).
- `speed_treshold`: The speed-based threshold that describes how slow a vehicle has to be to be recognized as halting (in m/s, default: 5/3.6m/s).
- `jam_treshold`: The minimum distance to the next standing vehicle in order to make this vehicle count as a participant to the jam (in m, default: 10m).
- `keep_for`: Information for how long the memory of the detector has to be (in s, default: 1800s).
- `measures`: Should contain the list of measures to compute (see above) or "ALL" to compute all measures (default: ALL).

E3-Detectors (Multi-Origin/Multi-Destination Detectors)

The descriptions of E3-detectors have to include the set of entry- and the set of exit-cross-sections. Due to this, it is not possible to use a single tag to specify a detector. Instead, the description consists of the following parts:

1. A beginning tag that describes some global attributes of the detector just as the descriptions of e1- and e2-detectors do. The format is either:

```
<detector id="<ID>" type="(multi_od|E3)" file="<OUTPUT_FILE>"
   freq="<AGGREGATION_TIME>" [measures="<MEASURES>"] [time_treshold="<FLOAT>"
   [speed_treshold="<FLOAT>"] [keep_for="<FLOAT>"]>
```

or:

```
<e3-detector id="<ID>" file="<OUTPUT_FILE>" freq="<AGGREGATION_TIME>"
   [measures="<MEASURES>"] [time_treshold="<FLOAT>"] [speed_treshold="<FLOAT>
   [keep_for="<FLOAT>"]>
```

As one can see, no information about the detector's position is stored herein. They are stored in embedded tags instead (2. and 3.)

2. A set of tags that describe the detector's entry points in the form:

   ```
   <det_entry lane="<LANE_ID>" pos="<POSITION_ON_LANE>"/>
   ```

3. A set of tags that describe the detector's exit points in the form:

   ```
   <det_exit lane="<LANE_ID>" pos="<POSITION_ON_LANE>"/>
   ```

4. A closing tag that must match the opening tag (1.):

   a) `</detector>`

   or

   b) `</e3-detector>`

The definition

```
<e3-detector id="e3_1" freq="300" file="./output/e3_1.xml">
   <det_entry lane="myEdge0_0" pos="0"/>
   <det_entry lane="myEdge0_1" pos="0"/>
   <det_exit lane="myEdge2_0" pos="0"/>
   <det_exit lane="myEdge2_1" pos="0"/>
</e3-detector>
```

will build an e3-detector starting at either lane 0 or 1 of the edge called "myEdge0" and end at the same lane of "myEdge2". All values will be computed as the default-value for measures is used and aggregated over a time of 300s. They will be written into the file "`e3_1.xml`" lying in the subfolder of the folder the configuration was read in/the program has been started within.

Most of the values have been discussed in the previous subchapters. The only thing that differs from e1/e2-detectors are the measures e3-detectors are able to compute. Possible values are:

- MEAN_TRAVELTIME: A vehicle that entered the detector through an entry-cross-section and leaves it through a leave-cross-section will store it's traveltime (in seconds) and the leaving-time into the detector. These traveltimes of the vehicles that left during the requested interval are averaged.
- MEAN_NUMBER_OF_HALTINGS_PER_VEHICLE: A vehicle that entered the detector through an entry-cross-section and leaves it through a leave-cross-section will store it's number of haltings and the leaving-time into the detector. The halting-values of the vehicles that left during the requested interval are averaged.
- NUMBER_OF_VEHICLES: A vehicle that entered the detector through an entry-cross-section and leaves it through a leave-cross-section will store it's leaving-time into the detector. The vehicles that left the detector during the requested interval will be summed up.

As for e2-detectors, you can use the value "ALL" for the attribute measures to compute all values.

This is also this attributes default value.

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>">
            <vehicle id="<VEHICLE_ID>" pos="<VEH_POSITION>" speed="<VEH_SPEED>"

            ... more vehicles if any on this lane ...

         </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.

Examples:

- *<SUMO_DIST>*`/data/examples/output_tests/cross3ltl_rawdump/` shows how the raw output is used. The output is written into the subfolder "`output`".

Recent changes:

- Please notice that this options has been earlier falsely named `--output` (`-o`)

Aggregated Lane/Edge States (Edge/Lane-Dumps)

This output is far more feasible than the network dump. There are two different types of these files, one is edge-based, the other lane-based. Both describe the situation on all of the network's edges/lanes in terms of traffic science by giving macroscopic values such as the mean vehicle speed, the mean density, etc.

In the following, it is described how both outputs are generated and which values they contain. Then, the meanings of the values are given as well as a description of intervals. At last, some additional possibilities to constraint the outputs are given.

Note

Please remark that "aggregated lane/edge states" are also called "meandata" or "edge/lane-dumps".

Note

Some people find the number of information within the lane/edge states quite minimalist. This is because this output is used as input for the DUAROUTER during the computation of a dynamic user assignment (see "Dynamic User Assignment and Alternative Routes") and due to this is meant to be fast. That's why it only contains values that are fast to compute.

Recent changes:

- The documentation has been updated to fit the real output when being rechecked for version 0.9.3
- The (even invalid) documentation of the file printed previously at the begin of the file was removed in version 0.9.3
- This documentation text was rewritten for version 0.9.5, because the previous text said that only those vehicles are regarded which have left the lane. The edge-dumps/lane-dumps contain instead the values of all vehicles that were on the edges/lanes within the interval.
- Furthermore, computation of the density and the occupancy has been debugged for version 0.9.5.

Edge-Based Network States

To force SUMO to generate edge-based state dumps, you have to use two command line options: a) `--dump-basename <PATH_AND_FILE_PREFIX>` describes where to write the dumps to and how the begin of the file name is, b) `--dump-intervals <INTERVAL>[;<INTERVAL>]*` describes over what time the values shall be collected and aggregated ("interval length"). Each aggregation interval is written into an own, single file. The file name is made up from the base name and the aggregation interval: `<FILENAME>` = `<PATH_AND_FILE_PREFIX>_<INTERVAL>.xml`. An example: `--dump-basename=./output/lanedump --dump-intervals=300;600` will build two files: `./output/lanedump_300.xml` and `./output/lanedump_600.xml`. For edge-based state dumps, the output file will look like the following:

```
<netstats>
   <interval begin="<INTERVAL_BEGIN>" end="<INTERVAL_END>">
      <edge id="<EDGE_ID>" traveltime="<MEAN_TRAVEL_TIME>" \
                 nSamples="<VEHICLE_NUMBER>" \
                 density="<MEAN_DENSITY>" occupancy="<MEAN_OCCUPANCY>" \
                 noStops="<NUMBER_OF_HALTS>" speed="<MEAN_SPEED>"/>

      ... more edges ...

   </interval>
```

```
   ... further intervals ...

</netstats>
```

Please remark, that in contrary to the example above, for each edge, all values are reported in one line.

Examples:

- `<SUMO_DIST>/data/examples/output_tests/cross3ltl_meandata_edges/` shows how to generate an edge-based aggregated state output. Herein, four outputs are written into the subfolder "`output`", one for each of the intervals 15s, 60s, 300s, and 900s.

Lane-Based Network States

Lane-dumps are generated analogous to edge-dumps using the options `--lanedump-basename <PATH_AND_FILE_PREFIX>` and `--lanedump-intervals <INTERVAL>[;<INTERVAL>]*`. The generated output looks like the following:

```
<netstats>
   <interval begin="<INTERVAL_BEGIN>" end="<INTERVAL_END>">
      <edge id="<EDGE_ID>">
         <lane id="<LANE_ID>" traveltime="<MEAN_TRAVEL_TIME>" \
                 nSamples="<VEHICLE_NUMBER>" \
                 density="<MEAN_DENSITY>" occupancy="<MEAN_OCCUPANCY>" \
                 noStops="<NUMBER_OF_HALTS>" speed="<MEAN_SPEED>"/>

         ... more lanes

      </edge>

      ... more edges ...

   </interval>

   ... further intervals ...

</netstats>
```

Please remark, that in contrary to the example above, for each edge, all values are reported in one line.

Examples:

- `<SUMO_DIST>/data/examples/output_tests/cross3ltl_meandata_lanes/` shows how to generate a lane-based aggregated state output. Herein, four outputs are written into the subfolder "`output`", one for each of the intervals 15s, 60s, 300s, and 900s.

Value Descriptions

Both the edge-dump and the lane-dump are computing the values the same way: every vehicle move - even those with v=0 - is recorded and saved during the interval. After the interval has passed, these values are written into the file after being normalised. In the case of the edge-dump the values are not only normalized by the number of the collected vehicle moves and the length of the lane, but also by the number of lanes of the edge.

The meanings of the written values are:

- `traveltime`: The mean travel time is computed from the collected vehicle velocities. If no vehicle has passed the edge, length/maximum_allowed_velocity is used; Measure: s
- `speed`: The mean of the collected vehicle velocities. If no vehicle has passed the edge, the max-

imum allowed velocity is used; Measure: m/s
- `density`: The mean vehicle density on the edge in veh/km.
- `occupancy`: The occupancy of the edge (in range of 0-1)
- `noStops`: The number of recognized stops
- `nSamples`: The number of collected samples - the number of vehicle steps done on this edge during the interval.

The interval end is the interval begin + aggregation time - 1, meaning that values were collected within these steps. If the simulation ends before the last interval is over, the interval will be prunned.

Constraining the State Outputs

If you need only information about the network states during certain time periods, you may constraint generation of the dumps using the options `--dump-begins <TIME>[;<TIME>]+` and `--dump-ends <TIME>[;<TIME>]+`. When at least one combination is given, dumps will be written only if an according begin/end-pair exists for the current time. This means, only those intervals will be saved for which dump-begins[x]<=INTERVAL_END and dump-ends[x]>=INTERVAL_BEGIN.

Of course, all dumps will cover the complete simulation if the options `--dump-begins` and `--dump-ends` are not set.

Examples:

- *`<SUMO_DIST>/`*`data/examples/output_tests/cross3ltl_meandata_constrained/` shows how to generate a restrained state output. Herein, eight outputs are written into the subfolder "`output`", four edge- and four lane-based, and for each of the intervals 15s, 60s, 300s, and 900s.

Net-Wide Vehicle Emission States & Travel Times

This output contains the simulation-wide number of vehicles that are loaded, emitted, running, waiting to be emitted, have reached their destination and how long they needed to finish the route. The last value is normalised over all vehicles that have reached their destination so far. The information containing all those values is computed for each time step and the output file looks like following:

```
<emissions>
   <emission-state time="<SIMULATION_TIME>"
               loaded="<LOADED_VEHICLE_NUMBER>" \
               emitted="<EMITTED_VEHICLE_NUMBER>" \
               running="<RUNNING_VEHICLE_NUMBER>" \
               waiting="<NUMBER_OF_VEHICLES_WAITING_FOR_EMISSION>" \
               ended="<ENDED_VEHICLE_NUMBER>" \
               meanWaitingTime="<MEAN_WAITING_TIME>" \
               meanTravelTime="<MEAN_TRAVEL_TIME>"/>

   ... further time steps ...

</emissions>
```

Please remark, that in contrary to the example above, for each time step, all those values are reported in one line.

The meanings of the written values are:

- `time`: the time step the entry describes
- `loaded`: the number of vehicles that were loaded into the simulation until the reported time step
- `emitted`: the number of vehicles already emitted until the reported time step

- `running`: the number of vehicles that were running within the reported time step
- `ended`: the number of vehicles that have reached their destination within the reported and the prior time step
- `meanWaitingTime`: the mean time all vehicles up to and within the reported time step had to wait for being emitted;-1 if no vehicle has been emitted, yet
- `meanTravelTime`: the mean travel time of all vehicles that have left the simulation within the previous and the reported time;-1 if no vehicle has been removed from the simulation, yet

You can force the simulation to generate this output using `--emissions-output <FILENAME>` or `--emissions <FILENAME>`.

Examples:

- `<SUMO_DIST>/data/examples/output_tests/cross3ltl_emissions/` shows how the emissions output is used. The output is written into the subfolder "`output`".

Recent changes:

- In versions prior to 0.9.3, the attribute "`time`" was named "`id`"

Vehicle-Oriented Trip Information

This output contains the information about each vehicle's departure time, the time the vehicle wanted to start at (which may be lower than the real departure time) and the time the vehicle has arrived. Such an information is generated for each vehicle as soon as the vehicle has arrived its destination and is removed from the network. The format is as following:

```
<tripinfos>
   <tripinfo vehicle_id="<VEHICLE_ID>" start="<DEPARTURE_TIME>" \
             wished="<WISHED_DEPARTURE_TIME>" \
             end="<ARRIVAL_TIME>" \
             duration="<TRAVEL_TIME>" \
             waited="<WAITING_TIME>" \
             reroutes="<REROUTE_NUMBER>" \
             devices="<DEVICE_LIST>" \
             vtype="<VEHICLE_TYPE>"/>

   ... information about further vehicles ...

</tripinfos>
```

Please remark, that in contrary to the example above, for each time step, all those values are reported in one line. An entry is written each time a vehicle has arrived at his destination. In prior to this, the written values would not be known.

The meanings of the written values are:

- `vehicle_id`: the id of the vehicle this entry describes
- `start`: The real departure time (the time the vehicle was emitted into the network)
- `wished`: The departure time the vehicles wanted to be emitted into the network
- `end`: The time the vehicle was removed from the simulation (due to arriving at the route end)
- `duration`: The time the vehicle needed to accomplish the route (in s)

- `waited`: The time the vehicle has waited until being emitted
- `reroutes`: The number the vehicle has been rerouted
- `devices`: List of devices the vehicle had. Each device is separated from the others by a ';'. If a vehicle may contain several devices (CPhones, f.e.) the number will be printed (example: cphones=1)
- `vtype`: The type of the vehicle

The simulation is forced to generate this output using: `--tripinfo-output <FILENAME>` or `--tripinfo <FILENAME>`.

Examples:

- `<SUMO_DIST>/data/examples/output_tests/cross3ltl_tripinfo/` shows how the tripinfo output is used. The output is written into the subfolder "`output`".

Recent changes:

- In versions prior to 0.9.3, the attribute "`vehicle_id`" was named "`id`"
- The documentation has been updated before releasing version 0.9.3
- The number of reroutes attribute was added for version 0.9.6
- The devices list attribute was added for version 0.9.6
- The vehicle type attribute was added for version 0.9.6

Vehicle Routes

The vehicle routes output contains information about which route a vehicle took and if his route was replaced at any time by a new one, each of the previous routes together with the edge at the time their replacement took place is reported. Furthermore, the vehicle emission and ending time is stored herein.

The generated file look like this:

```
<vehicleroutes>
   <vehicle id="<VEHICLE_ID>" emitedAt="<EMISSION_TIME>" endedAt="<ARRIVAL_TIME
      <route replacedOnEdge="<EDGE_ID>" replacedAtTime="<TIME>"><PREVIOUS_ROUTE

      ... further replaced routes ...

      <route><LAST_ROUTE></route>
   </vehicle>

   ... information about further vehicles ...

</tripinfos>
```

The values have the following meanings:

- `id`: the id of the vehicle this entry describes
- `emitedAt`: The time the vehicle was emitted into the network)
- `endedAt`: The time the vehicle was removed from the simulation (due to arriving at the route

end)

- `replacedOnEdge`: The edge the vehicle was on when the described route was replaced
- `replacedAtTime`: The time step of this replacement
- *<PREVIOUS_ROUTE>*: The replaced route
- *<LAST_ROUTE>*: The final vehicle route

Both the previous and the final routes are complete, that means that they contain all the edges the vehicle was meant to pass as long as the route was not replaced, yet. The information `replacedOnEdge` and `replacedAtTime` are available only for routes which were replaced.

In normal conditions, when all vehicles use predefined routes, the output does not contain any information that could not be retrieved from the routes and the tripinfo output. But as soon as you reroute your vehicles within the simulation, f.e. using rerouters (see "Rerouter"), it will contain new information.

The simulation is forced to generate this output using: `--vehroutes-output <FILENAME>` or `--vehroutes <FILENAME>`.

Examples:

- *<SUMO_DIST>*`/data/examples/output_tests/cross3ltl_vehroutes/` shows how the vehicle routes output is used. The output is written into the subfolder "`output`". This is just a basic example that the output is generated. Better take a look at *<SUMO_DIST>*`/data/examples/extended/rerouter/`.
- *<SUMO_DIST>*`/data/examples/extended/rerouter/` uses rerouters to change the vehicles' routes. A vehicle routes output into the `output`-subfolder.

Recent changes:

- This output was finally finished and validated for version 0.9.3

Output coupled to Traffic Lights

SUMO offers some possibilities to save states of traffic lights during the simulation, a feature mainly used to evaluate adaptive traffic light algorithms. We will now describe these outputs.

TLS States

To enable writing tls state information you have to add the following definition into one of your additional files: `<timed_event type="SaveTLSStates" source="<TLS_ID>" dest="<OUTPUT_FILE>"/>`. The attributes have herein the following meanings:

- `type`: type of the event trigger; always "`SaveTLSStates`" herein
- `source`: The id of the traffic light which state shall be written
- `dest`: The file to save the state into

The output looks like this:

```
<tls-states>
   <tlsstate time="<SIM_STEP>" id="<TLS_ID>" subid="<TLS_SUBID>"><STATE></tlsst
   ... further states ...
```

```
</tls-states>
```

The state is saved in each simulation second. The state itself is coded as a list of the characters 'G', 'Y', and 'R', standing for "green", "yellow", and "red", respectively. Each character describes a link controlled by the traffic light. Only the state of the current program is saved (see also "Adding new Programs"). The attributes have the following meaning:

- `time`: The simulation time this entry was generated for
- `id`: The id of the tls that is responsible for the link
- `subid`: The sub-id of the tls that is (currently) responsible for the link

Missing:

- An easy mapping from positions within the state to links.

Recent changes:

- Changes in version 0.9.5
  - This output is available since a long time, still several issues may made him unworking before version 0.9.5
- Changes in version 0.9.6
  - Since version 0.9.6 only the state of the current program is saved

TLS Switches

This output contains information about the green light phases of links (lane-to-lane connections). Each green light phase is describes by its begin, end and duration. An entry is written into the file as soon a green phase of a link ends. To enable writing tls switch information you have to add the following definition into one of your additional files: `<timed_event type="SaveTLSSwitchTimes" source="<TLS_ID>" dest="<OUTPUT_FILE>"/>`. The attributes have herein the following meanings:

- `type`: type of the event trigger; always "`SaveTLSSwitches`" herein
- `source`: The id of the traffic light which state shall be written
- `dest`: The file to save the state into

The output looks like this:

```
<tls-switches>
   <tlsswitch tls="<JUNCTION_ID>" subid="<JUNCTION_SUB_ID>" \
      fromLane="<LINKS_SOURCE_LANE>" toLane="<LINK_DESTINATION_LANE>" \
      begin="<BEGIN_OF_GREEN_PHASE>" end="<END_OF_GREEN_PHASE>" \
      duration="<DURATION_OF_GREEN_PHASE>"/>
   ... further switch points ...
</tls-switches>
```

Each entry is written into a single line. The values have the following meanings:

- `junction`: The id of the tls that is responsible for the link
- `subid`: The sub-id of the tls that is (currently) responsible for the link
- `fromLane`: The id of the lane the link starts at
- `toLane`: The id of the lane the link ends at
- `begin`: Begin of this link's last green phase
- `end`: End of this link's last green phase
- `duration`: Duration of this link's last green phase

Recent changes:

- Changes in version 0.9.5
  - This output is available since version 0.9.5
- Changes in version 0.9.6
  - The element "switch" was renamed to "tlsswitch" in version 0.9.6

TLS Switch States

This output saves tls-states as the TLS States - output does but not every second but only at times the phases or the program (see also "Adding new Programs") change. The output is instantiated by adding the following definition into one of your additional files: `<timed_event type="SaveTLSSwitchStates" source="<TLS_ID>" dest="<OUTPUT_FILE>"/>`. The attributes have herein the following meanings:

- `type`: type of the event trigger; always "`SaveTLSSwitches`" herein
- `source`: The id of the traffic light which state shall be written
- `dest`: The file to save the state into

The output looks like this:

```
<tls-switch-states>
   <tlsstate time="<SIM_STEP>" id="<TLS_ID>" subid="<TLS_SUBID>"><STATE></tlsst
   ... further states ...
</tls-switch-states>
```

Each entry is written into a single line. The values have the following meanings:

- `time`: The simulation time this entry was generated for
- `id`: The id of the tls that is responsible for the link
- `subid`: The sub-id of the tls that is (currently) responsible for the link

Recent changes:

- Changes in version 0.9.6
  - This output is available since version 0.9.6

Vehicles Handling Revisited

In the normal case, SUMO is meant to simulate urban areas where vehicles may start their trips from any edge. Still, there also some other approaches to feed a simulation with a demand and some of them where implemented in SUMO. You have the following possibilities to add vehicles into your network:

- Insert vehicles on any edge

  In this case, a vehicle from the list will be inserted at the given time into the edge his route starts at. The position of the insertion is random (by now), the rightmost lane will be used.
- Insert vehicles on feeding edges

  This is approach is often used in conjunction with od-matrices; each of the districts described in such od-matrices contains a list of "feeding" or "source" edges. If you use feeding edges, your vehicles will be inserted similar to insertion on normal edges as described above, but they will be always inserted at the end of the edge and all lanes of the feeding edge will be used.
- Using emitter

  Emitter are used to insert vehicles into the network at a well defined position. An emitter may be placed on a certain lane and gets a list of vehicles (or a flow amount) to emit. We use this approach often to insert vehicles into the network at places where induct loops have measured the flows.

We will now describe the emitters more deeply.

Emitter

Emitters may be used to define flows using induction loops as input data. For such modelling attempt, you should place emitters at those positions on the network where the induction loops are located and convert the values retrieved from the induction loops to the format emitters may read. *(TBD add references)* The format is described below, together with some additional methods to ease generation of emitter files. If you are working with such inputs extensively, you may be also interested in what the DFROUTER does (see "Using Detectors and DFROUTER" for a further documentation).

Recent changes:

- Although emitters are available for a long time already, their description has been added while working on version 0.9.5

Basic Definition

You can place an emitter onto a lane by adding the following declaration to one of your additional-files:

```
<trigger id="<ID>" objecttype="emitter" pos="<POS>" objectid="<LANE_ID>" \
   [friendly_pos="x"] file="<DEFINITION_FILE>"/>
```

The fields have the following meanings:

- `id`: A string holding the id of the emitter.
- `objecttype`: Always "`emitter`"; indicates this trigger is an emitter
- `pos`: Position on the lane in meters; if positive, then the following must be ensured: 0<=*<POS>*<<*<LANE_LENGTH>*, if negative: 0>*<POS>*>-*<LANE_LENGTH>*; in this case the position will be counted from the lane's end.
- `objectid`: The id of the lane the emitter shall be placed on
- `friendly_pos`: optional; if this is set and the position (`pos`) is not valid, the detector will be

placed at the lane's end (0.1meter away from it).
- `file`: The file the emitter shall read the definition of what/how/when to emit from

An emitter needs further information to know when, how many and what kind of vehicles shall be emitted. All this information must be written into *<DEFINITION_FILE>*. The easiest way to describe vehicle emissions herein is to list all of them explicitely:

```
<triggeredsource>
   <emit id="veh1" time="0" vehtype="my_type" route="my_route" speed="13.9"/>
   <emit id="veh2" time="4" vehtype="my_type" route="my_route" speed="13.9"/>
   <emit id="veh3" time="8" vehtype="my_type" route="my_route" speed="13.9"/>
</triggeredsource>
```

Using such a definition only would raise error because we have named the vehicle types and the routes but did not define them. We can either define them within another additional file or within a route file but we have to ensure that they're loaded before the emission definition is (see "Using the Files in a correct Way" on loading order). Let's assume we have done it. In this case, using such a definition we would emit three vehicles, having the names "veh1", "veh2", and "veh3" as given within the id-field, all being of type "my_type". All vehicles use the same route, "my_route", and will start with a velocity of 0 at the simulation seconds 0, 4, and 8. To summarize, a vehicle emission within an Emitter definition is described as following: `<emit [id="<VEHICLE_ID>"] [vehtype="<VEHICLE_TYPE>"] time="<EMISSION_TIME>" [route="<VEHICLE_ROUTE>"] [speed="<INITIAL_SPEED>"]/>`. The meanings of these values are:

- `id`: The id of the vehicle to emit
- `vehtype`: Name of the vehicle type the vehicle to emit shall have
- `time`: The time at which the vehicle shall be emitted (in simulation seconds)
- `route`: Name of the route the vehicle shall use
- `speed`: The speed the vehicle shall be emitted with in m/s

As you can see, several of the fields are marked as optional. If no id is given, the id will be constructed automatically. The vehicle will then have a name made up from the emitter's id followed by the time step the vehicle shall be emitted at and a running number, all divided by a '_' ("*<EMITTER_ID>_<DEPART>_<RUNNING>*"). Also, the emission speed is optional. If not given, the minimum of the maximum speed allowed on the lane and the vehicle's maximum velocity is used. If the emission time lies before the simulation begin, the vehicle will be discarded. The following sections describe how one can omit explicite attributes for vehicle type and route.

Describing Route Distributions

To avoid computing and assigning a vehicle type and a route to each vehicle emission definition explicitely, you can define a probability distribution by which routes/types are chosen from a set. For the routes, you can do this as shown in the next example:

```
<triggeredsource>
   <routedistelem id="my_route1" probability=".2"/>
   <routedistelem id="my_route2" probability=".8"/>

   <emit id="veh1" time="0" vehtype="my_type" speed="13.9"/>
   <emit id="veh2" time="4" vehtype="my_type" speed="13.9"/>
</triggeredsource>
```

Now, a random route is assigned to a vehicle, "my_route1" with a probability of .2, "my_route2" with a probability of .8. The probabilities are normed automatically, that means that you can also use numbers that do not sum to 1. Each occuring `routedistelem` will be added to the distribution (see also "Resetting the Distributions"). The meanings of the attributes of a `routedistelem`-element are:

- `id`: The name of the route to use (the route must have been loaded in prior to the occurence of the `routedistelem`-element)

- `probability`: The probability (value/sum of probabilities) of choosing the route

Describing Vehicle Type Distributions

Vehicle types may be assigned to vehicles from distributions, too:

```
<triggeredsource>
   <vtypedistelem id="my_type1" probability=".8"/>
   <vtypedistelem id="my_type2" probability=".8"/>

   <emit id="veh1" time="0" route="my_route" speed="13.9"/>
   <emit id="veh2" time="4" route="my_route" speed="13.9"/>
</triggeredsource>
```

In this example the probabilities for using one of the types are equal. The probabilities are normed automatically, that means that you can also use numbers that do not sum to 1. Each occuring `vtypedistelem` will be added to the distribution (see also "Resetting the Distributions"). The meanings of the attributes of a `vtypedistelem`-element are:

- `id`: The name of the vehicle type to use (the vehicle type must have been loaded in prior to the occurence of the `vtypedistelem`-element)
- `probability`: The probability (value/sum of probabilities) of choosing the vehicle type

Resetting the Distributions

As said before, all occurences of `vtypedistelem` are stored into the same distribution. This also holds for the occurences of `routedistelem`. Now, one maybe wants to model different distributions over time. To allow this, you can add a "`reset`"-element to your description:

```
<triggeredsource>
   <vtypedistelem id="my_type1" probability=".5"/>
   <vtypedistelem id="my_type2" probability=".5"/>
   <routedistelem id="my_route1" probability=".2"/>
   <routedistelem id="my_route2" probability=".8"/>

   <emit time="10" speed="13.9"/>
   ... further vehicle emits ...
   <emit time="20" speed="13.9"/>

   <reset/>

   <vtypedistelem id="my_type3" probability=".5"/>
   <vtypedistelem id="my_type4" probability=".5"/>
   <routedistelem id="my_route3" probability=".2"/>
   <routedistelem id="my_route4" probability=".8"/>

   <emit time="30" speed="13.9"/>
   ... further vehicle emits ...
   <emit time="40" speed="13.9"/>

</triggeredsource>
```

This would force the emitter to reset all distributions after emitting the vehicle at time 20. While vehicles emitted within the times 10 and 20 would use the vehicle types "my_type1" and "my_type2" and routes "my_route1" and "my_route2", the vehicles emitted between time 30 and 40 - after the `reset`-element - would use the vehicle types "my_type3" and "my_type4" and the routes "my_route3" and "my_route4".

Using Flows

Instead of describing each vehicle emission explicitely, you can specify a flow to emit. In this case, vehicle type and routes distributions must be given:

```
<triggeredsource>
   <vtypedistelem id="my_type1" probability=".5"/>
   <vtypedistelem id="my_type2" probability=".5"/>
   <routedistelem id="my_route1" probability=".2"/>
   <routedistelem id="my_route2" probability=".8"/>

   <flow no="1800" end="10"/>
   <flow no="900" end="20"/>
</triggeredsource>
```

The meaning of the attributes of a `flow`-element are:

- `no`: The flow to use in veh/h.
- `end`: The end of the interval for which this flow shall be emitted. If <0 (default) the flow will be used until the simulation's end.

Traffic Management and Other Structures

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

Traffic Lights

Normally, NETCONVERT 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. Besides the following descriptions, a further document, "SUMO - More on... Traffic Lights [http://sumo.sourceforge.net/docs/gen/sumo_moreon_tls.shtml]", exists which describes the usage of traffic lights more deeply.

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 advice 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.

Recent changes:

- Loading of additional tls programs is implemented since version 0.9.4
- The `incl anes` tag has been removed from the network description since version 0.9.4
- The tag `keyno` has been renamed to `subkey` since version 0.9.4

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.

Recent changes:

- WAUTs are implemented since version 0.9.4

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.

Public Transport

Possibilities to simulate public transport were firstly added in version 0.9.3. By now you may define positions of bus stops and let vehicles ("busses") stop at these positions for a pre-given time. Definitions of bus stop locations in SUMO have the following format: `<trigger id="<BUS_STOP_ID>" objecttype="bus_stop" objectid="<LANE_ID>" from="<STARTING_POSITION>" to="<ENDING_POSITION>" [line="<LINE_ID>[;<LINE_ID>]*"]/>`. That means that a bus stop is an area on a lane. The parameters have the following meanings:

- `id`: id of the bus stop; must be unique
- `objecttype`: always "bus_stop" herein
- `objectid`: the id of the lane the busstop shall be located at
- `from`: the begin position on the lane (the lower position on the lane) in meters
- `to`: the end position on the lane (the higher position on the lane) in meters
- `line`: A list of names separated by a semicolon (';') meant to be the names of the bus lines that stop at this bus stop. This is only used for visualisation purposes.

Figure 6.1. Visualization of a bus stop in SUMO (from `<SUMO_DIST>/data/examples/extended/busses1`)

Vehicles must be informed that they must stop at a bus stop. The following example shows how this should be done (taken from `<SUMO_DIST>/data/examples/extended/busses1`):

```
<vtype id="BUS" accel="2.6" decel="4.5" sigma="0.5" length="15" maxspeed="7
        color="1,1,0"/>

<vehicle id="0" type="BUS" depart="0" color="1,1,0">
    <route>2/0to2/1 2/1to1/1 1/1to1/2 1/2to0/2 0/2to0/1 0/1to0/0 0/0to1/0 1
                            2/0to2/1</route>
    <stop bus_stop="busstop1" duration="20"/>
    <stop bus_stop="busstop2" duration="20"/>
    <stop bus_stop="busstop3" duration="20"/>
    <stop bus_stop="busstop4" duration="20"/>
</vehicle>
```

What we have here is a vehicle named "0" being a "BUS". "BUS" is a referenced type declared earlier. The vehicle has an embedded route (written by hand in this case) and a list of stop places. Each stop place is described by two attributes, "`bus_stop`" and "`duration`" where "`bus_stop`" is the name of the bus stop the vehicle shall halt at and "`duration`" is the time the vehicle shall wait at the bus stop in seconds. Please remark that the order of bus stops the vehicle shall halt at must be correct.

You may also let a vehicle stop at another position than a bus stop. The complete definition of a vehicle's stop is: `<stop ( bus_stop="<BUS_STOP_ID>" | lane="<LANE_ID>" pos="<POSITION_AT_LANE>" ) duration="<HALTING_DURATION>"/>`. This means you can *either* use a bus stop *or* a lane position to define where a vehicle has to stop.

Again the list of attributes for the "`stop`"-element of a vehicle:

- Either:
  - `bus_stop`: id of the bus stop the vehicle shall halt at; the bus stop must be previously declared
- or:
  - `lane`: id of the lane the vehicle shall stop at; the lane must be within the network
  - pos: Position on the lane the vehicle shall stop at; double
- `duration`: the time the vehicle shall halt at the bus stop in seconds; int, mandatory

Examples:

- `<SUMO_DIST>/data/examples/extended/busses1` shows a small example for defining bus stops and letting a bus halt at them
- `<SUMO_DIST>/data/examples/extended/3busses1` is almost the same as `<SUMO_DIST>/data/examples/extended/busses1` but three busses are driving here and the first bus stop is longer than the others. This example shows how the length of bus stops determines how many busses actually can stop here.
- `<SUMO_DIST>/data/examples/extended/vehicle_stops` shows a small example where a vehicle halts

Some extensions still to be done:

- Definition of public transport lines instead of giving a list of stops for each vehicle?
- Halting times dependent to the number of passengers within the vehicle
- Optionally do not let vehicles halt if no person wants to leave/enter

Variable Speed Signs (VSS)

One of the trigger objects that may be specified within an additional file allows the simulation of variable speed signs. The syntax for such an object is: `<trigger id="<VSS_ID>" objecttype="lane" objectid="<LANE_ID>" attr="speed" file="<DEF_FILE>"/>`. This trigger is typed to be a vss by the combination of the values of the attributes `objecttype="lane"` and `attr="speed"`. Although no other combinations are implemented so far, this combination forces the simulation to change the attribute "speed" of a "lane"-object, exactly what vss do. Of course, the vehicles themselves do not override this maximum velocity what does not exactly represent the reality.

You may have noticed that a file name must be supplied, called `<DEF_FILE>` within the schema above. This file must contain the information about when a certain speed shall be set onto the lane. This file has the following format:

```
<vss>
   <step time="<TIME>" speed="<SPEED>"/>
   <step time="<TIME>" speed="<SPEED>"/>

   ... further entries ...

   <step time="<TIME>" speed="<SPEED>"/>
</vss>
```

Each `step` is a combination of the `time` the next new speed shall be set and the `speed` to set itself.

A small example for usage of vss' within SUMO may be found in `"data/examples/extended/variable_speed_signs"`.

Rerouter

Rerouter change the route of vehicles as soon as a vehicle moves on a specified edge. Although implemented earlier, were firstly described and tested within version 0.9.5.

A rerouter is set into the simulated by adding the following line to an "additional file": `<trigger id="<REROUTER_ID>" objecttype="rerouter"`

`<EDGE_ID>]"` `file="<DEFINITION_FILE>"` `[probability="<PROBABILITY>"]/>`. As you may see, rerouter may be placed on several edges, at least one edge is necessary. Furthermore, you may already give within this definition how many vehicles shall be rerouted by giving a number between 0 (none) and 1 (all). In addition to this definition a description file (`<DEFINITION_FILE>`) must be given which describes the behaviour of the rerouter over time. The definition values are

- `id`: the id of of the rerouter
- `objecttype`: always "`rerouter`"
- `objectid`: an edge id or a list of edge ids where vehicles shall be rerouted
- `file`: path to the definition file
- `probability`: the probability for vehicle rerouting (0-1)

Each definition of what a rerouter shall do is embedded in an interval definition which describes within which time period the rerouter shall work. This is set up as following:

```
<rerouter>
   <interval begin="<BEGIN_TIME>" end="<END_TIME>"/>
      ... action description ...
   </interval>

   ... further intervals ...

</rerouter>
```

A rerouter may work in several different ways. Within a time period you may close an edge, or assign new destinations or pregiven routes to vehicles. The next subchapters will describe these possibilities and how to describe them in detail.

Examples:

Recent changes:

- A complete description of rerouters was added in version 0.9.5; in accordace, definitions of rerouters have changed

Closing a Street

A "`closing_reroute`" forces the rerouter to close the edge `<EDGE_ID>`. Vehicles which normally would pass this edge will get a new route as soon as they reach one of the edges given in the `objectid`-attribute of the rerouter's declaration. A definition may look like this:

```
<rerouter>
   <interval begin="<BEGIN_TIME>" end="<END_TIME>"/>
      <closing_reroute id="<EDGE_ID>"/>
   </interval>

   ... further intervals ...

</rerouter>
```

The attributes used within such definitions are:

- `id`: the id of the closed edge; mandatory string, the id must be the id of an edge within the network

Assigning a new Destination

A "`dest_prob_reroute`" forces the rerouter to assign a new route to vehicles which pass one of the edges defined in the `objectid`-attribute of the rerouter's declaration. A new route destination is used, defined by the name of a new destination in the according element:

```
<rerouter>
   <interval begin="<BEGIN_TIME>" end="<END_TIME>"/>
      <dest_prob_reroute id="<EDGE_ID1>" probability="<PROBABILITY1>"/>
      <dest_prob_reroute id="<EDGE_ID2>" probability="<PROBABILITY2>"/>
   </interval>

   ... further intervals ...

</rerouter>
```

The route is computed automatically using the Dijkstra-algorithm and starting at the edge the vehicle is located at and ending at the new destination.

The attributes used within such definitions are:

- `id`: the id of the new destination; mandatory string, the id must be the id of an edge within the network
- `probability`: the probability with which a vehicle will use the given edge as destination; mandatory float, should be between 0 and 1; the sum of the probabilities should be 1

Assigning a new Route

A "`route_prob_reroute`" forces the rerouter to assign a new route to vehicles which pass one of the edges defined in the `objectid`-attribute of the rerouter's declaration. In this case, the id of a complete route shall be supplied instead of a new destination:

```
<rerouter>
   <interval begin="<BEGIN_TIME>" end="<END_TIME>"/>
      <route_prob_reroute id="<ROUTE_ID1>" probability="<PROBABILITY1>"/>
      <route_prob_reroute id="<ROUTE_ID2>" probability="<PROBABILITY2>"/>
   </interval>

   ... further intervals ...

</rerouter>
```

The attributes used within such definitions are:

- `id`: the id of a new route to assign; mandatory string, the id must be the id of a previously loaded route
- `probability`: the probability with which a vehicle will use the given edge as destination; mandatory float, should be between 0 and 1; the sum of the probabilities should be 1

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.

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.

We want to ask you to supply us any comments on this topic - it is not completely designed, yet.

Recent changes:

- A first support for vehicle classes was added in version 0.9.5

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 6.1. 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 6.2. 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	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. This process is described in chapter "Defining allowed Vehicle Types [ht-

".

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="7
        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.

Mixing Closings/Openings of Roads for Vehicle Classes

The importer for XML-edge description uses two lanes attributes, `allow` and `disallow` (see Defining allowed Vehicle Types [ht-

). Within the resulting network the values supplied this way are stored by listing all allowed and disallowed vehicle classes, divided by a ';'. Here, disallowed vehicle classes are marked by a leading '-'. This means that if a lane shall not allow pedestrians, it should have the attribute `vclasses="-pedestrian"`.

A vehicle class may use a lane if

- there is no vehicle class allowed/disallowed on this lane
- the vehicle class matches a class allowed on this lane if any allowed class is defined for this lane
- the vehicle class does not match any of the defined disallowed vehicle classes on for this lane

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 `route-steps` value not equal to zero or place your additional files at the begin of the route-files list.

Other Topics

This chapter includes some problems not described, yet.

Simulation of Accidents

SUMO uses a collision-free traffic flow model. So if everything works as it should, no accidents should occure. If you want to model an accident you have the following possibilities:

- Use variable speed signs to set a lane's maximum velocity down (see chapter "Variable Speed Signs [http://sumo.sourceforge.net/docs/gen/user_chp06.shtml#user_chp06-management-vss]")
- Let a vehicle stop at a predefined position (see chapter "Public Transport [http://sumo.sourceforge.net/docs/gen/user_chp06.shtml#user_chp06-management-public]")

Still, in some cases, for example if you insert a tls with no yellow phase, collisions may occure within the simulation. Earlier versions of SUMO reported an error in such cases and quit. We decided to change this behaviour. By now, the simulation reports a warning in such cases and tries to solve the problem internally, either by changing the position of the last car or - if this does not work because the lane the accident happened at is full - by removing one of the cars and trying to reinsert it as soon as possible (#TBD see also "Vehicle Teleportation" #). You still may force the simulation to

quit as soon an "accident" happens using the option `--quit-on-accident`.

Missing

--route-steps

--check-accidents

--too-slow-rtf

--no-duration-log

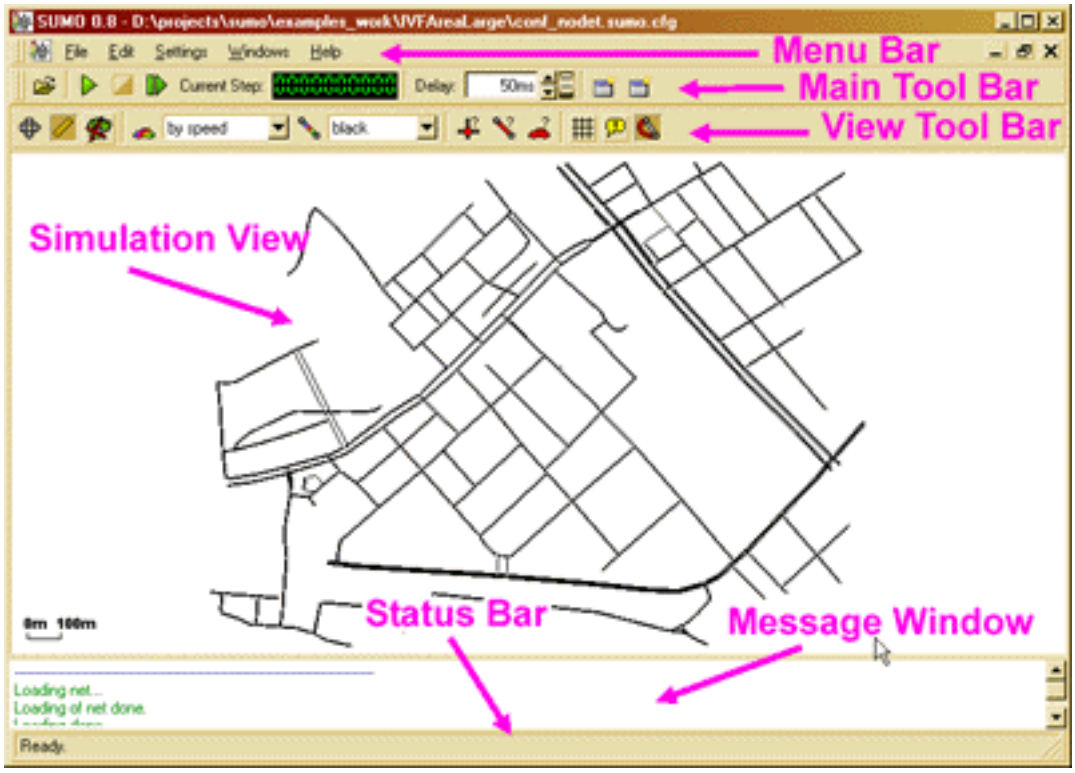
loading & saving of states

--time-to-teleport

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)



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 SUMO-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 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.

Caution

Under current development. (Better do not use it)

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 SUMO.

Tool Bar

File Operations

- Open Button

  Opens a file dialog that lets you choose a SUMO-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.

Caution

Aggregated views need additional simulation time and memory. For larer netweorks it seems wise to switch them off using the menu-bar option Settings->Application Settings->Allow aggregated Views

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.
- Show Legend - Button

  Switches the legend (only the scale) on and off.

View Settings

- Change Lane Colouring Scheme - Button

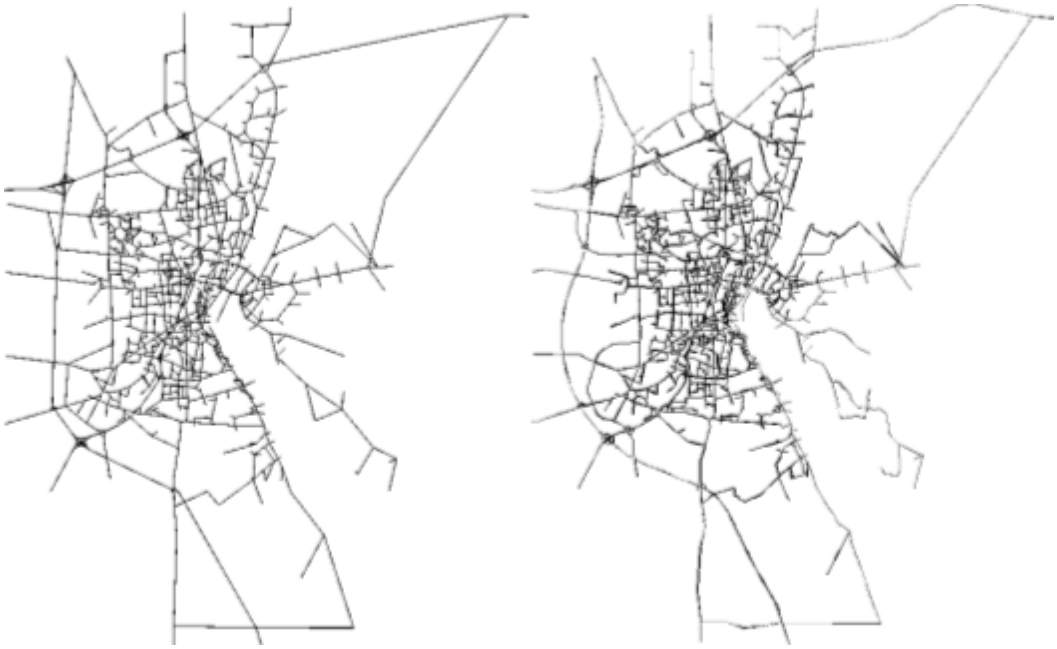
  The combobox allows you to choose how the view shall determine a displayed lane's colour. The following coluring schemes are avaliable for lanes:
  - black: All lanes are black
  - by purpose: normal lanes are coloured black, sources green, sinks red and internal lanes blue
  - by speed: The lanes are coloured by the speed allowed on them, blue for fast lanes, red for slow
- Show Grid - Button

  Turns the visualisation f the grid the network is mapped onto on/off. This is rather a debugging feature.
- Show Tool Tips - Button

  If this button is activated, the id for the object the mouse pointer is over will be displayed.
- Show Full Geometry - Button

  Switches between full and simple - geometry. The picture beyond shows the difference between full and simple geometry mode.

  Figure 7.2. The difference between simple (left) and full (right) geometry mode



Additional Microscopic View Controls

Tracking Settings

- Locate Vehicle - Button

  Opens a window that allows to choose a vehicle name from the set of vehicle currently driving within the simulation. Pressing ok with a chosen vehicle zooms the view to this vehicle.

Caution

Under current development. (Better do not use it, may terminate the application with an error)

View Settings

- Change Vehicle Colouring Scheme - Button

  The combobox allows you to choose how the view shall determine a displayed vehicle's colour. The following coluring schemes are avaliable for vehicles:
  - by speed: The vehicle colour change continuously from blue to red with decreasing speed
  - specified: The loaded vehicle colours are used
  - by type: The vehicles are coloured as defined within the vehicle's type definition
  - by route: The vehicles are coloured as defined within the vehicle's route definition
  - random#1: Vehicles are coloured randomly (each vehicle has a certain colour)
  - random#2: Vehicles are coloured randomly (each vehicle has a certain colour); Quite the same as "random#1", but a different method is used to compute the random colour
  - lanechange#1: ... tbd ...
  - lanechange#2: ... tbd ...
  - lanechange#3: ... tbd ...
  - waiting#1: ... tbd ...
- Change Lane Colouring Scheme - Button

The combobox allows you to choose how the view shall determine a displayed lane's colour. The following coluring schemes are avaliable for lanes:
- black: All lanes are black
- by purpose: normal lanes are coloured black, sources green, sinks red and internal lanes blue
- by speed: The lanes are coloured by the speed allowed on them, blue for fast lanes, red for slow
- by selection: Selected lanes are coloures blueish, the other black

Additional Aggregated View Controls

View Settings

- Change Lane Colouring Scheme - Button

  The combobox allows you to choose how the view shall determine a displayed lane's colour. The following coluring schemes are avaliable for lanes:
  - by density: The lanes are coloured by the mean density of vehicles over them
  - by mean speed: The lanes are coloured by the mean speed of vehicles passing them, blue for fast lanes, red for slow
  - by mean halts:

Caution

... not working, yet. ...
  - black: All lanes are black
  - by purpose: normal lanes are coloured black, sources green, sinks red and internal lanes blue
  - by speed: The lanes are coloured by the speed allowed on them, blue for fast lanes, red for slow
  - by selection: Selected lanes are coloures blueish, the other black
- Remembering-factor dialer

  This dialer enables you to change the "memory" of the aggregated 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.

- Add To Selected

  Adds the object to the list of selected objects.
- Remove from Selected

  Removes the object from the list of selected objects.
- Show Parameter

  Opens a window in which the object's fixed and dynamic parameter are displayed.

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.

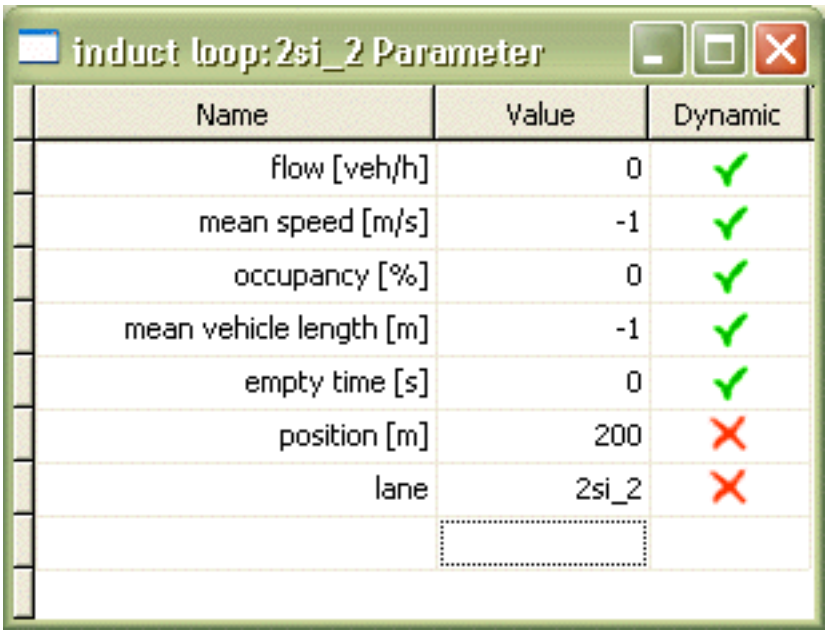
Caution

The handling of selected items is not yet designed cmopletely. Parts of the gui's behavior may change in the next time.

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 7.3. A sample Parameter Window (for an induct loop in this case)

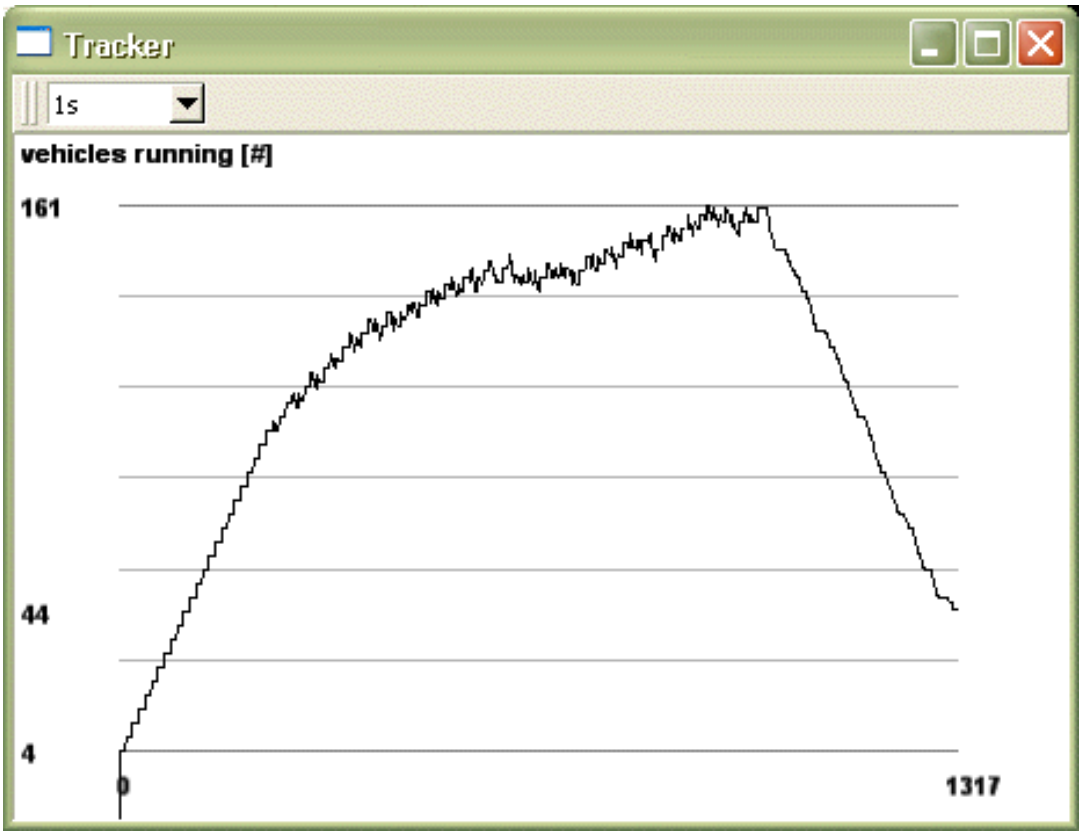

induct loop:2si_2 Parameter

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	✗

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.

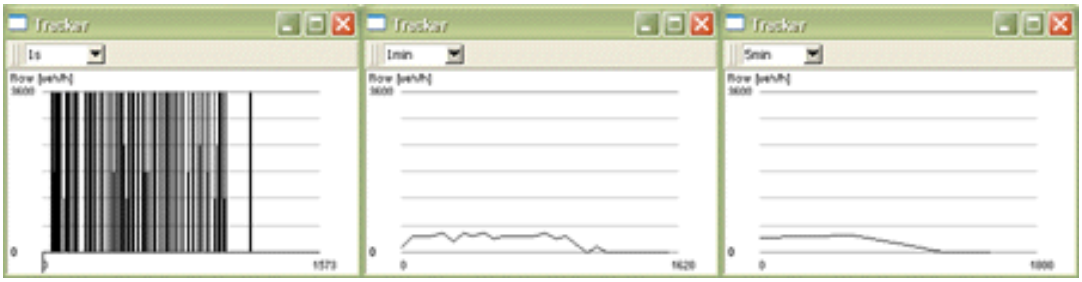
Pressing the right mouse button when being over a line marked as dynamic will show a small popup window with only a single command: "Open in new Tracker". Choosing this option will allow you to open another window where this parameter's values will be shown as a time line over the simulation run.

Figure 7.4. 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.

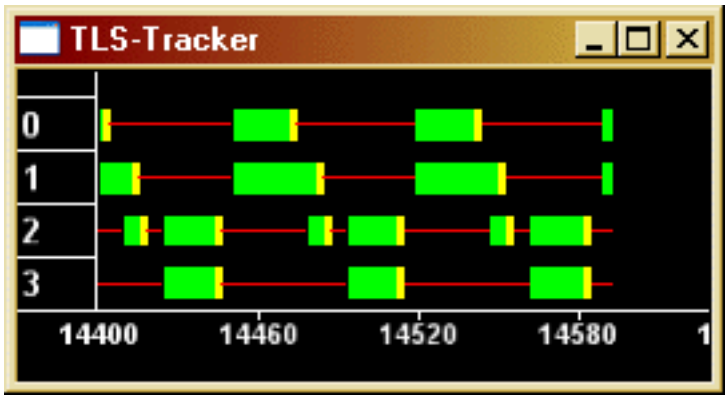
Figure 7.5. A sample usage of the aggregation option (for an induct loop in this case, for aggregation times of 1s, 1min, 5min (from left to right))



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 7.6. A sample usage of the tls-tracker



Additional Geometry Files

Since version 0.9.4 it is possible to load additional geometry files into GUISIM which may contain definition of polygons or points of interest. These shapes are currently meant to improve a simulation's appearence and to allow an easier debugging. No special interaction with them is implemented, yet.

Both polygons and points-of-interest may be located at a "layer". Shapes with lowest layer values are drawn first so that they can be below those with a higher layer number. The network itself is drawn as layer 0. An additional file may contain definitions for both points-of-interest and polygons. Any name may be used for the embedding element.

In the following subchapters, it is described how polygons and points-of-interest may be defined.

Polygon Definitions

A polygon is defined as following:

```
<poly id="<POLYGON_ID>" type="<TYPENAME>" color="<RED>,<GREEN>,<BLUE>" \
   fill="<FILL_OPTION>" layer="<LAYER_NO>"><POINT> [<POINT>]+</poly>
```

These attributes have the following meanings:

- `id`: The id (a unique name) of the polygon; mandatory string
- `type`: A typename for the polygon. This value has no meaning; optional string, default: ""
- `color`: The color with which the polygon shall be displayed; <RED>, <GREEN>, and <BLUE> must be floating point numbers between 0 and 1. They are devided using a ',' (no space); mandatory
- `fill`: An information whether the polygon shall be filled; optional bool, default: false
- `layer`: The number of the layer in which the polygon lies; optional int, default: -1

Point-of-interest Definitions

A point-of-interest is defined as following:

```
<poi id="<POLYGON_ID>" type="<TYPENAME>" color="<RED>,<GREEN>,<BLUE>" \
   layer="<LAYER_NO>" [(x="<X_POS>" y="<Y_POS>") | (lane="<LANE_ID>" pos="<LANE
```

It means that the position a point-of-interest is located at may be given either using explicite x/y-coordinates or a lane name and a position on this lane. So, the attributes have the following meanings:

- `id`: The id (a unique name) of the poi; mandatory string
- `type`: A typename for the poi. This value has no meaning; optional string, default: ""

- `color`: The color with which the poi shall be displayed; <RED>, <GREEN>, and <BLUE> must be floating point numbers between 0 and 1. They are devided using a ',' (no space); optional, default "1,0,0"
- `layer`: The number of the layer in which the polygon lies; optional int, default: 1
- `x`: The position of the poi along the x-axis; float
- `y`: The position of the poi along the y-axis; float
- `lane`: The name of the lane the poi is located at; string, the lane must be a part of the loaded network
- `pos`: The position on the named lane at which the poi is located at; float

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.

Using Configuration Files

Most simulations have to be executed more than only one time. Furthermore, some experiments require the execution of similar, slightly different settings, for example the same network with a different route set. To avoid retyping of all parameter at the input line, all of the main applications can be fed with a configuration file. This configuration file contains the values the user normally would give to the program at the command line. For example, instead of typing

```
duarouter --cell=myCellFile --net=mySUMONet.net.xml --output-file=MySUMORoutes.
   -b 0 -e 3600
```

you can start the router with a configuration file only:

```
duarouter -c=myConfig.rou.cfg
```

The `-c <FILE>` - option may be passed to all of the package's main applications.

Of course, you have to build the configuration file "`myConfig.rou.cfg`" first. You can find templates for configuration files within the data/cfg_templates - folder and all examples coming with the release contain configuration files, too.

A configuration file is a simple XML-file in which each of the command line parameters is represented as a XML-element with the parameter's value being given as text between the begin and end tag of this parameter. So if you want to set a parameter "foo" to the value "bar" within your configuration file, write `<foo>bar<foo/>` into the configuration file. Do not forget that each XML-file has to have a root element, so that the whole configuration file would look like this:

```
<configuration>
   <foo>bar<foo/>
</configuration>
```

Between the starting at the ending tag, any type of values may be set, use a 'x' to mark boolean values as set. If a parameter allows a set of values (normally separated by a ';'), you have to use a single element and embed these value into it as you would on command line. A different approach will maybe be invented in future. You can find the templates for each of the package's application's configuration files within the folder "`<SUMO_DIST>/data/cfg_templates`".

Additional Meta-Information

All applications of the SUMO-package print a help-screen is printed including all options the application knows when the application is started with the `--help` (`-?` for short) option. You can also list all current option settings using `--print-options` (or `-p` for short).

Recent changes:

- This chapter has been moved to this place while working on version 0.9.5
- The option `--version` that printed the currnt build number was removed in version 0.9.5. As we assume our users to build the software by themselves, a build number does not really make sense.
- The description of `--print-options` was added in version 0.9.5.

Additional Tools

You can some find helpful tools within the `<SUMO_DIST>/tools` - folder. We will now introduce some of them. The following chapters are devided by the topic the tools cover.

Polygon Conversion

Since version 0.9.5 a further application was added to the suite: POLYCONVERT, a tool which allows you to convert polygons from Elmar's format into a description that may be used by SUMO. As the offset that was applied (automatically, see!!!) to the network during the conversion using NETCONVERT is needed, one has to supply the network name using `--net-file <SUMO_NET>` (`--net` or `-n` for short). Additionally the name of the file that contains the polygons to import must be given using `--elmar <ELMAR_POLYGON_FILE>`. The conversion from geocoordinates to cartesian is recommended, initiated using `--use-projection` and defined using `--proj <PROJ_DEFINITION>` (see also "Converting from Geocoordinates").

Defaults for the polygon's color and layer as well as a name prefix and the name of the type to assign can be given using the options `--color <COLORDEF>`, `--layer <LAYER_NO>`, `--prefix <PREFIX>`, and `--type <TYPENAME>`, respectively. As some inputs may contain different polygon types, you can also use a file which contains a type map which defines which values shall be set in dependance to the type. A single entry for this typemap should look like this: `<polytype id="<PREVIOUS_NAME>" name="<NEW_NAME>" color="<COLORDEF>" fill="<BOOL>" layer="<LAYER_NO>" discard="<BOOL>"/>`. The values are:

- `id`: The name of the type as read from the input file
- `name`: The name to use for the type in the output (type-name replacement)
- color: Definition of the color to assign
- fill: Information whether a filling of the polygon must be prohibited
- layer: Layer to use for this type of polygons
- discard: Information whether polygons of this type shall not be written to output

An example type-map for Elmar's polygons can be found in `<SUMO_DIST>/data/`add/elmar_type_map.xml. It is given to POLYCONVERT using the `--typemap <TYPEMAP_FILE>` option.

Since version 0.9.6, POLYCONVERT can also import single points of interest from Elmar's point-collection files. To import such a file use the option `--elmar-points <FILENAME>`, where `<FILENAME>` is the name of the file to import. You can use a type map in this case, too. In this case, the attribute "filled" will be ignored, all other attributes are processed as in the case of importing polygons.

Also since this version, POLYCONVERT can import polygons and pois from Visum-networks. The options herefore are `--visum <VISUM_NET>` and `--visum-points <VISUM_NET>`. In the first case, polygons from "BEZIRK" and "GEBIET" are imported, in the second "POI". "POIKATEGORIE" is parsed in the second steps and the values stored herein are used as type names; for "BEZIRK", the type name "district" is used, "area" for "GEBIET". These type names may be references in the type map file.

In some cases, it is wished not to import all polygons/pois. You can constrain which polygons/pois shall be written using the by assigning the attribute "`discard`" to certain types of polygons(pois within the type-map. You can also prune those polygons/pois that are not lying within a certain bounding box. This is done by calling POLYCONVERT with the option `--prune.boundary <BOUNDARY>`. `<BOUNDARY>` is in this case the bounding box in which a polygon/poi must be located in order to be written into the output. It is a list of four floats, separated using ',' that describe the minimum x-value, the minimum y-value, the maximum x-value, and the maximum y-value of the bounding box. If one wishes to use the network's dimensions as the bounding box, he/she can do this using the option `--prune.on-net`. Additionally, one can supply offsets to the network's dimensions using `--prune.on-net.offsets <BOUNDARY>`.

All options (#TBD excluded loading configuration and output options, see #):

`( --net-file | --net | -n ) <SUMO_NET>`	The SUMO-net to use as reference. *Mandatory, type:filename, default: none*
`--elmar <ELMAR_POLYGON_FILE>`	Reads polygons from the given Elmar polygon file. *Optional, type:filename, default: none*
`--elmar-points <ELMAR_POI_FILE>`	Reads pois from the given Elmar pointcollection file. *Optional, type:filename, default: none*
`--visum <VISUM_NET>`	Reads polygons from the given VISUM net. *Optional, type:filename, default: none*
`--visum-points <VISUM_NET>`	Reads pois from the given VISUM net. *Optional, type:filename, default: none*
`--typemap <TYPEMAP_FILE>`	Reads type maps from the given file. *Optional, type:filename, default: none*
`--use-projection`	Enables conversion from geocoordinates to cartesian. *Optional, type:bool, default: false*
`--proj <PROJ_DEFINITION>`	Defines the projection to use. *Optional, type:string, default: "+proj=utm +zone=33 +ellps=bessel +units=m"*
`--color <COLORDEF>`	Defines the color to use as default. *Optional (pregiven), type:color, default: "0.2,0.5,1." (light blue)*
`--layer <LAYER_NO>`	Defines into which layer the polygons shall be put by default. *Optional (pregiven), type:int, default: -1 (one layer below the road network)*
`--prefix <PREFIX>`	Defines the type-dependant prefix to apply to polygons. *Optional (pregiven), type:string, default: <empty>*
`--type <TYPENAME>`	Defines the name of the type to set for the polygons. *Optional (pregiven), type:string, default: "water"*
`--prune.boundary <BOUNDARY>`	Defines the clipping bounding box in which a polygon/pois must lie in order to be written to the output. *Optional, type:boundary definition, default: none*
`--prune.on-net`	Let POLYCONVERT use the network dimensions as the clipping bounding box. *Optional (pregiven), type:bool, default: false*
`--prune.on-net.offsets <BOUNDARY>`	Defines additional offsets that are added to the network dimensions' components. *Optional, type:boundary definition, default: "0,0,0,0"*

Recent changes:

- POLYCONVERT is available since version 0.9.5
- Since version 0.9.6, POLYCONVERT is able to import Elmar's pointcollection files
- The possibility to constrain the imported points using bounding boxes was introduced in version 0.9.6
- Since version 0.9.6, POLYCONVERT is able to import Visum polygons and points

Helpers for DUA-Computation

dua-iterate.pl

This script performs a dua computation by runing the DUAROUTER and SUMO a given number of times and using the previous outputs. A detailed description may be found in the subchapter "Automatic Iteration using 'dua-iterate.pl' [http://sumo.sourceforge.net/docs/gen/user_chp05.shtml#user_chp05-dua-iterate]".

Usage: **dua-iterate.pl *<PATH_TO_SUMO_BINARIES>* [*[<BEGIN_ITERATION_STEP>]*]*<END_ITERATION_STEP>***

Output: see "Automatic Iteration using 'dua-iterate.pl' [http://sumo.sourceforge.net/docs/gen/user_chp05.shtml#user_chp05-dua-iterate]"

Location: `<SUMO_DIST>/tools/dua_tools`

Handling Routes and Route Alternatives

oldStyle2newStyle_Routes.pl

This tool converts route files as generated by DUAROUTER/JTRROUTER from their old-style representation where the route and the according vehicle where in separate tags into the new style where the route-description is.

Usage: **oldStyle2newStyle_Routes.pl *<SUMO_ROUTES_FILE>***

Output: The tool prints the modified route file in the new-style on the command line

Location: `<SUMO_DIST>/tools/route_tools`

randomizeDepart.pl

This tool randomizes the departure time of vehicles within a given route/route alternatives file.

Usage: **randomizeDepart.pl *<SUMO_ROUTES_FILE> <MAX_DEPARTURE_TIME>***

Output: The tool prints the modified route / route alternatives file in the new-style on the command line

Location: `<SUMO_DIST>/tools/route_tools`

Caution

This tool is meant to be used for tests only - routes in randomized order may yield in an unexpected behaviour!

removeRouteId.pl

Removes the ids of routes from their description within the given route file.

Usage: **removeRouteId.pl *<SUMO_ROUTES_FILE>***

Output: The tool prints the modified route file in the new-style on the command line

Location: `<SUMO_DIST>/tools/route_tools`

Caution

This tool is meant to be used for tests only - you may get an unexpected behaviour if you delete route ids which are still needed!

removeRouteReference.pl

Removes the references to routes from the descriptions of vehicles within the given route file.

Usage: removeRouteReference.pl ***<SUMO_ROUTES_FILE>***

Output: The tool prints the modified route file in the new-style on the command line

Location: `<SUMO_DIST>/tools/route_tools`

Caution

This tool is meant to be used for tests only - you may get an unexpected behaviour if you delete the information which route shall be used if it still needed!

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 SUMO (both command line and GUI-version)
  - *.netc.cfg

    Configuration file for NETCONVERT
  - *.netg.cfg

    Configuration file for NETGEN
  - *.dua.cfg (sometimes also *.rou.cfg)

    Configuration file for DUAROUTER
  - *.jtr.cfg

    Configuration file for JTRROUTER
  - *.od2t.cfg

    Configuration file for OD2TRIPS
- Data files:
  - *.net.xml

    SUMO - network file

    Contents: the SUMO-network including definitions for all streets, lanes and junctions

    Generated by: NETCONVERT or NETGEN

    Used by: SUMO, GUISIM, DUAROUTER, JTRROUTER, OD2TRIPS
  - *.rou.xml

    sumo - routes file

    Contents: vehicle type definitions, route definitions, vehicle definitions

    Generated by: DUAROUTER, JTRROUTER or the user

    Used by: SUMO, GUISIM, DUAROUTER
  - *.add.xml

    sumo - additional definitions file

    Contents: The definitions of detectors to build, sources to build etc.

    Generated by: the user

    Used by: SUMO, GUISIM
  - *.out.xml

    sumo - output file

    Contents: The "raw" output with edges, lanes and vehicles on them

    Generated by: SUMO, GUISIM

Used by: the user

- *.edg.xml

  NETCONVERT - edges file

  Contents: definitions of edges to build the network from

  Generated by: the user

  Used by: NETCONVERT
- *.nod.xml

  NETCONVERT - nodes file

  Contents: definitions of nodes to build the network from

  Generated by: the user

  Used by: NETCONVERT
- *.con.xml

  NETCONVERT- connection file

  Contents: definitions of connections between edges

  Generated by: the user

  Used by: NETCONVERT
- *.trips.xml

  trip definitions for DUAROUTER

  Contents: A list of trip definitions

  Generated by: the user

  Used by: DUAROUTER
- *.flows.xml

  flow definitions for JTRROUTER/DUAROUTER

  Contents: A list of flow definitions

  Generated by: the user

  Used by: JTRROUTER/DUAROUTER

- Other used file types
  - *.inp

    VISSIM network files
  - *.net

    VISUM network files
  - Destinations.txt, Detectors.txt, Flows.txt, Generators.txt, HVdests.txt, Incidents.txt, Int-Links.txt, Lanes.txt, Links.txt, Nodes.txt, Project Vars.txt, Signal Groups.txt, Signal Phases.txt, Signals.txt

    ARTEMIS input files
  - *.edg

    CELL edge files
  - *.nod

CELL node files

- *.driver

  CELL driver files

- *.rinfo

  CELL route files

- *.rindex

  index files generated for CELL route 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

Several examples are included in the distribution. You may find them in `<SUMO_DIST>/data/examples`. The subfolders have the following contents:

Table B.1. Supported example folders

Folder	Contains examples for (topic)	Contains examples for (application)
`dua`	the dynamic user assignment	SUMO/GUISIM, DUAROUTER
`duarouter`	building routes	DUAROUTER
`emissions`	vehicle emissions	SUMO/GUISIM
`extended`	additional simulation structures	SUMO/GUISIM
`jtrrouter`	building routes	JTRROUTER
`netbuild`	network generation	NETCONVERT, NETGEN
`output_tests`	simulation outputs	SUMO/GUISIM
`real_world`	networks from the real world	all
`simple_nets`	some simple scenarios	all
`traffic_lights`	traffic light algorithms	SUMO/GUISIM

SIMPLE_NETS: Basic Examples

Some smaller networks, mainly for testing purposes may be found within the `<SUMO_DIST>/data/examples` - folder. In respect to their shape, they are named `"eight..."`, `"cross..."` and `"box..."`. They differ in the number of lanes the edges have and whether a possibly existing junction is a simple right-of-way or a traffic light junction. The name pattern is as follows `<SHAPE>`Xl for networks with right-of-way junctions, where X is the number of lanes and `<SHAPE>`Xltl for networks with a traffic light. `<SHAPE>` is the name of the coarse shape of the network.

NETBUILD: Examples for NETCONVERT'S XML-Import

These examples may be found in the folder `<SUMO_DIST>/data/examples/netbuilding`. They shall show how networks may be imported/defined using the NETCONVERT application.

"types": Using Type Definitions to describe edges

Both of these two examples describe the same network (a simple cross), but "cross_notypes" describes each edge while "cross_usingtypes" uses types.

Location: `<SUMO_DIST>/data/examples/netbuilding/types`

"speed_in_kmh": Defining Edges' Speed Limits in km/h

Both of these two examples describe the same network (a simple cross) as the settings in `<SUMO_DIST>/data/examples/netbuilding/types`, but the speed is given in km/h, both within edge definitions (cross_notypes_kmh) and within the version using types (cross_usingtypes_kmh).

Location: `<SUMO_DIST>/data/examples/netbuilding/speed_in_kmh`

"shapes": Defining the Shapes of Edges

One can pass an additional parameter to edges to describe a more complex shape. This example is one for using this attribute.

Location: `<SUMO_DIST>/data/examples/netbuilding/shapes/hokkaido-japan`

ROUTER: Examples for DUAROUTER and JTRROUTER

These examples may be found in the folder `<SUMO_DIST>/data/examples/router`. They shall show how trips and flows may be used to define vehicles.

"trips2routes", "trips2routes_repetition", "flows2routes": Different Definition Types for the Same

All these three examples generate 100 vehicles which all have the same route. In "trips2routes" each vehicle has an own route. They all are the same, only the vehicles' ids differ. "trips2routes_repetition" generates one vehicle but which is duplicated within the simulation 100 times. "flows2routes" generates 100 vehicles and routes using a flow definition. The vehicles differ from those generated by "trips2routes" only by their ids.

Location: `<SUMO_DIST>/data/examples/router`

"flows2routes" vs. "flows2routes_*X*s_interval" vs. "flows2routes_*X*s_interval_ext": Spreading vehicles over an interval

These examples show the usage of the interval length in flows. While in "flows2routes" all vehicles are emitted at the same time, in "flows2routes_100s_interval" the departure times are spread over 100s and in "flows2routes_200s_interval" over 200s. The example ending with "_ext" show how the interval may be defined using an enclosing element.

Location: `<SUMO_DIST>/data/examples/router`

EXTENDED: Examples for using additional SUMO-structures

These examples may be found in the folder `<SUMO_DIST>/data/examples/extended`. They show how simulations may be equipped with additional structures such as variable speed signs, bus routes etc..

"busses1" vs. "3busses1": Examples for Bus Stops

"busses1" shows how bus stops are defined and a bus is forced to stop at these. "3busses1" is almost the same, but three busses are moving around. "3busses1" shows also that the length of bus stops determines how many busses may stop here.

Location: `<SUMO_DIST>/data/examples/extended`

"vehicle_stops": Defining Stop Positions for Vehicles

"vehicle_stops" shows how a vehicle can be forced to stop at a certain postion.

Location: `<SUMO_DIST>/data/examples/extended`