SUMO User Documentation

Introduction

TrafficSimulations

Traffic Simulation Classes

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 ChowdhurySantenSchadschneider2000). Within SUMO, the microscopic model developed by Stefan KrauÃ is used (see Krauss1998_1, Krauss1998_2), extended by some further assumptions. *Mesoscopic* simulations are located at the boundary between microscopic and macroscopic simulations. Herein, vehicle movement is mostly simulated using queue approaches and single vehicles are moved between such queues. *Sub-microscopic* models regard single vehicles like microscopic, but extend them by dividing them into further substructures, which describe the engine's rotation speed in relation to the vehicle's speed or the driver's preferred gear switching actions, for instance. This allows more detailed computations compared to simple microscopic simulations. However, sub-microscopic models require longer computation times. This restrains the size of the networks to be simulated.

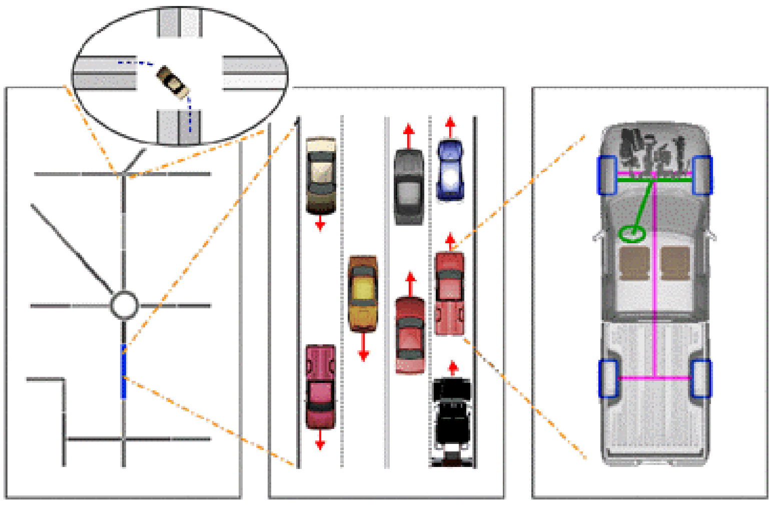


Figure: The different simulation granularities; from left to right: macroscopic, microscopic, sub-microscopic (within the circle: mesoscopic)

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

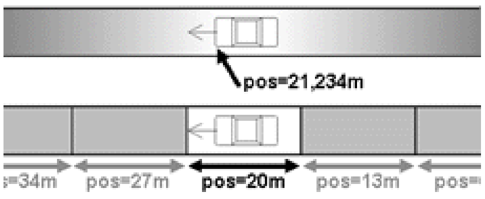


Figure: The difference between a space-continuous (top) and a space-discrete (bottom) simulation

Almost every simulation package uses its 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.

User Assignment

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 *user assignment*. For solving this, several approaches are available and SUMO uses the *dynamic user assignment* (DUA) approach developed by Christian Gawron (see Gawron1998_1).

SumoAtAGlance

SUMO at a Glance

The development of "**S**imulation of **U**rban **MO**bility", or "SUMO" for short, started in the year 2000. The major reason for the development of an open source, microscopic road traffic simulation was to support the traffic research community with a tool into which own algorithms can be implemented and evaluated with, without the need to regard all the artifacts needed to obtain a complete traffic simulation, such as implementing and/or setting up methods for dealing with road networks, demand, and traffic controls. By supplying such a tool, the DLR wanted to i) make the implemented algorithms more comparable, as a common architecture and model base is used, and ii) gain additional help from other contributors.

Since 2001, with the first running version, SUMO has been used within a large number of projects done within the DLR. The main application was to implement and evaluate traffic management methods, such as new traffic light systems or new traffic guidance approaches. Additionally, SUMO was used for short-term (30min) traffic forecast during large events with many participants, and was used for evaluating traffic surveillance using GSM networks.

Since 2002, SUMO is also in use at other institutions. Here, the major interest seems to be the evaluation of vehicle-to-vehicle and vehicle-to-infrastructure communication. Two major third-party projects should be mentioned in this context, the first, TraCI, is an extension of SUMO by the possibility to communicate with external applications, done at the University of LÃ¼beck by Axel Wegener. The second project with a high impact is "TraNS", a direct coupling between SUMO and the network simulator ns2 which uses TraCI for communication and that was set up by Michal Piorkowski and Maxim Raya at the EPFL Lausanne.

Features

- Complete workflow (network and routes import, DUA, simulation)
- Simulation
    - Collision free vehicle movement
    - Different vehicle types
    - Multi-lane streets with lane changing
    - Junction-based right-of-way rules
    - Hierarchy of junction types
    - A fast openGL graphical user interface
    - Manages networks with several 10.000 edges (streets)
    - Fast execution speed (up to 100.000 vehicle updates/s on a 1GHz machine)
    - Interoperability with other application on run time using TraCI
    - Network-wide, edge-based, vehicle-based, and detector-based outputs
- Network
    - Many network formats (VISUM, Vissim, Shapefiles, OSM, Tiger, RoboCup, XML-Descriptions) may be imported
    - Missing values are determined via heuristics
- Routing
    - Microscopic routes - each vehicle has an own one
    - Dynamic User Assignment
- High portability
    - Only standard c++ and portable libraries are used
    - Packages for Windows main Linux distributions exist
- High interoperability through usage of XML-data only

Included Applications

SUMO is not only the name of the simulation application, but also the name of the complete software package which includes several applications needed for preparing the simulation. The package includes:

Application Name	Short Description
SUMO	The microscopic simulation with no visualization; command line application
GUISIM	The microscopic simulation with a graphical user interface
NETCONVERT	Network importer and generator; reads road networks from different formats and converts them into the SUMO-format
NETGEN	Generates abstract networks for the SUMO-simulation
DUAROUTER	Computes fastest routes through the network, importing different types of demand description. Performs the DUA
JTRROUTER	Computes routes using junction turning percentages
DFROUTER	Computes routes from induction loop measurements
OD2TRIPS	Decomposes O/D-matrices into single vehicle trips
POLYCONVERT	Imports points of interest and polygons from different formats and translates them into a description that may be visualized by GUISIM
AdditionalTools	There are some tasks for which writing a large application is not necessary. Several solutions for different problems may be covered by these tools.

SoftwareDesignCriteria

Software design criteria

Two major 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 that does everything. Also, it 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.

Basic Usage

Notation

The documentation within this wiki uses coloring to differ between different type of information. Below, the meaning of colors is described.

Command Line

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
```

Application Options

Command line option names are normally coloured this way. Their values <*LIKE THIS*>.

XML Examples

XML-elements and attributes are shown are coloured like this. Their values, if variable, <*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>
```

Further Schemes

Brackets '[' and ']' indicate that the enclosed information is optional. Brackets '<' and '>' indicate a variable - insert your own value in here.

<*SUMO*> is the path you have saved your SUMO-package into.

DataTypes

Used Data Types

- <*INT*>: an integer value, may be negative
- <*UINT*>: an unsigned integer value, must be >=0
- <*FLOAT*>: a floating point number
- <*STRING*>: any string, but use ASCII-characters only
- <*ID*>: a string which must not contain the following characters: '#'

Caution:
The list of not allowed characters is incomplete

- <*FILE*> or <*FILENAME*>: the (relative or absolute) path to a file; see also #Referenced File Types
- <*PATH*>: a (a relative or absolute) path (mainly to a folder)
- <*COLOR*>: a triple of floats separated by ',' (<*FLOAT*>,<*FLOAT*>,<*FLOAT*>), which describe the red, green, and blue component, respectively
- <*2D-POSITION*>: two floats separated by ',' (<*FLOAT*>,<*FLOAT*>), which describe the x- and the y-offset, respectively
- <*2D-BOUNDING_BOX*>: four floats separated by ',' (<*FLOAT*>,<*FLOAT*>,<*FLOAT*>,<*FLOAT*>), which describe x-minimum, y-minimum, x-maximum, y-maximum, repsectively
- <*PROJ_DEFINITION*>: a string containing the projection definition as used by proj.4; please note that you have to embed the definition string in quotes

Referenced File Types

- *<NETWORK_FILE>*: a SUMO network file as built by NETGEN or NETCONVERT
- *<ROUTES_FILE>*: a SUMO routes file as built by DUAROUTER or JTRROUTER or by hand

InstallingSumo

You should decide whether you primary want to use the software or if you also want to extend it.

- If you just want to **use** SUMO, simply download and run one of our pre-built packages. Everything needed to run SUMO is already included. The only things you might also want are Python and Perl, which are needed for some of the additional scripts in the **tools**-folder.
    - download and install the binaries

- If you also want to **extend** SUMO, you can download the source distribution and build SUMO. 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. Depending on your operating system, please see either
    - building SUMO under Windows
    - building SUMO under Linux

InstallBinary

Install Binary Distribution

Windows

Download the archive named **sumo-winbin-<*VERSION*>.zip** and extract it into a desired folder using 7Zip, Winzip, or a similar tool.

Within this folder, you will find a folder named "**bin**". Here, you can find the executables (programs). You may double click on GUISIM and take a look at the examples located in **data/examples**. All other applications (DUAROUTER, DFROUTER, etc.) have to be run from the command line.

Starting with release 0.9.7 SUMO is build using Microsoft Visual Studio 2005 SP1 which results in the need for the installation of the Microsoft Visual C++ 2005 SP1 Redistributable Package (x86) in certain configurations. If you are not able to start any SUMO application, try installing this package first.

Linux

Precompiled binaries for different distributions can be found at these repositories for binary Linux versions. In the case your system is not listed here, you have to build SUMO from sources.

MacOS

You have to build SUMO from sources under this OS. If you are successfull, drop us a screen shot.

WindowsBuild

This document describes how to build Sumo under Windows using only freely available (this does **not** mean "open source") tools. Instructions on how to build SUMO on Windows using an Open Source toolchain are included in our building on Linux pages.

Please read the whole document before you start, there are some detours and shortcuts included.

If you do not need the GUI, you can skip everything concerning Fox. If you don't need precise geodata conversion, you can skip everything concerning GDAL, Proj and FWTools.

- Download Visual C++ 2005 Express Edition (or a newer one) and the Platform SDK and install them. Be sure to configure Visual Studio correctly to find the platform includes and libs.
- Download Python and install it. It is used to dynamically include the SVN revision into compilation. (The build will work without it, but you may see nasty warnings and even editor windows popping up.)

Python 3 won't work, Python 2.6 may work but has not been tested yet.

- Download the Xerces-C prebuilt binaries for your Visual Studio version (for VS 2005 you need the *windows-vc-8.0.zip), the Fox sources (Version 1.6.36 is confirmed to work) and either the FWTools (Version 1.1.3 should work, 2.4.7 mysteriously fails) or PROJ and GDAL sources separately.
- Note on installation paths: MSVC seems to have difficulties with include and library paths containing spaces (for instance `C:\Program Files`). Thus try to avoid installing any of the following libraries in such paths.

Everything described will only work with current SVN or with a release later than 0.9.5. Please note that for compiling from svn either Python has to be installed on your system (which is a good idea, because also many Sumo tools are python scripts) or you have to undefine HAVE_VERSION_H in src/windows_config.h.

Libraries

Xerces-C

Up to (and including) sumo 0.10.3 (precisely up to svn revision 6993) sumo used xerces-c 2.8 by default, now it uses xerces-c 3.0.1. The changes needed to compile with a different xerces version should be limited to changing src/windows_config.h, build/msvc8/Win32.vsprops and / or build/msvc8/x64.vsprops.

- You just need to install the binary Xerces 3.0.1 distribution or its 64bit version for the 64bit build.
- Create a environment variable XERCES pointing to your xerces-c directory, e.g.

```
set XERCES=D:\libs\xerces-c-3.0.1-x86-windows-vc-8.0.
```

- Copy dll-files `%XERCES%\bin\xerces-c_3_0.dll` and `%XERCES%\bin\xerces-c_3_0D.dll` to some directory which is in your PATH (last resort: into windows' system32 directory).
- For the 64bit build the name of the variable is XERCES_64

Fox

- If you don't need a GUI, you can skip this section.
- Up to (and including) sumo 0.10.3 (precisely up to svn revision 7025) sumo used fox 1.4, now it uses fox 1.6. The building instructions are the same just replace 1.6 by 1.4 (and FOX16 by FOX14) whereever you need it.
- Go to the fox directory and open the VC project e.g. D:\libs\fox-1.6.36\windows\vcpp\win32.dsw.
- Confirm the conversion to VC 8.0 and build the foxdll project as release and debug (if you think you might wish to use the Visual Studio debugger) version. If you want to build for the 64bit platform you need to add a new configuration to the foxdll project using the Configuration Manager.
- You might get approximately 240 warnings and one error, which can be ignored.
- Errors on not finding windows.h mean the SDK was not installed properly.
- Create a environment variable FOX16 pointing to your fox directory, e.g.

```
set FOX16=D:\libs\fox-1.6.36.
```

- Copy dll-files `%FOX16%\libs\fox-1.6.36\lib\FOXDLL-1.6.dll` and `%FOX16%\lib\FOXDLLD-1.6.dll` to some directory which is in your PATH (last resort: into windows' system32 directory).

PROJ and GDAL

If you don't need transformation of geocoordinates you can disable proj and gdal in src/windows_config.h, build/msvc8/Win32.vsprops and / or build/msvc8/x64.vsprops and skip this section. Otherwise you have the choice between downloading the binary FWTools distribution or compiling for yourself. Building from source should only be necessary if you want a 64bit build or if you want to ship a copy of the sumo executable(s) without the need to ship the FWTools as well.

FWTools

- Just execute the binary and select a target directory.
- Create a environment variable PROJ_GDAL pointing to that directory, e.g.

```
set PROJ_GDAL=D:\libs\FWTools1.1.3.
```

- Add `%PROJ_GDAL%\bin` to your PATH (or copy the contents to some directory in your PATH).

Building from source

If you want to build for some reason PROJ and GDAL yourself, please follow the relevant build instructions for PROJ and for GDAL. You should install in a common directory then and let the PROJ_GDAL variable point to it as above. You should also copy proj.dll and gdal*.dll to a directory in your PATH.

If you do a 64bit build, the name of the environment variable needs to be PROJ_GDAL_64.

Sumo

Configuration

- If you installed all libraries and defined the environment variables correctly there is no need for further configuration and you can skip to the build section.
- The Visual Studio build is configured using .vsprops files in the build/msvc8 subdirectory. If you change some setting which should apply to all subprojects, be sure to edit those files (either with a text editor or the property manager of Visual Studio) and not the project configuration (.vcproj).
- If you do not like to define the places of the includes and libraries via environment variables you can enter the location directly into x64.vsprops or Win32.vsprops (or both, depending on your target platforms). You should also disable PROJ and GDAL in those files (if you don't need them) by setting the value for the appropriate "LIB" Usermacro to the empty string.

Build

- Open the project sumo\build\msvc8\prj.sln and build the configurations you need.
- The x64 build is still in experimental stage. It is configured not to use Proj, GDAL and Fox and therefore does not build the guisim.

Troubleshooting

Linker reports something similar to "LINK: fatal error LNK1104: cannot open file 'C:\Program.obj'"

You probably have installed a library to a path containing white spaces in its name. In such a case, the according environment variable should be embedded in quotes (").
Example: set FOX="D:\my libs\fox-1.4.29".

Failure on pre-build event

If Visual Studio reports a failed pre-build event you can safely ignore this, unless you are building from the subversion repository. In this case you should probably install Python.

LinuxBuild

This document describes how to install SUMO on Linux from sources. If you don't want to **extend** SUMO, but merely **use** it, you might want to download one of our pre-built binary packages instead.

To be able to run SUMO on Linux, just follow these four steps:

1. Install all of the required tools and libraries
2. Get the source code
3. Build the SUMO binaries
4. Install the SUMO binaries to another path (optional)

Each of these steps is outlined below.

Installing required tools and libraries

This document contains instructions for two basic ways of installing tools and libraries.

- Either install them manually for the current user (no root privileges required)
- Or let your distribution install them system-wide (root privileges required)

You can follow either way.

Manuall install

This document describes how to build and install (higher level) libraries SUMO depends on from scratch under Linux. That is, there is no need for package management (RPM or such) or root access. All you need is a working gcc (version 3.3 and up should suffice) and probably some basic libraries Fox depends on.

It has been tested successfully on several SuSE Linux installations and with a recent cygwin. Everything which needs extra work on cygwin is described in the cygwin section.

- First check whether your Linux distribution comes with a xerces-c package and install it together with the header files (for SuSE users this is the -dev package). Version 3.0 is used for the current SVN version. You should also check whether Proj, GDAL and Fox already come with your distribution and (if so) install them together with the header files (devel packages). Every package installed this way you will not need to build by yourself.
- Second download the sources of SUMO, Proj, GDAL, Fox (if you want a GUI), and Xerces-C (if it does not come with your distribution).
- If you are building a fresh repository checkout and don't need precise geocoordinates you can leave out Proj and GDAL.

The tools

We use the GNU autotools for generating configure and Makefiles for Unix-like environments. Documentation can be found in the

- Gnu autoconf pages,
- Manual for autoconf,
- Manual for automake,
- Manual for libtool, and
- the Autobook.

The Libraries

- Build and install the libraries (if you don't have root access choose a different target for the libs and includes such as $HOME). For GDAL, Fox and Proj this is more or less straightforward:

```
tar xzf fox-1.6.36.tar.gz
cd fox-1.6.36
./configure --prefix=$HOME && make install
cd ..
tar xzf gdal-1.5.1.tar.gz
cd gdal-1.5.1
```

```
./configure --prefix=$HOME && make install
cd ..
tar xzf proj-4.6.0.tar.gz
cd proj-4.6.0
./configure --prefix=$HOME && make install
```

- Note: The make install for gdal may fail due to the Python bindings which it wants to install into some only-root-writable directory. You can safely ignore this.
- It is strongly recommended to use a user writable installation directory and do "make install" instead of trying to specify directories inside the directory trees of gdal, proj and fox when building Sumo later on.
- Note: It seems like some distributions of FOX are built with disabled openGl-support. If you get unresolved references to methods such as "glColor...", "glVertex3f...", etc. during compilation of guisim you have to enable openGL-support before compiling the FOX-library using "./configure --with-opengl=yes --prefix=$HOME && make install"; Still, this is the default normally.

- In contrast to the other libraries, it is very likely that Xerces-C also comes with your Linux distribution (at least it does with a recent SUSE). If so, please try first install it from there and do not forget to install the developer package as well.
- The Xerces build is somewhat more involved:

```
tar xzf xerces-c-current.tar.gz
export XERCESCROOT=${HOME}/xerces-c-src_3_0_1
cd $XERCESCROOT/src/xercesc
autoconf
./runConfigure -plinux -cgcc -xg++ -minmem -nsocket -tnative -rpthread -P$HOME
make
make install
```

Concluding Remarks

After installing all of the required libraries and tools in this fashion, note that, when building SUMO, the following parameters will be required for the "./configure" run:

```
./configure --with-fox=$HOME --with-proj-gdal=$HOME --with-xerces=$HOME
```

Distribution-specific instructions

If you want to build and optionally install SUMO on Ubuntu systems, you can either use this bash script or follow the instructions below.

Building on Ubuntu 8.04 "Hardy Heron"

On Ubuntu 8.10 "Intrepid Ibex", the following packages are needed to build SUMO:

```
sudo aptitude install libtool libgdal1-dev proj libxerces27-dev
```

For building the SUMO GUI, the following additional packages are needed:

```
sudo aptitude install libfox1.4-dev libgl1-mesa-dev libglu1-mesa-dev
```

Ubuntu 8.04 does not ship with **libgdal.so**, but only with **libgdal1.4.0.so.1**, so you either need to create a symlink **/usr/lib/libgdal.so** -> **/usr/lib/libgdal1.4.0.so.1**

```
sudo ln -s /usr/lib/libgdal1.4.0.so.1 /usr/lib/libgdal.so
```

or if you are using the SVN version, you can instead patch the **configure.ac** file to look for **libgdal1.4.0.so.1** instead by changing **LIB_GDAL="gdal"** to **LIB_GDAL="gdal1.4.0"** and running **make -f Makefile.cvs** to re-generate the **configure** script.

Note that, when building SUMO, the following parameters will be required for the "./configure" run:

```
./configure --with-fox-includes=/usr/include/fox-1.4
 --with-gdal-includes=/usr/include/gdal --with-proj-libraries=/usr
 --with-gdal-libraries=/usr --with-proj-gdal
```

Ubuntu 8.10 "Intrepid Ibex"

On Ubuntu 8.10 "Intrepid Ibex", the following packages are needed to build SUMO:

```
sudo aptitude install libtool libgdal1-dev proj libxerces-c2-dev
```

For building the SUMO GUI, the following additional packages are needed:

```
sudo aptitude install libfox1.4-dev libgl1-mesa-dev libglu1-mesa-dev
```

Ubuntu 8.10 does not ship with **libgdal.so**, but only with **libgdal1.5.0.so**. The easiest way to fix this is to run

```
cd /usr/lib; sudo ln -s libgdal1.5.0.so libgdal.so
```

Instead, you might want to patch the configure script, e.g. if you are using the SVN version, you can patch the **configure.ac** file to look for **libgdal1.5.0.so** instead by changing **LIB_GDAL="gdal"** to **LIB_GDAL="gdal1.5.0"** and running **make -f Makefile.cvs** to re-generate the **configure** script.

Note that, when building SUMO, the following parameters will be required for the "./configure" run:

```
./configure --with-fox-includes=/usr/include/fox-1.4
 --with-gdal-includes=/usr/include/gdal --with-proj-libraries=/usr
 --with-gdal-libraries=/usr --with-proj-gdal
```

Ubuntu 9.04 "Jaunty Jackalope"

For Ubuntu 9.04 "Jaunty Jackalope", follow the build instructions above for Ubuntu 8.10.

Ubuntu 9.10 "Karmic Koala"

For Ubuntu 9.10 follow the instructions for Ubuntu 8.10 with one exception:

Instead of *libfox1.4-dev* install *libfox-1.6-dev* (don't miss the additional dash '-'):

```
sudo aptitude install libfox-1.6-dev libgl1-mesa-dev libglu1-mesa-dev
```

Note that, when building SUMO, the following parameters will be required for the "./configure" run:

```
./configure --with-fox-includes=/usr/include/fox-1.6
 --with-gdal-includes=/usr/include/gdal --with-proj-libraries=/usr
 --with-gdal-libraries=/usr --with-proj-gdal
```

Ubuntu 10.04 "Lucid Lynx"

Lucid comes with version 1.6.0 of libgdal. So to build and install on lucid take above instructions for karmic (9.10) which reference to intrepid (8.10) and replace "gdal1.5.0" with "gdal1.6.0" where applicable.

Cygwin

Cygwin comes with a quite old xerces-c (version 2.5). After installing it you need to make a symbolic link in /usr/lib such that the linker finds the correct library:

```
cd /usr/lib; ln -s /usr/lib/libxerces-c25.dll.a /usr/lib/libxerces-c.dll.a
```

Now everything (but the GUI) should build fine.

Getting the source code

release version

```
tar xzf sumo-src-<version>.tar.gz
cd sumo-<version>/
```

subversion checkout

The process is essentially the same as building a release except that one needs the GNU autotools (which were also necessary when building xerces). The call to the autotools is hidden in Makefile.cvs.

The following commands are needed:

```
svn co https://sumo.svn.sourceforge.net/svnroot/sumo/trunk/sumo
make -f Makefile.cvs
```

Building the SUMO binaries

```
./configure [options]
make
```

You probably need to tell the configure script where you installed all of the required libraries (e.g. **--with-fox=...**). Please see the above instructions on installing required tools and libraries to find out how to do that.

Other common options to ./configure include **--prefix=$HOME** (so installing SUMO means copying the files somewhere in your home directory) and **--enable-debug** (to build a version of SUMO that's easier to debug).

For additional options please see

```
./configure --help
```

Additional notes for Cygwin users

At the moment GUI building is still troublesome. It depends whether you want to use the X-Server or native Windows GUI. We tried native Windows GUI only and had to change the installed libFOX-1.4.la such that it contains

```
dependency_libs=' -lgdi32 -lglaux -ldl -lcomctl32 -lwsock32 -lwinspool -lmpr
-lpthread -lpng /usr/lib/libtiff.la /usr/lib/libjpeg.la -lz -lbz2 -lopengl32 -lglu32'
```

Your mileage may vary.

Installing the SUMO binaries

This (optional) step will copy the SUMO binaries to another path, so that you can delete all source and intermediate files afterwards. If you do not want (or need) to do that, you can simply skip this step and run SUMO from the source subfolder (src/sumo-guisim and src/sumo).

If you want to install the SUMO binaries, run

```
make install
```

Troubleshooting

Unresolved references to openGL-functions

Build reports unresolved references to openGL-functions, saying things such as

```
"./utils/glutils/libglutils.a(GLHelper.o): In function
`GLHelper::drawOutlineCircle(float, float, int, float, float)':
/home/smartie/sumo-0.9.5/src/utils/glutils/GLHelper.cpp:352:
undefined reference to `glEnd'"
```

SUMO needs FOX-toolkit to be build with openGL-support enabled. Do this by compiling FOX-toolkit as following:

```
tar xzf fox-1.4.34.tar.gz
cd fox-1.4.34
./configure --with-opengl=yes --prefix=$HOME && make install
```

Further comment from Michael Behrisch ([sumo-user], 4.4.2007): *Probably there is something wrong with your OpenGL installation. Make sure you have the libGL.so and libGLU.so which are most likely symbolic links to libGL.so.1.2 or something like this. They should appear in /usr/lib and case does matter (so "libgl.so" won't do).*

Trouble with SVN r4182

Just wanted to drop a short note that I was only able to build r4182 after removing l.472 from `src/Makefile`:

```
@@ -469,7 +469,6 @@
 $(sumo_LDADD) \
 ./utils/glutils/libglutils.a \
 ./gui/GUIManipulator.o \
-./utils/gui/drawer/GUICompleteSchemeStorage.o \
 -l$(LIB_FOX)
```

Problems with the socket subsystem

Problem:

```
recv ./foreign/tcpip/libtcpip.a(socket.o)  (symbol belongs to implicit dependency /usr/lib/libsoc
```

Solution: http://lists.danga.com/pipermail/memcached/2005-September/001611.html

MacBuild

Basically the build steps are the same as when building under Linux. The most easy way to get the needed libraries is to install MacPorts under /opt/local/, including fox, gdal, and proj. This also includes the openGL-headers. Then, SUMO can be built using:

```
export CPPFLAGS="$CPPFLAGS -I/opt/local/include"
export LDFLAGS="$LDFLAGS -L/opt/local/lib"
make -f Makefile.cvs
./configure --with-fox=/opt/local --with-proj-gdal=/opt/local \
  --with-xerces=/opt/local --prefix=/opt/sumo
```

Thanks to Katharina Marzok for supplying this information.

DocumentationBuild

We provide a make target **make doc** for building documentation. It requires xsltproc and fop which come with recent Linux distributions. Unfortunately fop requires a graphics library, like JAI or JIMI. We use JAI, which needs to be downloaded and installed by copying jai_core.jar and jai_codec.jar to {fop-install-dir}/lib. At least my version of fop (0.20.5 shipped with openSUSE 10.2) gives lots of errors but the resulting pdf looks OK.

Update: Fop 0.93 together with Java 1.6 as installed from openSUSE 10.3 seems to work fine without additional libraries. Java 1.5 does not suffice.

There is no support for building docs under Windows but the Windows binary package already contains the pdf docs.

Using the Command Line Applications

Most of the applications from the SUMO package are command line tools. Currently, only GUISIM is not. Because we have noted that using command line applications is not common to some, especially for those who are not information scientists, a short introduction is given below. Please note that this is not a complete guide to command line usage, just a short introduction scoping at SUMO usage.

Basics

Starting a Command Line

At first you have to open the command line. On Windows, you have to start "cmd.exe" for this (Start->Execute->cmd.exe). A black window should appear. This is your command line. When using Linux, you have to start a terminal (like xterm).

Using SUMO Applications from the Command Line

SUMO applications are plain executables. You just start them by typing their name from the command line; for example NETGEN is called by

```
netgen.exe
```

under Windows and by

```
netgen
```

under Linux.

This simply starts the application (NETGEN in this case). As no parameter have been given, the application does not know what to do and prints only an information about itself:

```
SUMO netgen Version <VERSION>
 (c) DLR 2001-2010; http://sumo.sourceforge.net
 Use --help to get the list of options.
```

Options

Each application has a set of options which define which files shall be process, or generated, or which define the application's behaviour. Normally, an application needs at least two parameter - an input file and an output file - but almost always more parameter are needed. Each application's options are described within the application's description. In the following, it is described how options are set.

Setting Options on the Command Line

There are two kinds of options: boolean options (which do not require an argument and are set to true if (and only if) the option is present) and options with an argument. Setting an option with an argument on the command line consists of two parts - the option name, and the option's value. For example, if one wants the simulation to load a certain road network, "mynet.net.xml", the following must be written:

```
--net mynet.net.xml
```

The '--' in front indicates that the option's long name is following ("net") in this case. After a whitespace the option's value must be given. It is also possible to use a '=' instead of the whitespace:

```
--net=mynet.net.xml
```

Some often used options have can be abbreviated, the abbreviation for the --net-option is -n. The following has the same effect as the two examples above:

```
-n mynet.net.xml
```

Please not that an abbreviation is indicated using a single '-'.

Option Value Types

The SUMO applications know what kind of a value they expect to be set. For example, NETGEN allows you to set the default number of lanes, which of course must be an integer value. In the case, a string or something else is given, this is recognized and the application answers with an error message on startup.

A special case of value types are lists, for example the list of additional files to load into a simulation. When giving more than a single file, the files must be divided using ','.

Configuration Files

Because the list of options may get very long, configuration files were introduced. You can set up a configuration file which contains all the parameter you want to start the application with and have to start the application with only this configuration file given.

A configuration file is an XML-file that has a root element named configuration. The options are written as element names, with the wanted value being stored in attribute value (or v); the option --net-file *test.net.xml* given on the command line would become <net-file value="test.net.xml"/> within the configuration file. For the boolean options the value should be either "true", "on", "yes", "1", or "x" for the activation and "false", "off", "no", or "0" for deactivating the option (case does not matter here).

For the example above, the configuration file (let's save it under "test.sumo.cfg", see below) would look like:

```
<configuration>
    <n v="test.net.xml"/>
    <r v="test.rou.xml"/>
    <a v="test.add.xml"/>
</configuration>
```

A more verbose but equivalent version would look like:

```
<configuration>
    <input>
        <net-file value="test.net.xml"/>
        <route-files value="test.rou.xml"/>
        <additional-files value="test.add.xml"/>
    </input>
</configuration>
```

The section input given above serves only documentation purposes and has no functional meaning.

The according SUMO execution call - working with both configuration versions - would be:

```
sumo.exe -c test.sumo.cfg
```

This means that instead of the parameters, we only give the name of the configuration file using the option --configuration-file *<FILE>* or -c *<FILE>*.

Naming Conventions for Configuration Files

Dependencing on the targetted application, the configuration files' have different extensions. It is highly recommended to follow this convention. For using simulation configurations with GUISIM this is even required - GUISIM can only read simulation configurations named "*.sumo.cfg". The extensions are:

- *.sumo.cfg: Configuration file for SUMO and GUISIM
- *.netc.cfg: Configuration file for NETCONVERT
- *.netg.cfg: Configuration file for NETGEN
- *.rou.cfg: Configuration file for DUAROUTER
- *.jtr.cfg: Configuration file for JTRROUTER
- *.df.cfg: Configuration file for DFROUTER
- *.od2t.cfg: Configuration file for OD2TRIPS

(See also the list of known extensions.)

Configuration Files vs. Command Line Parameter

In addition to a configuration file, further command line parameter can also be given on the command line. (This is not true for the simulation gui, which needs all parameters inside the configuration file.) If a parameter is set both within the named configuration file as well as given on the command line, the value given on the command line is used (overwrites the one within the configuration file).

Generating Configuration File Templates

The applications of the SUMO package allow you to generate configuration file templates. It is possible to save an empty configuration - a configuration template. This can be done using the --save-template *<FILE>*. In this case, the configuration will only contain the parameters, filled with their default values. If further information on the parameters is wanted, one can also pass the option --save-template.commented. Then, some further comments on each parameter are generated.

Saving the current Configuration into a File

Also, it is possible to save a configuration file which contains the currently set values. An application can be forced to do this using the option --save-configuration *<FILE>*.

Common Options

The applications from the SUMO suite share several options. They are given in the following.

Reporting Options

Option	Description

--verbose	The application is more verbose in reporting what it is doing
--suppress-warnings	Warnings will not be written
--print-options	The options are printed giving the currently set values
--help -?	The application's help screen is printed
--log-file *<FILE>*	All outputs are written onto the console and into the given file

Random Number Options

These options configure how the seed of the random number generator is determined. The same seed leads to the same sequence of generated random numbers.

By default the seed is a hard-coded fixed value. So, as long as all configuration settings keep the same the outputs of several simulation runs will be exactly the same. To change this use one of the following options.

Option	Description
--srand *<INT>*	Set a particular seed for the random number generator. By using different values you can have different but still reproducible simulation runs.
--abs-rand	Make SUMO choose a seed. If available the seed will be based on output of /dev/urandom otherwise the seed will be derived from current system time. This option has **precedence** over option --srand *<INT>*.

Attention: The precedence of --abs-rand over --srand *<INT>* means that it is **impossible** to set --abs-rand in a configuration file and overwrite it by --srand *<INT>* in the command line if needed. There might be a solution to this in the future.

Developer Ressources

There is a documentation on the options sub system for developers available.

BestPractices

Best Practices

- Set an evironment variable to the folder SUMO is located in
- Use configuration files

Tutorials and First Steps

SettingSimulationUp

Setting up a Simulation for SUMO

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 needed values that are missing within the imported data. 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 many traffic simulations use a statistical distribution which is laid over the network, each vehicle within SUMO knows its route, where "route" is a list of edges to pass.

You basically have to perform the following steps in order to implement a simulation scenario:

1. Build your network

> This can be either done by a) generating an abstract network using NETGEN, b) setting up an own description in XML and importing it using NETCONVERT or by c) importing an existing road network using NETCONVERT; see building the networks for further information

2. Build the demand

> This can be either done by a) describing explicit vehicle routes, b) using flows and turning percentages only, c) generating random routes, d) importing OD-matrices, or e) importing existing routes; see building the demand for further information

3a. If needed, compute the dynamic user assignment

3b. Calibrate the simulation using given measures

4. Perform the simulation to get your desired output;

> See performing the simulation and available simulation outputs

This process is also visualised within the next figure.

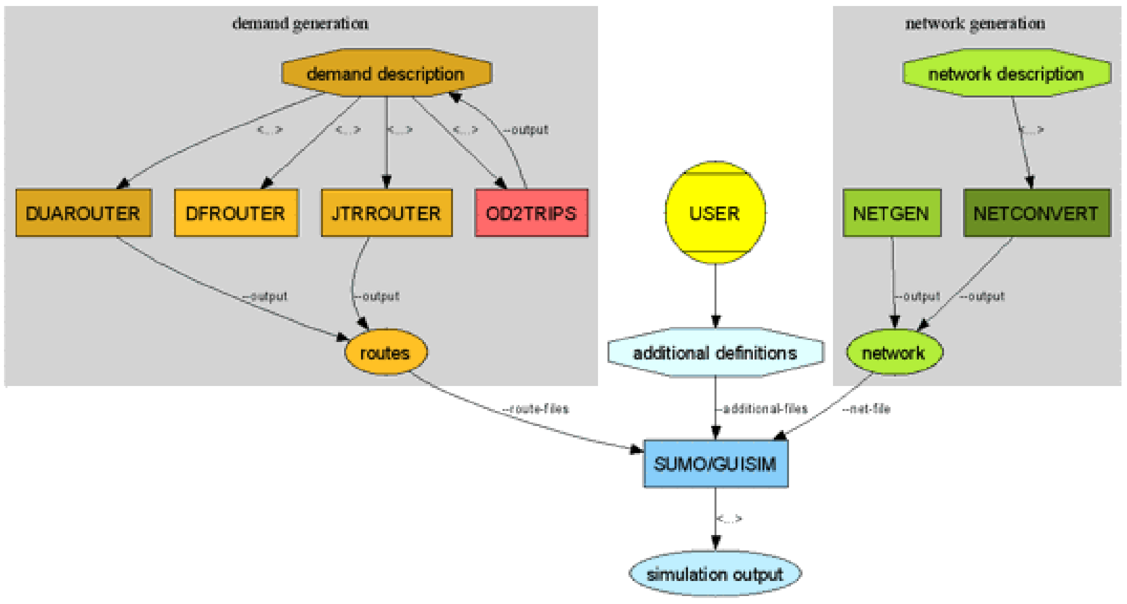


Figure 1.3. Process of simulation with SUMO (rounded: definite data types; boxes: applications; octagons: abstract data types)

Best Practice

Choosing a proper scenario depends on the done investigation. Abstract, generated networks may be the best solution if one wants to keep the results easily evaluable. Abstract networks have for example been used for evaluating the vulnerability of C2C-networks by Clemens Honomichl.

If one wants to investigate real life traffic, he should try to reuse the data he has. If no data is available, one could try to use the free scenarios available within SUMO.

Tutorials

Beginner Tutorials

This tutorials assume very basic computer skills. If you run into any questions please consult SumoBasicComputerSkills.

- HelloSumoTutorial - the most simple net and a single car set up "by hand"
- QuickStartTutorial - a more complex tutorial; first steps in SUMO
- TraCI4TrafficLightsTutorial - an example how to connect an external application to SUMO via TraCI

File:Example.jpg

Network Building

SUMO Road Networks

SUMO Road Networks

SUMO uses an own road network description. **Although being readable (xml) by human beings, SUMO's road networks are not meant to be edited by hand.** Rather, they should be built by either converting an existing map using NETCONVERT or by using NETGEN to generate geometrically simple, abstract road maps.

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. But networks generated using NETGEN are of course useless as soon as you want to simulate a real-world network.

Using NETCONVERT one can import road networks from several sources, among them VISUM, shape files, and OSM databases. This is of course the tool to use for setting up real-world networks.

Still, most examples within the data-section were written as XML files by hand and converted using NETCONVERT for several reasons. At first, the examples are small enough and one may steer the output better than when using NETGEN. Furthermore, defining own networks using XML-data is more flexible.

Network Format

At a coarse scale, SUMO road networks are directed graphs. "Nodes" represent intersections/junctions, and "edges" roads/streets.

Edges and Lanes

The attributes of an edge are:

- id: The id of the edge
- from: The id of the node it starts at
- to: The id of the node it ends at
- priority: A numerical value showing how important the road is
- function: An abstract edge purpose

For the simulation, only the "function" attribute is of interest. It describes how the edge is used, and whether it is an edge that can be found within the real world or only a helper construct used for assignment, f.e. The following purposes are defined:

- normal: The edge is a plain part of a road network, like a highway or a normal street which connects two roads

- connector: The edge is a macroscopic connector - not a part of the real world road network. Still, within the simulation, no distinction is made between "connector" roads and "normal" nodes. Only GUISIM allows to hide connector edges.
- internal: The edge is a part of an intersection (is located within the intersection). The usage within the simulation differs from the one of normal edges.

Each edge includes the definitions of lanes it consists of. Each lane is defnied by:

- id: The id of the lane
- depart: Information whether vehicles starting on this edge normally use this lane (=1) or not (=0)
- vclasses: The list of vehicle classes allowed/disallowed on this lane
- maxspeed: The maximum velocity allowed on this lane
- length: The length of this lane

Furthermore, a lane contains a shape. Even though the network is shifted to start at (0,0), it is not guaranteed that all of the network's parts have positive coordinates.

Nodes

Some notes:

- It is possible that two different nodes have the same position

Coordinates and alignment

The networks are using cartesian coordinates where the leftmost node is at x=0 and the node being most at the bottom is at y=0. This means that when being imported, NETCONVERT and NETGEN are projecting the network, first, if the original network was not using cartesian coordinates. Then, they move the road network to the origin at (0,0).

This process is documented within the generated network within the element location. You may find here the following attributes:

Name	Type	Description
netOffset	offset (*<2D-POSITION>*)	The offset applied for moving the network to (0,0)
convBoundary	boundary (*<2D-BOUNDING_BOX>*)	The boundary spanning over the nodes of the converted network (the network as now given)
origBoundary	boundary (*<2D-BOUNDING_BOX>*)	The network's initial boundary before projecting and translating it
projParameter	projection parameter	Information how the network was projected (see below)

projParameter may not necessaraly be a proj-string. This is only the case if the network was projected using proj from another coordinate system. The following encodings are currently used:

Encoding	Description
'!'	No projection was applied.
'-'	"Simple" projection was applied
'.'	?
proj-definition (<PROJ_DEFINITION>)	The projection was done using proj, initialising it using these parameter.

For obtaining the original coordinates from the x/y-coordinates SUMO reports, one has to subtract the network offset first. Then, one has to apply an inverse transformation of the initial projection, regarding projParameter. For proj-projected networks, one can use proj's function named pj_inv which performs the inverse transformation. proj has to be initialised with the projection parameters from projParameter, first. For proj-projected networks this would look like:

```
Position2D cartesian = sumoXY(x, y);
projPJ myProjection = pj_init_plus(projParameter.c_str());
cartesian.sub(netOffset);
projUV p;
p.u = cartesian.x();
p.v = cartesian.y();
p = pj_inv(p, myProjection);
p.u *= RAD_TO_DEG;
p.v *= RAD_TO_DEG;
cartesian.set((SUMOReal) p.u, (SUMOReal) p.v);
```

For networks with no projection (projParameter='!'), only the offset must be applied. For networks with "simple" projection (projParameter='-'), a back-projection method is currently not known.

SUMO road networks are meant to be aligned to the north. Of course, it is up to the user how he defines a road network, but when being imported from sources as Open Street Map or shape files, the up-direction is meant to correspond to north.

Further Documentation

- Network Import
    - Importing networks with NETCONVERT
        - Defining own networks using XML and NETCONVERT
        - Importing foreign networks using NETCONVERT
    - NETCONVERT manual
- Generation of abstract road networks
    - Generating abstract networks using NETGEN
    - NETGEN manual

Recent Changes

Caution:
Please read this when moving from 0.10.x (or prior) to versions after 24.07.2009:
Between the 22.07.2009 and 24.07.2009 some major changes have been performed on the network format; they can be summarized as moving all character data to attributes, explicitely:

- lane shapes are no longer stored within the lane's character sections, but within lane's shape attribute
- the network projection/location/boundary description is no longer given using elements net-offset, conv-boundary>, orig-boundary, and orig-proj; instead, a location element was introduced and the values are stored in this element's attributes netOffset, convBoundary, origBoundary, and projParameter, respectively.
- sub-elements key, requestsize, and lanenumber of the row-logic element were moved into row-logic as attributes id, requestSize, and laneNumber, respectively.
- sub-elements key, subkey, and offset of the tl-logic element were moved into tl-logic as attributes id, programID, and offset, respectively. The element phaseno was discarded.
- sub-elements inclanes, intlanes, and shape of the junction element were moved into junction as attributes incLanes, intLanes, and shape, respectively.
- district shapes are no longer stored within the district's character sections, but within district's shape attribute
- polygon shapes are no longer stored within the polygon's character sections, but within polygon's shape attribute

Building Networks from own XML-descriptions

Almost 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 (see also Glossary). Besides defining the nodes and edges, you can also join edge attributes by type and set explicit connections between edges or lanes. We will describe how each of these four file types should look like in the following chapters. Further information on road networks may be also found in the SUMO Road Networks description.

After you have generated the files as described below - you need at least the edges and the nodes-files, using type and/or a connections file is optional - you should run NETCONVERT to build the network. If you only use edges and nodes, stored in "MyEdges.edg.xml" and "MyNodes.nod.xml" respectively, 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 also use connections and types the call is:

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

The options used here, including their abbreviations, are documented on the NETCONVERT manual page.

Maybe your edge definitions are incomplete or buggy. If you still want to import your network, you can try passing --dismiss-loading-errors to NETCONVERT. 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.

The structures of the files described in the following are also available as XML Schema Definitions:

- nodes files: http://sumo.sourceforge.net/xsd/nodes_file.xsd
- edges files: http://sumo.sourceforge.net/xsd/edges_file.xsd

- types files: http://sumo.sourceforge.net/xsd/types_file.xsd
- connections files: http://sumo.sourceforge.net/xsd/connections_file.xsd

Node Descriptions

Within the nodes-files, normally having the extension ".nod.xml" (see Known Extensions), 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:

Attribute Name	Mandatory	Value Type	Description
id	y	id (string)	The name of the node
x	y	float	The x-position of the node on the plane in meters
y	y	float	The y-position of the node on the plane in meters
type		enum ("priority", "traffic_light", "right_before_left")	An optional type for the node

If you leave out the type of the node, it is automatically guessed by NETCONVERT 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.
- right_before_left: Vehicles will let vehicles coming from their right side pass.

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>***/data/examples/netbuild/types/cross_usingtypes/" and "***<SUMO>***/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).

Edge Descriptions

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

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

For each edge, some further attributes should be supplied, being the number of lanes the edge has, the maximum speed allowed on the edge, the length the edge has (in meters). Furthermore, the priority may be defined optionally. All these values - beside the length in fact - may either be given for each edge using according attributes or you can omit them by giving the edge a "type". In this case, you should also write a type-file (see Type Descriptions). 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.

Let's list an edge's attributes again:

Attribute Name	Mandatory	Value Type	Description
id	y	id (string)	The name of the edge
fromnode		referenced node id	The name of a node within the nodes-file the edge shall start at
tonode		referenced node id	The name of a node within the nodes-file the edge shall end at

type	referenced type id	The name of a type within the types-file
nolanes	int	The number of lanes of the edge; must be an integer value
speed	float	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")
priority	int	The priority of the edge
length	float	The length of the edge in meter
shape	List of positions; each position is encoded in x,y (do not separate the numbers with a space!) in meters	The start and end node are omitted from the shape definition; an example: <edge id="e1" fromnode="0" tonode="1" shape="0,0 0,100"/> describes an edge that after starting at node 0, first visits position 0,0 than goes one hundred meters to the right before finally reaching the position of node 1
spread_type	enum ("right", "center")	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"

Missing:
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
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>***/data/examples/netbuild/types/cross_notypes/" the whole edges-file looks like this ("cross3l.edg.xml"):

```
<edges>

   <edge id="1fi" fromnode="1" tonode="m1" priority="2" nolanes="2" speed="11.11"/>
   <edge id="1si" fromnode="m1" tonode="0" priority="3" nolanes="3" speed="13.89"/>
   <edge id="1o" fromnode="0" tonode="1" priority="1" nolanes="1" speed="11.11"/>

   <edge id="2fi" fromnode="2" tonode="m2" priority="2" nolanes="2" speed="11.11"/>
   <edge id="2si" fromnode="m2" tonode="0" priority="3" nolanes="3" speed="13.89"/>
   <edge id="2o" fromnode="0" tonode="2" priority="1" nolanes="1" speed="11.11"/>

   <edge id="3fi" fromnode="3" tonode="m3" priority="2" nolanes="2" speed="11.11"/>
   <edge id="3si" fromnode="m3" tonode="0" priority="3" nolanes="3" speed="13.89"/>
   <edge id="3o" fromnode="0" tonode="3" priority="1" nolanes="1" speed="11.11"/>

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

</edges>
```

Within this example, we have used explicit definitions of edges. An example for using types is described in the chapter Type Descriptions.

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).

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 edge's lanes. Assume you want to allow only busses to use the leftmost lane of edge "2si" from the example above. Simply change this edge's definition into:

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

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

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

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

Attribute Name	Mandatory	Value Type	Description
id	y	int	The enumeration id of the lane (0 is the rightmost lane, <NUMBER_LANES>-1 is the leftmost one)
allow		list of vehicle classes	The list of explicitely allowed vehicle classes
disallow		list of vehicle classes	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" for further information about allowed vehicle classes and their usage.

Road Segment Refining

Normally, each edge has a certain number of lanes; road widenings are defined using consequent edges. Though, in some cases, it is more comfortable to set up a single edge and change the number of its lanes along its length.

This is possible using "split" - a subelement of an XML-edge definition after 02.09.2009. A split may be given as following:

```
... previous definitions ...
   <edge id="2si" fromnode="m2" tonode="0" priority="3" nolanes="3" speed="13.89">
      <split pos="30" lanes="0 1"/>
   <edge>
... further definitions ...
```

What happens here is the following: 30meters from its begin (pos="30") the edge is split, inserting a node named <EDGE_ID>.<POSITION>. Until this node, the edge includes all lanes. After this node, only the lanes given in the lanes attribute are inserted. This yields in two edges which replace the initial one and the second edge contains only two edges.

The most common usage example is the other way round: lanes must be added, not removed along a road. The following snipplet shows the by example:

```
... previous definitions ...
   <edge id="2si" fromnode="m2" tonode="0" priority="3" nolanes="3" speed="13.89">
      <split pos="0" lanes="0 1"/>
      <split pos="30" lanes="0 1 2"/>
   <edge>
... further definitions ...
```

There are two things to note:

- In the road definition, we use the maximum lane number (3)
- We add an "split" at position 0; in this case, no node is built, but the lane number is applied to the edge directly.

The definition of a split uses the following attributes:

Attribute Name	Mandatory	Value Type	Description
pos	y	float	The position along the edge at which the split shall be done (in m); if a negative position is given, the split is inserted counting from the end of the edge.
lanes	y	list of lane ids (ints)	Information which lanes should exist after the split

Type Descriptions

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

The attributes of a type are of course exactly the same as for edges themselves:

Attribute Name	Mandatory	Value Type	Description
id	y	id (string)	The name of the type
nolanes		int	The number of lanes of the edge; must be an integer value
speed		float	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")
priority		int	The priority of the edge
discard		bool	Whether edges of this type shall not be imported;

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>***/data/examples/netbuild/types/cross_notypes/").

Connection Descriptions

"Connections" describe how a node's incoming and outgoing edges are connected - in exactly, which edges may be approached from an incoming edge, and which lanes may be used to approach them.

Though possible, it should be avoided to set more than one connection from a certain incoming edge to a certain outgoing lane. SUMO vehicles are not aware of the conflicts set up this way.

Explicite setting which Edge / Lane is connected to which

Though guessed if not given, definitions of connections between edges or lanes may be manually set up and given to NETCONVERT using connection files. 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, the definition is: <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:

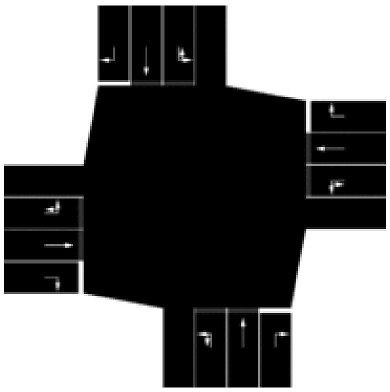
Attribute Name	Mandatory	Value Type	Description
from	y	referenced edge id	The name of the edge the vehicles leave
to		referenced edge id	The name of the edge the vehicles may reach when leaving "from"

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:

Attribute Name	Mandatory	Value Type	Description
from	y	referenced edge id	The name of the edge the vehicles leave
to		referenced edge id	The name of the edge the vehicles may reach when leaving "from"
lane		<*INT*>:<*INT*>	

		the numbers of the connected lanes, separated with ':'; lanes are counter from right to left beginning with 0
pass	bool	if set, vehicles which pass this (lane-2-lane) connection) will not wait

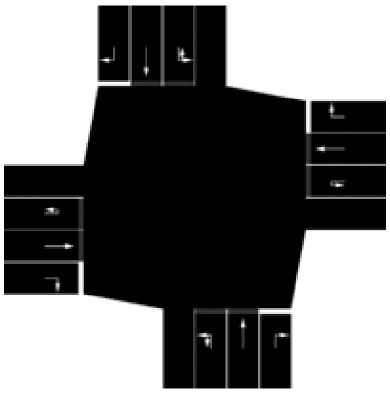
There are two examples within the distribution. Both use the nodes and edges descriptions from the example located in "***<SUMO>***/data/examples/netbuild/types/cross_notypes/". The junction in the center of this example looks like shown within the next figure. We will now call it the "unconstrained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



Unconstrained Network
The example ***<SUMO>***/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>
```

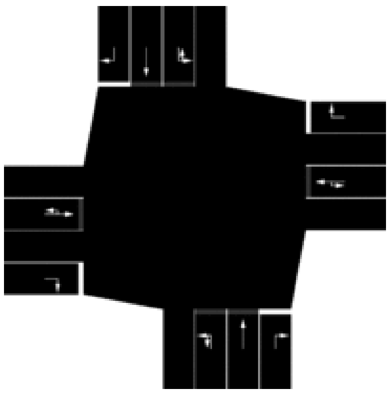
As you may see in the next picture, the horizontal edges within the result network contain no left-moving connections.



Network with explicit edge-2-edge connections
In the second example located in ***<SUMO>***/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:



Network with explicit lane-2-lane connections
Caution:
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.

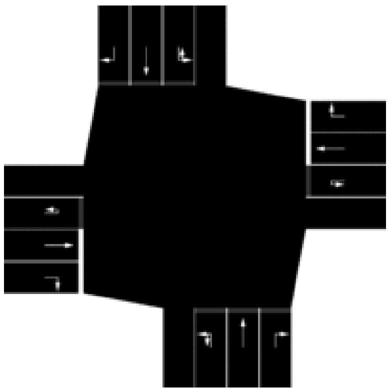
Setting Connection Priorities

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

```
<connections>

   <!-- The next four connection definitions are same as in
        "Network with explicit edge-2-edge connections" -->
   <connection from="1si" to="3o"/>
   <connection from="1si" to="2o"/>

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

   <!-- now, let's prohibit the vertical connections by the horizontal -->
   <!-- prohibit moving right from top to left by straight from right to left -->
   <prohibition prohibitor="2si->1o" prohibited="4si->1o"/>
```

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

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

</connections>
```

As one may see, it was necessary to prohibit all connections from a vertical edge by the 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:



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 "->".

NetworkImportForeign

Importing Networks

NETCONVERT allows to import road networks from different formats. By now, the following formats are supported:

- PTV VISUM (a macroscopic traffic simulation package), see Importing networks, from VISUM
- PTV VISSIM (a microscopic traffic simulation package), see Importing networks from Vissim
- ArcView-data base files, see Importing ArcView networks (shapefiles)
- Elmar Brockfelds unsplitted and splitted NavTeq-data, see Importing Elmar's networks
- TIGER databases, see Importing networks from TIGER
- OpenStreetMap databases, see OpenStreetMapImport

- RoboCup Rescue League folders, see Importing networks from Robocup Simulation League

In most of these cases, NETCONVERT 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). In the case, a VISUM network shall be imported, the following code will convert it into a SUMO-network:

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

The import can be influenced using further NETCONVERT options.

VisumNetworkImport

Importing Networks from VISUM

NETCONVERT can import native VISUM-network files. Their extension is ".net". If you do not have a file with this extension, but a ".ver"-file only, you have to generate the ".net"-file using VISUM by exporting it from the loaded version description (".ver"-file).

The option to load a VISUM ".net"-file into NETCONVERT in order to convert it into a SUMO-network is named --visum *<FILE>*. So, the following call to NETCONVERT imports the road network stored in "my_visum_net.net" and stores the SUMO-network generated from this data into "my_sumo_net.net.xml":

```
netconvert --visum my_visum_net.net -o my_sumo_net.net.xml
```

The following table shows which information is parsed from a given VISUM network.

Information NETCONVERT reads from VISUM networks

Table name	Imported attributes	Description
VSYS	VSysCode (CODE) VSysMode (TYP)	Traffic modes
STRECKENTYP	Nr v0-IV (V0IV) Rang Kap-IV (KAPIV)	Edge types
KNOTEN	Nr XKoord YKoord	Nodes
BEZIRK	Nr NAME (unused later) Proz_Q Proz_Z XKoord YKoord	Districts
STRECKE (STRECKEN)	Nr VonKnot (VonKnotNr) NachKnot (NachKnotNr)	Edges

	Typ (TypNr) Einbahn	
ANBINDUNG	BezNr KnotNr Proz t0-IV Typ Richtung	District connections
ABBIEGEBEZIEHUNG (ABBIEGER)	VonKnot (VonKnotNr) UeberKnot (UeberKnotNr) NachKnot (NachKnotNr) VSysCode (VSYSSET)	Edge Connections
STRECKENPOLY	VonKnot (VonKnotNr) NachKnot (NachKnotNr) INDEX XKoord YKoord	Edge geometries
FAHRSTREIFEN	KNOTNR STRNR FSNR RICHTTYP LAENGE	Lane descriptions
LSA (SIGNALANLAGE)	Nr Umlaufzeit (UMLZEIT) StdZwischenzeit (STDZWZEIT) PhasenBasiert	Traffic lights
KNOTENZULSA (SIGNALANLAGEZUKNOTEN)	KnotNr LsaNr	Nodes->TLS
LSASIGNALGRUPPE (SIGNALGRUPPE)	Nr LsaNr GzStart (GRUENANF) GzEnd (GRUENENDE)	Signal groups
ABBZULSASIGNALGRUPPE (SIGNALGRUPPEZUABBIEGER)	SGNR (SIGNALGRUPPENNR) LsaNr VonKnot / VONSTRNR NachKnot / NACHSTRNR UeberKnot (UeberKnotNr) LsaNr	Edge connections->TLS
LSAPHASE (PHASE)	Nr LsaNr GzStart (GRUENANF) GzEnd (GRUENENDE)	Signal phases
LSASIGNALGRUPPEZULSAPHASE	PsNr LsaNr SGNR	Signal groups->phases
FAHRSTREIFENABBIEGER		

	KNOT (KNOTNR) VONSTR (VONSTRNR) NACHSTR (NACHSTRNR) VONFSNR NACHFSNR	Lane-to-lane descriptions

Well, basically that's all for network import, isn't it? Well, actually not. In the following, the basic possibilities and further advices and tricks for working with these is described.

Note:
"Rang" from "Streckentypen" is used as edge priority information. As streets with a lower "Rang" are normally higher priorised, an edge type's priority is computed as 100-Rang

Lane Number

VISUM does not work with the lane numbers of streets, instead, the streets' capacities are used. This means that under circumstances the informationabout the edges' lane numbers are missing. Still, one can try to obtain the lane number from the given edges' capacities. An approximation is:

```
LANE_NUMBER = MAXIMUM_FLOW / CAPACITY_NORM
```

The value of CAPACITY_NORM is controlled via the option --capacity-norm *<FLOAT>* (default: 1800).

In the case the "*ANZFAHRSTREIFEN*" (lane numbers) field within the net's "*STRECKEN*" (edges)-table is empty, and also the types do not hold any information about the according edges' lane numbers, this conversion is done automatically. Still, we also had VISUM nets in our hands, where a capacity was given, but the lane number field was set to 1 for all edges. In this case, one has to force NETCONVERT to ignore the lane number attribute and use the capacity. This is done by using the option **--visum.recompute-laneno**.

Dealing with Connectors

VISUM as a macroscopic tool, does not regard single vehicles. As one consequence, VISUM uses "connections" through which traffic is fed from the districts into the network.

The effects on using such connectors within a microscopic simulations should be described at a different page. Here, we want to show the possibilities to change the connector attributes using NETCONVERT.

References

- PTV AG's VISUM-page (29.11.2008)
- list of known and used extensions

VissimNetworkImport

Importing Networks from Vissim files

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, does not work with all networks, and manual work on the imported network is necessary. A usage example for NETCONVERT's Vissim import may look like:

```
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 *<FLOAT>* 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 *<FLOAT>* 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 for joining connections to nodes, a further factor steers the process of converting Vissim-networks: simply spoken, --vissim.offset *<FLOAT>* 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.

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 are built

ArcViewNetworkImport

Importing Networks from ArcView data (shapefiles)

NETCONVERT is able to directly read binary ArcView databases. To convert such databases, at least three files are needed: 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. The option to load a shape-file into NETCONVERT in order to convert it into a SUMO-network is named --arcview *<FILE>*. Because shape-file descriptions use more than a single file, one has to supply the file name without extension, only. So, the following call to NETCONVERT should be used to import the road network stored in "my_shape_files.shp", "my_shape_files.shx", "my_shape_files.proj", and "my_shape_files.dbf":

```
netconvert --arcview my_shape_files
```

Unfortunately, shape files describe how information is stored physically, but neither which is stored nor how the entries of the according database (*.dbf) are named. Due to this, one has to examine how a given road network is stored within the database file and give this information to NETCONVERT; a plain call almost always fails.

The table below shows which information NETCONVERT is trying to read from the given input files, what the standard values are, and how they can be changed. Also, it shows whether the information is mandatory - must be given - or optional.

Information NETCONVERT reads from shape-files

Information	Mandatory	Default field name	Option
The id of an edge	y	**LINK_ID**	--arcview.street-id *<FIELD_NAME>*
The name of an edge (not really used)	n	**ST_NAME**	
The name of the node the edge starts at	y	**REF_IN_ID**	--arcview.from-id *<FIELD_NAME>*
The name of the node the edge ends at	y	**NREF_IN_ID**	--arcview.to-id *<FIELD_NAME>*
The type of the street	n	**ST_TYP_AFT**	--arcview.type-id *<FIELD_NAME>*
Speed category	n (see below)	SPEED_CAT	
Lane category	n (see below)	**LANE_CAT**	
Road class, used to determine the priority	n (see below)	**FUNC_CLASS**	
Allowed speed on a road	n (see below)	**SPEED**	
Number of lanes a road has	n (see below)	**NOLANES** or **rnol**	

If being familiar with NavTeq, you may have noticed that the defaults are the ones that are used by NavTeq.

Some shape file databases do not contain explicite information about the edges' attributes (number of lanes, priority, allowed speed) at all. Since version 0.9.4 it is possible use road types to describe the edges' attributes. In this case, the column to retrieve an according street's type name from must be named using --arcview.type-id *<ID>* and a type-file must be given to NETCONVERT using --xml-type-files *<FILE>*. If something fails with the types or the explicite values, it can be catched using --arcview.use-defaults-on-failure. In these cases, the default NETCONVERT values are used. Besides this, it is possible to load own connection descriptions.

Note:
One can insert own attributes into the database using a GIS. This means, one can also insert own fields for the number of lanes, priority, or allowed speed, naming them as described above.
ArcView-networks are encoded using geocoordinates which have to be converted to the cartesian coordinates system used by SUMO. To describe how to convert the coordinates, one should know in which UTM-zone your network is located. This can be passed to NETCONVERT using --arcview.utm *<INT>*. If the conversion can not be initialised, the option --arcview.guess-projection may be used for letting NETCONVERT guess the conversion by its own.

Examples

'36_NEW_YORK' from TIGER database

The network is available at the TIGER2008 ftp server (?). **Missing copyright information; please add**.

Opening the edges-dbf (*tl_2008_36061_edges.dbf*) of which we hope that it contains information about roads shows us, that:

- the from- and the to-nodes of the streets are described in fields named **tfidl** and **tfidr**, respectively.
- the field **tlid** may be stored as the one to name the edges by
- there is a further information about the type (mtfcc)

As a begin, we try to import the road graph. We use the information we have found and apply a projection:

```
netconvert -v --arcview tl_2008_36061_edges -o net.net.xml \
   --add-node-positions \
   --arcview.from-id tfidl --arcview.to-id tfidr --arcview.street-id tlid \
   --arcview.use-defaults-on-failure --use-projection --arcview.guess-projection
```

As a result, we obtain the network shown below.

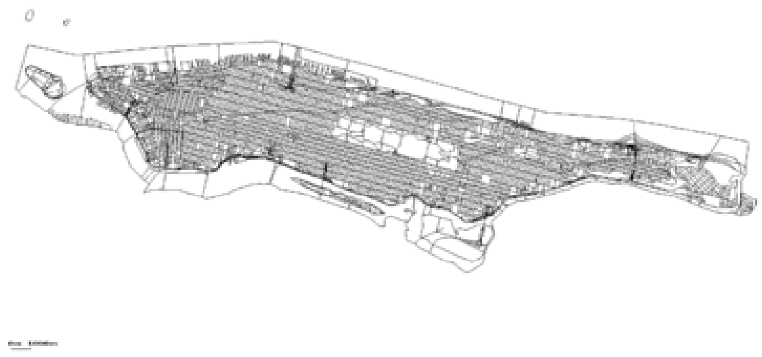


Figure 1.1. The converted network of New York

There are several issues one should note:

- The types were not used - no information about their meanings was investigated
- All roads are unidirectional
- The projection has not been veryfied (though it should be valid)
- Maybe the further files which are included in the zip contain additional information. This has not been investigated.

'Frida' network (city of OsnabrÃ¼ck)

The network is available at the Frida-homepage and is licensed under the GPL.

Our main interest is of course the street network. The following files describe this: *strassen.dbf*, *strassen.shp*, *strassen.shx* ("strassen" is the german word for "streets"). When opening "strassen.dbf" we have to realize that there is only a few information stored herein - neither the node names are given nor the street attributes. Instead, the street attributes seem to be stored in an additional database and are references by type names (column "strTypID" - strassen_typ_id = street_type_id). Also, the names of this database's columns have other names than expected.

Ok, let's solve these problems one after another.

- Different field naming

  The only problem with this is that we can not extract street names properly. Still, within FRIDA, the edges are numbered, and we may use the street id as name.
  The call has to be extended by: --arcview.street-id strShapeID

- Missing node names

  NETCONVERT can deal with networks which do not have node names (since version 0.9.4).

- Parsing street attributes from a type file

  The possibility to describe edges using attributes was already available in NETCONVERT and may be used in combination with ArcView files since version 0.9.4. Still, the types have to be translated into XML. The generated file (*frida.typ.xml*) looks like this:

```
<types>
    <!-- "noch nicht attributiert" (= not yet attributed) -->
    <type id="0"  priority="1" nolanes="1" speed="13.89"/>
    <!-- "Autobahn" (highway) -->
    <type id="1"  priority="5" nolanes="3" speed="41.67"/>
    <!-- "Bundesstrasse" (motorway) -->
    <type id="2"  priority="4" nolanes="1" speed="22.22"/>
    <!-- "Hauptstrasse" (main (city) road) -->
    <type id="3"  priority="3" nolanes="2" speed="13.89"/>
    <!-- "Nebenstrasse" (lower priorised (city) road) -->
    <type id="4"  priority="2" nolanes="1" speed="13.89"/>
    <!-- "Weg" (path) -->
    <type id="5"  priority="1" nolanes="1" speed="5"/>
    <!-- "Zone 30" (lower street with a speed limit of 30km/h) -->
    <type id="6"  priority="2" nolanes="1" speed="8.33"/>
    <!-- "Spielstrasse" (a street where children may play (10km/h)) -->
    <type id="7"  priority="1" nolanes="1" speed="1.39"/>
    <!-- "Fussgaengerzone" (pedestrains zone) -->
    <type id="8"  priority="0" nolanes="1" speed="0.1"/>
    <!-- "gesperrte Strasse" (closed street) -->
    <type id="9"  priority="0" nolanes="1" speed="0.1"/>
    <!-- "sonstige Strasse" (something else) -->
    <type id="10" priority="0" nolanes="1" speed="0.1"/>
    <!-- "Fussweg" (way for pedestrians) -->
    <type id="11" priority="0" nolanes="1" speed="0.1"/>
</types>
```

The call has to be extended by: -t frida.typ.xml --arcview.type-id strTypID

After having applied all those changes, the network was buildable, but looked quite messy. After having played with geocoordinate projections, this was fixed. So the call (so far) looks like this:

```
netconvert --arcview strassen -o frida.net.xml \
  --arcview.street-id strShapeID -t frida.typ.xml \
  --arcview.type-id strTypID --use-projection
```

Data Quality

Looking a bit deeper at the network, we had to realise two further problems. At first, highway on- and off-ramps are marked as "highway". this yields in a network where on- and offramps have the same number of lanes as the highways themselves. And it's definitely not fitting to reality, as the next picture shows:

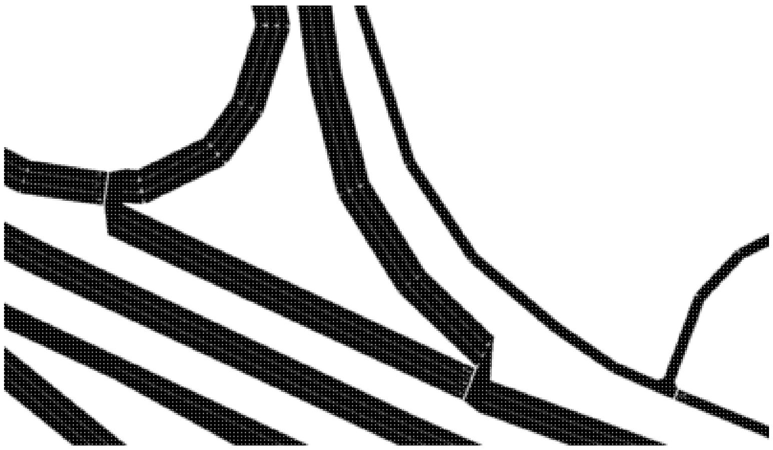


Figure 2.2. Detail view showing problems with (unidirectional) highway on-/off-ramps

Furthermore, all streets are unidirectional - even highways. This makes the network not usable for traffic simulations when left in the orignal state. Trying to convert the network with --arcview.all-bidi, that means trying to insert edges bidirectional, makes the city usable, but the highways are even worse, now, because also the on-/off-ramps are bidirectional, then...

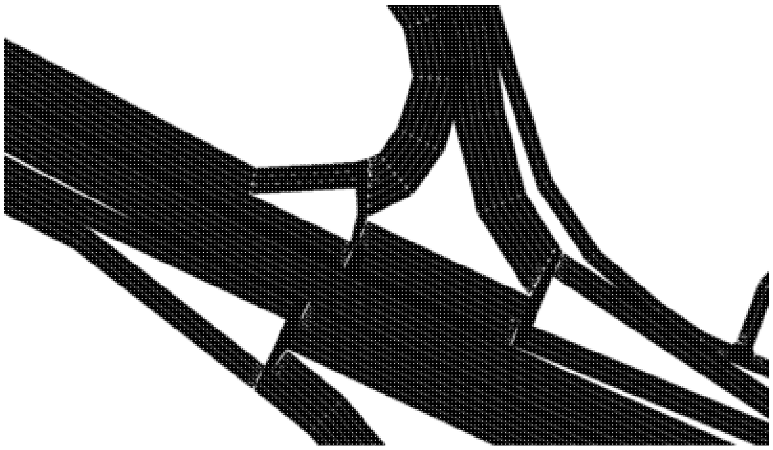


Figure 2.3. Detail view showing problems with (bidirectional) highway on-/off-ramps

Demand

There is no demand available for Frida - at least none we know about.

Conlusion

Using the current features we are able to parse the network from the Frida-project but we can not state it is completely usable for traffic simulations. At least areas around highways are not realistic, because on-/offramps lack an explicite declaration and are due to this as wide as the highways themselves. Furthermore, all streets within the network are coded in just one direction. Extending them to be bidirectional solves the problem in inner-city areas, but yields in an unacceptable result for highways.

ElmarNetworkImport

Importing Elmar's networks

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 *<FILE>* loads the splitted definitions, --elmar2 *<FILE>* 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".

TigerNetworkImport

Importing Networks from TIGER

Note:
TIGER networks have found their way into OSM (OpenStreetMap); please use the OSM networks, here.
Note:
Further, some iof the recent TIGER files we've seen were stored in ArcView's shape file format. Please read the according chapter on importing ArcView files in the case your TIGER files are stored as such.
The possibility to import TIGER files directly was removed.

OpenStreetMapImport

Introduction

From http://www.openstreetmap.org/: *"OpenStreetMap is a free editable map of the whole world. It is made by people like you."*

Map data from the OpenStreetMap project (OSM-data) comes in the shape of one or more XML-files.

Note:
OSM-data can be downloaded via several different methods as described in OpenStreetMapDownload

Importing the Road Network

NETCONVERT can import OSM-files natively. The according option is named --osm-files *<FILE>[,<FILE>]**.

Yhe following call to NETCONVERT imports the road network stored in "berlin.osm.xml" and stores the SUMO-network generated from this data into "berlin.net.xml":

```
netconvert --osm-files berlin.osm.xml -o berlin.net.xml
```

OSM-data has always WGS84 geo coordinates which will be automatically UTM transformed by netconvert (since sumo 0.11.1). Thus you need explicit projection parameters only if you need a different projection. Refer to the NETCONVERT documentation for other conversion options.

Importing additional Polygons (Buidlings, Water, etc.)

OSM-data not only contains the road network but also a wide range of additional polygons such as buildings and rivers. These polygons can be imported using POLYCONVERT and then added to a `sumo-gui`-configuration.

To interpret the OSM-data an additional *typemap*-file is required:

```
 <polytypes>
        <polytype id="waterway" name="water" color=".71,.82,.82" layer="-10"/>
        <polytype id="natural.water" name="water" color=".71,.82,.82" layer="-10"/>
        <polytype id="natural.wetland" name="water" color=".71,.82,.82" layer="-10"/>
        <polytype id="sport" name="sport" color=".31,.90,.49" layer="2"/>
        <polytype id="landuse.forest" name="forest" color="0.55,.77,.42" layer="-11"/>
        <polytype id="natural.wood" name="forest" color="0.55,.77,.42" layer="-11"/>
        <polytype id="natural" name="natural" color="0.55,.77,.42" layer="2"/>
        <polytype id="landuse.park" name="park" color=".81,.96,.79" layer="-11"/>
        <polytype id="leisure" name="leisure" color=".81,.96,.79" layer="0"/>
        <polytype id="leisure.park" name="tourism" color=".81,.96,.79" layer="-10"/>
        <polytype id="tourism" name="tourism" color=".81,.96,.79" layer="2"/>
        <polytype id="landuse" name="landuse" color=".76,.76,.51" layer="-4"/>
        <polytype id="landuse.residential" name="residential" color=".92,.92,.89" layer="-11"/>
        <polytype id="landuse.commercial" name="commercial" color=".82,.82,.80" layer="-11"/>
        <polytype id="shop" name="shop" color=".93,.78,1.0" layer="2"/>
        <polytype id="landuse.industrial" name="industrial" color=".82,.82,.80" layer="-11"/>
        <polytype id="man_made" name="building" color="1.0,.90,.90" layer="2"/>
        <polytype id="building" name="building" color="1.0,.90,.90" layer="2"/>
        <polytype id="amenity" name="amenity" color=".93,.78,.78" layer="2"/>
        <polytype id="amenity.parking" name="parking" color=".72,.72,.70" layer="-2"/>
        <polytype id="military" name="military" color=".60,.60,.36" layer="-10"/>
        <polytype id="landuse.military" name="military" color=".60,.60,.36" layer="-10"/>
        <polytype id="landuse.farm" name="farm" color=".95,.95,.8" layer="-11"/>
        <polytype id="landuse.greenfield" name="farm" color=".95,.95,.8" layer="-11"/>
        <polytype id="landuse.village_green" name="farm" color=".95,.95,.8" layer="-11"/>
        <polytype id="power" name="power" color=".1,.1,.3" layer="5"/>
        <polytype id="natural.land" name="land" color=".98,.87,.46" layer="-9"/>
        <polytype id="boundary.administrative" name="boundary.administrative" color=".5,.0,.2" la
        <polytype id="aeroway" name="aeroway" color=".5,.5,.5" layer="1"/>
        <polytype id="aerialway" name="aerialway" color=".2,.2,.2" layer="1"/>
        <polytype id="historic" name="historic" color=".5,1,.5" layer="2"/>
 </polytypes>
```

Using the typemap file *typemap.xml* the following call to POLYCONVERT imports polygons from OSM-data and produces a Sumo-polygon file.

```
polyconvert --net-file berlin.net.xml --osm-files berlin.osm --typemap typemap.xml -o berlin.poly
```

The created polygon file *berlin.poly.xml* can then be added to a `sumo-gui` configuration:

```
 <configuration>
     <input>
         <net-file value="berlin.net.xml"/>
         <additional-files value="berlin.poly.xml"/>
     </input>
 </configuration>
```

Import Scripts

The help script *osmGet.py* allows downloading a large area by splitting the requests (see OpenStreetMapDownload#Downloading a large area). This results in multiple OSM-data files which can be imported with the scripts *osmBuild.Py* and *osmBuildPolys.py*, located in ***<SUMO>***/tools/import/osm.

The call is:

```
osmGet.py <PREFIX> <BOUNDING_BOX> <TILES_NUMBER>
osmBuild.py <PREFIX> <TILES_NUMBER> <OUTPUT> [(all|road|passenger),ramps,tls]
osmBuildPoly.py <PREFIX> <TILES_NUMBER> <SUMONET> <OUTPUT>
```

<PREFIX> and <TILES_NUMBER> must be the same as given for the *osmGet.py* call. <SUMONET> must match the <OUTPUT> parameter of the *osmBuild.py* call.

Note:
The directory in which this script is executed must also contain a typemap file named `osm_type_map.xml`.

If "road" is given as last parameter, only roads usable by road vehicles are extracted, if "passenger" is given, only those accessible by passenger vehicles. If "ramps" is given, the option --guess-ramps is set for the NETCONVERT call, if "tls" is set, the option --guess-tls.

Further Notes

Traffic Lights

OSM does not have the possibility to assign several nodes to a single traffic light. This means that near-by nodes, normally controlled by one traffic light system are controlled by two after the network is imported. It is obvious that traffic collapses in such areas if both traffic lights are not synchronized. Better representation of the reality can be achieved by giving the option --try-join-tls to NETCONVERT. NETCONVERT then assigns near-by nodes to the same traffic light.

Highway On- and Off-Ramps

OSM networks often lack additional lanes for highway on- and off-ramps. They can be guessed via NETCONVERT using the --guess-ramps option.

Editing OSM networks

JOSM

From George Dita, on 01.07.2009 JOSM can be used to edit OSM-data (i.e. for trimming a rectangular map and deleting unwated features). After you delete the part that does not interest you you have to alter the file using xmlstarlet which actually deletes the nodes.

and xmlstarlet can be used like this:

```
xmlstarlet ed -d "/osm/*[@action='delete']" < input.osm > output.osm
```

OSMOSIS

From Christian Klotz, on 01.07.2009, tip by Christoph Sommmer

The java tool osmosis (http://wiki.openstreetmap.org/index.php/Osmosis) can be used to filter out unwanted features from an OSM-file. The following command keeps motorways and motorway links while filtering out everything else:

```
java -jar osmosis.jar --read-xml file="orginal.osm.xml" --way-key-value \
    keyValueList="highway.motorway,highway.motorway_link" \
    --used-node --write-xml file="filtered.osm.xml"
```

NETCONVERT Details

Road Types

When importing road networks, NETCONVERT searches for the street type, encoded in OSM as a key/value-pair where the key is either "*highway*" or "*railway*". Only if such a key occures in the edge definition, the edge is imported (see also below). The edge's type name is built from the found key/value pair by building a name as: <*KEY*>.<*VALUE*>. Using this type name, the edge's attributes are determined using a predefined map of type names to type definitions. It is possible to override the default types by giving own type definitions; of course the type names must match the read ones.

Type Name	#lanes	speed	priority	allowed vehicle classes	one-way is default
highway.motorway	3	160 km/h	13	all (SVC_UNKNOWN)	x
highway.motorway_link	3		12		x

		80 km/h		all (SVC_UNKNOWN)
highway.trunk	2	100 km/h	11	all (SVC_UNKNOWN)
highway.trunk_link	1	80 km/h	10	all (SVC_UNKNOWN)
highway.primary	2	100 km/h	9	all (SVC_UNKNOWN)
highway.primary_link	1	80 km/h	8	all (SVC_UNKNOWN)
highway.secondary	2	100 km/h	7	all (SVC_UNKNOWN)
highway.tertiary	1	80 km/h	6	all (SVC_UNKNOWN)
highway.unclassified	1	80 km/h	5	all (SVC_UNKNOWN)
highway.residential	1	50 km/h	4	all (SVC_UNKNOWN)
highway.living_street	1	10 km/h	3	all (SVC_UNKNOWN)
highway.service	1	20 km/h	2	SVC_DELIVERY
highway.track	1	20 km/h	1	all (SVC_UNKNOWN)
highway.pedestrian	1	30 km/h	1	SVC_PEDESTRIAN
highway.services	1	30 km/h	1	all (SVC_UNKNOWN)
highway.unsurfaced	1	30 km/h	1	all (SVC_UNKNOWN)
highway.footway	1	30 km/h	1	SVC_PEDESTRIAN
highway.pedestrian	1	30 km/h	1	SVC_PEDESTRIAN
highway.path	1	10 km/h	1	SVC_PEDESTRIAN
highway.bridleway	1	10 km/h	1	SVC_PEDESTRIAN
highway.cycleway	1	20 km/h	1	SVC_BICYCLE
highway.footway	1	10 km/h	1	SVC_PEDESTRIAN
highway.step	1	5 km/h	1	SVC_PEDESTRIAN
highway.steps	1	5 km/h	1	SVC_PEDESTRIAN
highway.stairs	1	5 km/h	1	SVC_PEDESTRIAN

highway.bus_guideway	1	30 km/h	1	SVC_BUS
railway.rail	1	30 km/h	1	SVC_RAIL_FAST
railway.tram	1	30 km/h	1	SVC_CITYRAIL
railway.light_rail	1	30 km/h	1	SVC_LIGHTRAIL
railway.subway	1	30 km/h	1	SVC_CITYRAIL
railway.preserved	1	30 km/h	1	SVC_LIGHTRAIL
railway.monorail	1	30 km/h	1	SVC_LIGHTRAIL

These values are based on http://wiki.openstreetmap.org/wiki/Map_Features.

Caution:
The values were set-up ad-hoc and are not yet verified. It would be a great help if someone would revisit and improve them. Please let us know.

Explicite Road Attributes

In the case an edge contains the definition about the number of lanes (key="*lanes*") or the allowed speed (key="*maxspeed*"), this information is used instead of the according type's value. Also, the per-edge information whether the edge is a one-way edge is read (key="*oneway*"). In the case the edge belongs to a roundabout (key="*junction*" and value="*roundabout*"), it is also set as being a one-way edge.

Dismissing unwanted traffic modes

In most cases, tracks and edges which not may be crossed by motorised traffic are not interesting for road traffic research. It is possible to exclude these edges from being imported using the NETCONVERT-option --remove-edges.by-vclass *<STRING>[,<STRING>]**.

For removing all edges which can not be used by passenger vehicles the call must be extended by:

```
--remove-edges.by-vclass hov,taxi,bus,delivery,transport,lightrail,cityrail, \
  rail_slow,rail_fast,motorcycle,bicycle,pedestrian
```

For removing all edges which can not be used by road vehicles the call must be extended by:

```
--remove-edges.by-vclass rail_slow,rail_fast,bicycle,pedestrian
```

Missing Descriptions

- TLS computation
- computation of lane-2-lane connections

- what is exactly imported (how edge attributes are determined)
- other traffic modes
- Network quality

References

- http://www.openstreetmap.org/ - the home site
- http://www.openstreetmap.de/ - the German home site
- http://wiki.openstreetmap.org/index.php/Map_Features - information about database attributes

RoboCupNetworkImport

Importing Networks from RoboCup Rescue League

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 --robocup-net *<FILE>* (or shorter: --robocup *<FILE>*).

FurtherNetconvertOptions

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.

Setting default Values

It was mentioned, that edge parameter may be omitted and defaults will be used in this case. It is possible to set default values for the imported edges' type, number of lanes, allowed speed, and priority, using the options --type *<ID>* (or -T *<ID>* for short), --lanenumber *<INT>* (or -L *<INT>* for short), --speed *<FLOAT>* (or -S *<FLOAT>* for short), --priority *<INT>* (or -P *<INT>* for short), respectively.

Adding Turnarounds

Normally, turnarounds are added as a possible edge continuations and play an important role during network building (see Publications#Krajzewicz_et_al2005_2). Still, one may want not to add them. In this cases, it is possible to disallow their appending using option --no-turnarounds.

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 ading a further geometry point at the removed node's position.

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 option --speed-in-kmh to NETCONVERT.

Importing Networks without Traffic Light Logics

Some of the supported network formats 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.

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.

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.

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.

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 used to "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 0.10.0 SUMO will by default 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, you may want to return to the old behavior using the option --no-internal-links.

Pruning the imported network

NETCONVERT offers you some possibilities 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 *<ID>[,<ID>]** where each *<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 *<FILE>*. 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 *<FLOAT>* for this where *<FLOAT>* is the minimum velocity an edge must allow in order to be included in the output in m/s.

NetconvertOutput

Additional NETCONVERT output

NETCONVERT and NETGEN allow to generate additional output files besides 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 Node Descriptions and Edge Descriptions). This shall ease processing of networks read from other formats than XML. The option --plain-output *<FILE>* forces NETCONVERT and NETGEN to generate a file named "*<FILE>*.nod.xml" which contains the previously imported nodes, a file named "*<FILE>*.edg.xml" which contains the previously imported edges, and a file named "*<FILE>*.con.xml" which contains the previously imported connections. 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").

Information about Geometry Removal

The option --map-output *<FILE>* (or -M *<FILE>* for short) generates a file which contains the information about which edges have been joined (see chapter "Removing Geometry Nodes").

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
```

Node Geometries Dump

The option --node-geometry-dump *<FILE>* 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.

NetworkGeneration

Network Generation

NETGEN allows to build three types of abstract networks: 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.

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. It is possible to give another option --attach-length, which adds streets of the given length at the boundary of the grid such that all crossings have four streets (It is not yet possible to have different attach lengths for x- and y-direction).

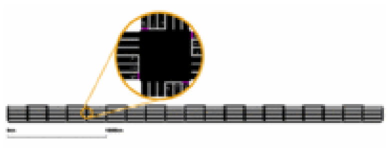
An example usage for building could be:

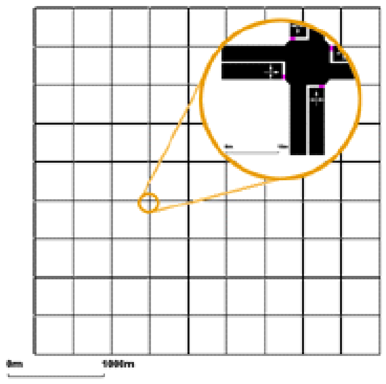
```
netgen --grid-net --grid-number=10 --grid-length=400 --output-file=MySUMOFile.net.xml
```

Another one:

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

These calls will generate the following networks, respectively:





Spider-like Networks

Spider-net networks are defined by the number of axes dividing them (parameter --spider-arm-number or --arms, default is 13), the number of the circles they are made of (--spider-circle-number or --circles, default is 20) and the distance between the circles (--spider-space-rad or --radius in meters, default is 100).

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 unregulated.
Optionally you can omit the central junction of the network by specifying --spider-omit-center or --nocenter. This also gives an easy way of generating a circle network. Using for instance

```
netgen --spider-net --spider-omit-center --output-file=MySUMOFile.net.xml
```

will create a circle consisting of 13 elements with a radius of 100m.

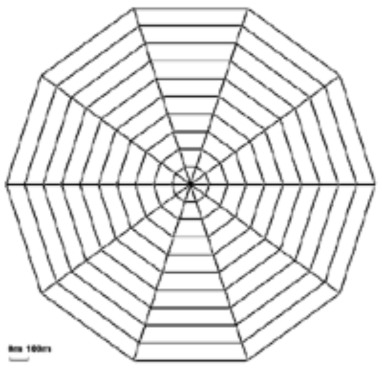
Two examples of usage:

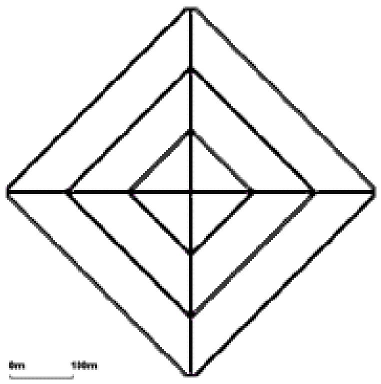
```
netgen --spider-net --spider-arm-number=10 --spider-circle-number=10 \
 --spider-space-rad=100 --output-file=MySUMOFile.net.xml
```

and:

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

These calls will generate the following networks, respectively:





Random Networks

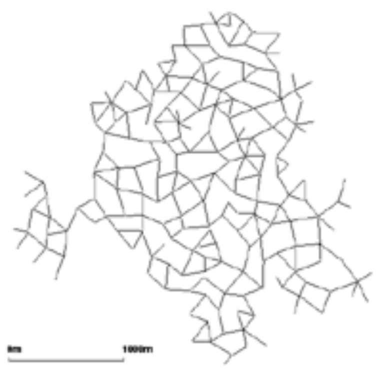
The random network generator does just what his name says, it builds random networks... Several settings may be changed:

- --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:

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

This call will generate the following network:



Further Options

All abstract network 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.

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

How Networks are built

Both NETCONVERT and NETGEN share the same process of building networks which is started as soon as data are read (in NETCONVERT) or after an internal description is generated (in NETGEN).

The process requires that at least nodes and edges are given, optionally also connections between edges and/or lanes and other optional information. Hints about how the network shall be built - whether turnarounds or highway on/off-ramps shall be added, traffic lights or roundabouts shall be guessed, etc. are retrieved from the options.

Below, you may find a list of all steps performed while building the network, of which some are optional.

1. Removing dummy edges
2. Joining double connections
3. Finding isolated roads
4. Removing empty nodes and geometry nodes (optional)
5. Removing unwished edges (optional)
6. Rechecking nodes after edge removal (optional)
7. Splitting geometry edges (optional)
8. Normalising node positions
9. Guessing and setting on-/off-ramps
10. Guessing and setting TLs
11. Computing turning directions
12. Sorting nodes' edges
13. Computing Approached Edges
14. Computing Approaching Lanes
15. Dividing of Lanes on Approached Lanes
16. Appending Turnarounds (optional)
17. Rechecking of lane endings
18. Computing node shapes
19. Computing edge shapes
20. Computing tls logics
21. Computing node logics
22. Computing traffic light logics
23. Transposing network (optional)

These computation steps are perfomed within void NBNetBuilder::compute(OptionsCont &oc). These steps are described more detailed in the following.

Steps #1-#7: Deciding what to import

In steps #1-#7 (Removing dummy edges, Joining double connections, Finding isolated roads, Removing empty nodes and geometry nodes, Removing unwished edges, Rechecking nodes after edge removal, Splitting geometry edges) the nodes/edges stored during importing/generation are revisited in order to:

- avoid multiple edges join the same nodes in same direction (Joining double connections, to be discussed)
- avoid unused parts of the network (Finding isolated roads, to be discussed)
- remove/build geometry nodes (Removing empty nodes and geometry nodes, Splitting geometry edges)
- remove unwanted edges (Removing unwished edges)

As a result, the internal representation of the read/generated network contains the sets of nodes/edges the user wanted.

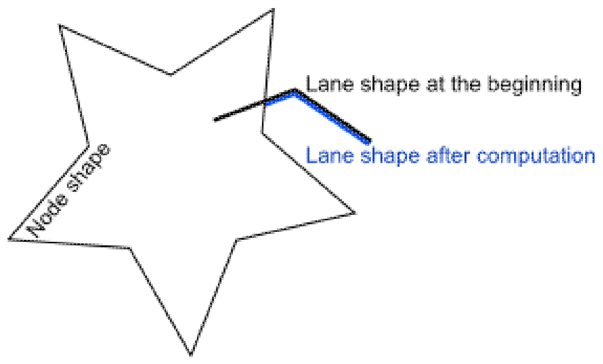
Step #8: Move the network to coordinates' origin

Steps #9-#10: Computing certain signalling/right-of-ways

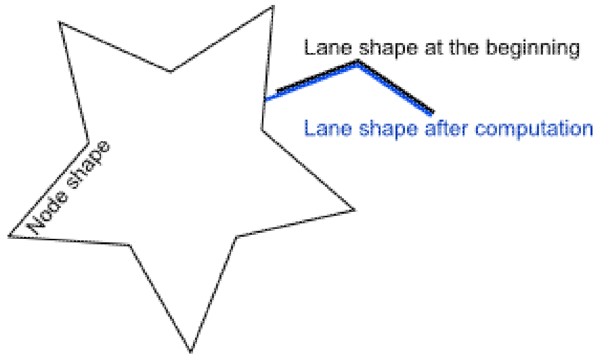
Steps #11-#17: Computing and setting lane-to-lane connections

Steps #18-#19: Computing geometries

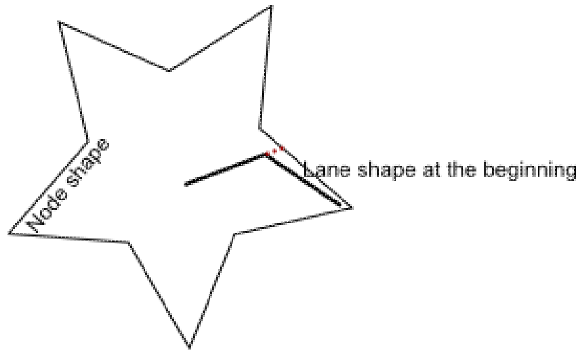
I am currently concerned with the edge shapes, because even a eworld or a TrafficModeler export stops here in debug mode. It is in NBEdge::computeEdgeShape. (Actually, as far as I understand this method, it is about the lane shapes of this edge.) In this algorithm, the following three cases must be distinguished:



The edge shape intersects with the node shape. Then, the shape is trimmed to the intersection point. This looks quite reasonable.



The edge shape is outside the node shape. The the present algorithm extrapolates the last line of the edge shape and enlarges the edge to the new intersection point. This looks reasonable to me as well.



The edge shape is completely within the node shape. What should happen here? (The red dots show, what netconvert does at present.)

So I am wondering, what the right behaviour of netconvert would be in the third case.

Steps #20-#22: Computing right-of-way rules

Steps #23: Applying additional offset

Demand Modelling

IntroductionDemand

Introduction to demand modelling in SUMO

After having generated a network, one could take a look at it using GUISIM, but no cars would be driving around. One still needs some kind of description about the vehicles. 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. SUMO and GUISIM need routes as input for vehicle movements. 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

- Using flow definitions

  This is mostly the same approach as using trip definitions, but one may join vehicles having the same departure and arrival edge using this method
- Using flow definitions and turning ratios

  One may also leave out the destination edges for flows and use turning ratios at junctions instead
- Using OD-matrices

  OD-matrices have to be converted to trips first, then from trips to routes
- By hand

  You can of course generate route files by hand
- Using random routes

  This is fast way to fill the simulation with life, but nothing that has something to do with reality
- Describing the population in the net

  This method is called activity-based demand modelling.
- By importing available routes

By now, the SUMO-package contains four applications for processing routes. DUAROUTER is responsible for importing routes or their definitions from other simulation packages and for computing routes using the shortest-path algorithm by Dijkstra. Additionally, in combination with the simulation, the DUAROUTER can compute the dynamic user assignment formulated by C. Gawron. 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. The DFROUTER computes routes from given observation point measures.

Definition of Vehicles, Vehicle Types, and Routes

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. Before starting, it is of importance to know that a vehicle in SUMO consists of three parts: a vehicle type which describes the vehicle's physical properties, a route the vehicle shall take, and the vehicle itself. Both routes and vehicle types can be shared by several vehicles.

In the following, building a route file which contains all these information will be presented by example. The file will be named "routes.rou.xml".

Vehicle Types

At first a vehicle type will be defined:

```
<routes>
    <vtype id="type1" accel="0.8" decel="4.5" sigma="0.5" length="7.5" maxspeed="70"/>
</routes>
```

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

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

For allowing to use different car-following models than the one developed by KrauÃ , the vehicle type definition was extended. The initial one still can be used, but an extension allows to additionally choose a different model and give its parameter. Here is how the new description looks like (for the same vehicle type as above):

```
<routes>
    <vtype id="type1" length="7.5" maxspeed="70">
        <carFollowing-Krauss accel="0.8" decel="4.5" sigma="0.5"/>
    </vtype>
</routes>
```

You may note that the car-following model's parameter are now listed in a child-element of the vehicle type definition. Please note that the values of the carFollowing-element are overwriting values given in the vtype-element.

These values have the following meanings:

Attribute Name	Mandatory	Value Type	Description
id	y	id (string)	The name of the vehicle type
accel	n (belongs to the cf-model)	float	The acceleration ability of vehicles of this type (in m/s^2)
decel	n (belongs to the cf-model)	float	The deceleration ability of vehicles of this type (in m/s^2)
sigma	n (belongs to the cf-model)	float	The driver imperfection (between 0 and 1)
length		float	The vehicle's **brutto**-length (length + offset to leader while standing still) (in m)
maxspeed		float	The vehicle's maximum velocity (in m/s)

color	color	This vehicle type's color
vclass	class (enum)	An abstract vehicle class (see below)
emissionClass	emission class (enum)	An abstract emission class (see below)
guiShape	shape (enum)	How this vehicle is rendered
guiWidth	float	The vehicle's width [m]
guiOffset	float	Empty space after leader [m]

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

Abstract Vehicle Class

A SUMO vehicles may be assigned to an "abstract vehicle class", defined using the attribute vclass. These classes are used in lane definitions and allow/disallow the usage of lanes for certain vehicle types. One may think of having a road with three lanes, where the rightmost may only be used by "taxis" or "busses". The following vehicle classes exist:

- "private"
- "public_transport"
- "public_emergency"
- "public_authority"
- "public_army"
- "vip"
- "ignoring"
- "passenger"
- "hov"
- "taxi"
- "bus"
- "delivery"
- "transport"
- "lightrail"
- "cityrail"
- "rail_slow"
- "rail_fast"
- "motorcycle"
- "bicycle"
- "pedestrian"

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

Vehicle Emission Classes

The emission class represents a certain HBEFA-based emission class. It is defined using the emissionClass attribute. Template:Missing

Visualization

For a nicer visualization of the traffic, the appearence of a vehicle type's vehicles may be changed by assigning them a certain shape using the guiShape attribute. The following shapes are known:

- "pedestrian"
- "bicycle"
- "motorcycle"
- "passenger"
- "passenger/sedan"
- "passenger/hatchback"
- "passenger/wagon"
- "passenger/van"
- "delivery"
- "transport"
- "transport/semitrailer"
- "transport/trailer"
- "bus"
- "bus/city"
- "bus/flexible"
- "bus/overland"
- "rail"
- "rail/light"
- "rail/city"
- "rail/slow"
- "rail/fast"
- "rail/cargo"
- "evehicle"

Caution:
Not all of these named shapes are implemented.
In addition, one can determine the width of the vehicle using the attribute guiWidth and how large the offset to the leader shall be using the attribute guiOffset. If a vehicle is drawn, the free space of guiOffset length is left blank, first, then the remaining size (length-guiOffset) is used for the vehicle's visual appearence.

When using shapes, one should consider that different vehicle classes (passenger vehicles or buses) have different lengths. Passenger vehicles with more than 10m length look quite odd, buses with 2m length, too.

Car-Following Models

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

Element	Short Name	Description
carFollowing-Krauss	SUMOKrauÃ	The KrauÃ -model with some modifications which is the default model used in SUMO
carFollowing-KraussOrig1	SKOrig	The original KrauÃ -model
carFollowing-PWagner2009	PW2009	A model by Peter Wagner, using Todosiev's action points
carFollowing-BKerner	Kerner	A model by Boris Kerner **Caution:** currently under work
carFollowing-IDM	IDM	The Intelligent Driver Model by Martin Treiber **Caution:** Problems with lane changing occur

Mostly, each model uses its own set of parameters. The following table lists which parameter are used by which model(s). Please note that car-following itself and the car-following models are not discussed, here. Their development and evaluation is one of our work's topics, so they should be described on a different page more verbose.

Attribute	Description	Models
accel	The acceleration ability of vehicles of this type (in m/s^2)	SUMOKrauÃ , SKOrig, PW2009, Kerner, IDM
decel	The deceleration ability of vehicles of this type (in m/s^2)	SUMOKrauÃ , SKOrig, PW2009, Kerner, IDM
sigma	The driver imperfection (between 0 and 1)	SUMOKrauÃ , SKOrig
tau	The driver's reaction time in s	SUMOKrauÃ , SKOrig, PW2009, Kerner
timeHeadWay		IDM
minGap		IDM
k		Kerner
phi		Kerner

Vehicles and Routes

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

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

   <vehicle id="0" type="type1" depart="0" color="1,0,0">
```

```
      <route edges="beg middle end rend"/>
   </vehicle>

</routes>
```

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

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

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

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

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

</routes>
```

A vehicle may be defined using the following attributes:

Attribute Name	Mandatory	Value Type	Description
id	y	id (string)	The name of the vehicle
type	y	float	The id of the vehicle type to use for this vehicle
route		id	The id of the route the vehicle shall drive along
color		color	This vehicle's color
depart		float (s)	The time step at which the vehicle shall enter the network
departlane		int/string (0,"random","free","departlane","allowed","best")	The lane on which the vehicle shall be inserted. "departlane" uses the departlane-value of the first edge on that vehicles

			route.
departpos		float(m)/string ("random","free","random_free","base")	The position at which the vehicle shall enter the net; "free" means the point closest to the start of the departlane where it is possible to insert the vehicle. "random_free" tries forcefully to find a free random position and if that fails, places the vehicle at the next "free" position. "base" sets the vehicle's depart position to the vehicle's length + eps (eps=.1m), this means the vehicle is completely at the begin of the depart lane.
departspeed		float(m/s)/string (0,"random","max")	The speed with which the vehicle shall enter the network.
arrivallane		int/string (0,"current")	The lane at which the vehicle shall leave the network **Note:** This option is currently not evaluated
arrivalpos		float(m)/string (0(1),"random","max")	The position at which the vehicle shall leave the network

			Note: This option is currently not evaluated
arrivalspeed		float(m/s)/string (0,"current")	The speed with which the vehicle shall leave the network **Note:** This option is currently not evaluated

One may notice, that the route itself also got a color definition, so the attributes of a route are: A vehicle may be defined using the following attributes:

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

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

- Routes have to be connected. At the moment the simulation does not raise an error if the next edge of the current route is not a successor of the current edge. The car will simply stop at the end of the current edge and will possibly be "teleported" to the next edge after a waiting time. This is very likely to change in future versions.
- Routes have to contain at least two edges. The simulation stops the car at the start of the last edge, thus a route consisting of a single edge is empty. This is likely to change in future versions of SUMO.
- The starting edge has to be at least as long as the car starting on it. At the moment cars can only start at a position which makes them fit on the road completely.
- The route file has to be sorted by starting times. In fact this is only relevant, when you define a lot of routes or have large gaps between departure times. The simulation parameter --route-steps, which defaults to 200, defines the size of the time interval with which the simulation loads its routes. That means by default at startup only route with departure time <200 are loaded, if all the vehicles have departed, the routes up to departure time 400 are loaded etc. pp. This works only if the route file is sorted. This behaviour may be disabled by specifying --route-steps 0.

The first three conditions can be checked using <SUMO_DIST>/tools/routecheck.py.

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

Route and vehicle type distributions

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

```
<routes>
    <vtypeDistribution id="typedist1">
        <vtype id="type1" accel="0.8" length="5" maxspeed="70" probability="0.9"/>
        <vtype id="type2" accel="1.8" length="15" maxspeed="50" probability="0.1"/>
    </vtypeDistribution>
</routes>

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

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

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

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

Stops

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

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

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

RouterInputs

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>"]/>.

Attribute Name	Mandatory	Value Type	Description
id	y	id (string)	The name of vehicles that will be generated using this trip definition
depart	y	int	The departure time of the (first) vehicle which is generated using this trip definition
from		edge id	The name of the edge the route starts at; the edge must be a part of the used network
to		edge id	The name of the edge the route ends at; the edge must be a part of the used network
fromtaz		district id	The name of the district the route starts at
totaz		district id	The name of the district the route ends at
period		int	The time after which another vehicle with the same route shall be emitted (optional)
repno		int	The number of vehicles to emit which share the same route (optional)
color		color	This generated vehicle's color
departlane		int/string (0,"random","free","departlane")	The lane on which the vehicle shall be inserted
departpos		float(m)/string ("random","free","random_free","base")	The position at which the vehicle shall enter the net; "free" means the

			point closest to the start of the departlane where it is possible to insert the vehicle. "random_free" tries forcefully to find a free random position and if that fails, places the vehicle at the next "free" position. "base" sets the vehicle's depart position to the vehicle's length + eps (eps=.1m), this means the vehicle is completely at the begin of the depart lane.
departspeed		float(m/s)/string (0,"random","max")	The speed with which the vehicle shall enter the network.
arrivallane		int/string (0,"current")	The lane at which the vehicle shall leave the network **Note:** see Definition of Vehicles, Vehicle Types, and Routes#Vehicles and Routes
arrivalpos		float(m)/string (0(1),"random","max")	The position at which the vehicle shall leave the network **Note:** see Definition of Vehicles, Vehicle Types, and Routes#Vehicles and Routes
arrivalspeed		float(m/s)/string (0,"current")	The speed with which the vehicle shall leave the network

			Note: see Definition of Vehicles, Vehicle Types, and Routes#Vehicles and Routes

Trip definitions can be converted into routes using DUAROUTER and the option "--trip-defs" or "-t":

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

Trips may contain source and destination districts as well as edges. If the districts shall be used for routing a districts file has to be specified using the option "--with-taz". Furthermore (if the net does not contain the districts) a districts file needs to be provided:

```
duarouter --trip-defs=<TRIP_DEFS> --net=<SUMO_NET> \
  --output-file=MySUMORoutes.rou.xml --districts=<DISTRICTS> --with-taz
```

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:

```
<flows>
    <flow id="0" from="edge0" to="edge1" begin="0" end="3600" no="100"/>
</flows>
```

and

```
<flows>
    <interval begin="0" end="3600">
        <flow id="0" from="edge0" to="edge1" no="100"/>
    </interval>
</flows>
```

Let's review flow parameter:

Attribute Name	Mandatory	Value Type	Description
id	y	id (string)	The name of vehicles that will be generated using this trip definition; vehicles

			and routes will be named "<id>_<RUNNING>" where <RUNNING> is a number starting at 0 and increased for each vehicle.
from	y	edge id	The name of the edge the routes start at; the edge must be a part of the used network
to		edge id	The name of an the edge the routes end at; the edge must be a part of the used network
type		type id	The type id of the vehicles to generate
begin		int	The begin time for the described interval
end		int	The end time for the interval; must be greater than <begin>; vehicles will be emitted between <begin> and <end>-1
no		int	The number of vehicles that shall be emitted during this interval
color		color	Defines the color of the vehicles and their routes
departlane		int/string (0,"random","free","departlane")	The lane on which the vehicle shall be inserted
departpos		float(m)/string ("random","free","random_free","base")	The position at which the vehicle shall enter the net; "free" means the point closest to the start of the departlane where it is possible to insert the vehicle. "random_free" tries forcefully to find a free random position and if that fails, places the vehicle at the next "free" position. "base" sets the vehicle's

		depart position to the vehicle's length + eps (eps=.1m), this means the vehicle is completely at the begin of the depart lane.
departspeed	float(m/s)/string (0,"random","max")	The speed with which the vehicle shall enter the network.
arrivallane	int/string (0,"current")	The lane at which the vehicle shall leave the network **Note:** see Definition of Vehicles, Vehicle Types, and Routes#Vehicles and Routes
arrivalpos	float(m)/string (0[1],"random","max")	The position at which the vehicle shall leave the network **Note:** see Definition of Vehicles, Vehicle Types, and Routes#Vehicles and Routes
arrivalspeed	float(m/s)/string (0,"current")	The speed with which the vehicle shall leave the network **Note:** see Definition of Vehicles, Vehicle Types, and Routes#Vehicles and Routes

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

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

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

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

Importing O/D Matrices

OD2TRIPS computes trips tables from O/D (origin/destination) matrices. OD2TRIPS assumes the matrix / the 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 with NETCONVERT districts stored in the VISUM input file are parsed and stored within the generated 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 *<FILE>* (--net *<FILE>* or -n *<FILE>* for short) option.

Because OD2TRIPS was used only to import data stored in VISUM/VISION/VISSIM formats, it assumes O/D to be stored in one of the formats used by these applications. Not all VISUM/VISION/VISSIM formats are supported, by now only two, namely the "V"- and the "O"-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"). 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 *<FILE>[,<FILE>]** (--od *<FILE>[,<FILE>]** or -d *<FILE>[,<FILE>]** for short) option followed by the list of files separated using a ','.

OD2TRIPS reads all matrices and generates 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 *<FILE>* option (--output *<FILE>* or -o *<FILE>* for short). 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. If a type is spuulied, but you do not want to include it within the output, set the --no-vtype option. The command line 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 *<INT>* (-b *<INT>*) and --end *<INT>* (-e *<INT>*), 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 be emitted 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 a O/D-matrix cell describes. 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 *<FLOAT>* option which assumes a float as parameter. All read flows will be multiplied with this value, 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.

Describing the Districts

A file containing a mapping from districts to edges 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 vehicle type information is used by OD2TRIPS by passing it to the generated vehicles. The type definition itself will not be generated, but the vehicle will have set the attribute type="*<TYPE>*". The time informations are assumed to be in the form <HOURS>.<MINUTES>. Please note that the end is exclusive; for example, if

```
0.00 1.00
```

is given, the generated vehicles' depart times will be second 0 to second 3599.

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 2 Sum = 15
         4          5          6
* District 2 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 seem to be left-aligned 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

OD2TRIPS allows splitting matrices which define a long time period into smaller parts which contain definite percentages of the whole. There are two ways of defining the amounts the matrix shall be split into. In both cases, the probabilities are automatically normed.

Free Range Definitions

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.

Daily Time Lines

The second case is rather common in transportation science. It allows to split the matrix into 24 subparts - this means the number of fields 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.

Some common daily time lines from Germany may be retrieved from: Schmidt, Gerhard; Thomas, Bernd: Hochrechnungsfaktoren fÃ¼r manuelle und automatische KurzzeitzÃ¤hlungen im Innerortsbereich. Hrsg.: Bundesministerium fÃ¼r Verkehr, Abteilung StraÃ enbau: Forschung StraÃ enbau und StraÃ enverkehrstechnik. Heft 732, Bonn-Bad Godesberg, 1996

Applicability of generic Time Lines

Name	Description	Reference
TGw2_PKW.txt	passenger vehicles, Tuesday-Thursday, cities in West Germany, type#2 = ~streets at inner city border	S.95
TGw3_PKW.txt	passenger vehicles, Tuesday-Thursday, cities in West Germany, type#3 = ~streets at city border	S.95
TGs1_PKW.txt	passenger vehicles, Saturday/Sunday, cities in West Germany, group#1 = ~inner city streets, large amount of trips to/back work, freeways with no connection to recreation areas	S.100 - 103
TGw_LKW.txt	transport vehicles, Monday-Thursday, cities in West Germany	S.98
TGs_LKW.txt	transport vehicles, Sunday, type: ~long distance roads, strong heavy duty vehicle numbers	S.102, 105

The time lines as such are given below, they can be directly copied into the command line

Generic Time Lines

Name	Time Line
TGw_LKW	0.3,0.4,0.4,0.6,0.8,2.0,4.8,7.5,9.0,8.7,9.0,9.0,7.5,8.4,7.8,6.9,5.4,4.0,2.7,1.8,1.2,0.9,0.6,0.3

TGw3_PKW	0.9,0.5,0.2,0.2,0.5,1.3,7.0,9.3,6.7,4.2,4.0,3.8,4.1,4.6,5.0,6.7,9.6,9.2,7.1,4.8,3.5,2.7,2.2,1.9
TGw2_PKW	0.8,0.5,0.4,0.3,0.4,1.2,4.5,7.4,6.6,5.2,5.0,5.0,5.2,5.3,5.6,6.7,8.4,8.6,7.4,5.0,3.9,3.0,2.1,1.6
TGs(1)_PKW	3.3,2.8,2.0,1.5,1.2,1.3,1.2,1.5,2.5,3.7,4.8,5.5,6.0,6.7,7.0,7.1,6.9,7.4,7.0,6.0,4.7,4.1,3.5,2.3
TGs_LKW	1.3,1.1,0.6,0.8,0.9,1.5,2.6,3.1,3.5,3.8,4.5,4.9,5.0,5.3,5.6,5.7,5.9,6.0,5.7,5.3,4.8,4.6,10.0,7.6

Dealing with broken Data

OD2TRIPS behaves here as following:

incomplete districts

- missing origin OR destination district: error
- missing origin AND destination district: warning

incomplete connections

- missing connection to an origin OR a destination district: error
- missing connection to an origin AND a destination district: error

See also

- OD2TRIPS

Random Routes

DUAROUTER and JTRROUTER used to generate random routes for a given road network with the option --random-per-second *<FLOAT>* (or -R *<FLOAT>* for short). Due to a number of bugs in this feature and because those routes are highly unrealistic, this behavior was replaced with a python random trip generation script which can also produce a route set employing the DUAROUTER.

See Also

- randomTrips.py script for random routes generation
- "bug" random routes only created for timestep 1
- "bug" About Generating random Routes

RouteGuessing

Missing:
This text is not complete. We hope to find someone who actually does the job as a part of his study. Please contact us if you are interested in this topic and know someone who can contribute.

Why random Routes are evil and some Bullets

In the past, both route computation applications from the SUMO suite, namely DUAROUTER and JTRROUTER had the possibility to generate random routes. Some people were happy with this feature; we were not, especially because of the frequent questions on the usage of this feature.

This text explains why it is strongly NOT recommended to use random routes. It also shows some attempts for solving the problem of having no demand by some more sophisticated approaches for route generation.

As a base, we will use two example networks for which we know the demand:

- The first is a network from the city of Bologna. It stems from the iTETRIS project and arrived us as a Vissim simulation scenario, including both the network and the routes. It is an urban scenario.
- The second is a higway scenario. We have reconstructed the routes using information collected on observation points (induction loops) for the TrafficOnline project. We suppose the demand to be valid.

Ok, now we build random routes for both scenarios, choosing randomly a source edge ("road") and a destination edge. We generate as many vehicles as are simulated within the real, correct demand. Here are the results:

Ok, what has happened, what is different?

- The probability to use a road differs between the original and the random demand
- The mean velocities differ
- In the original scenarios, there is less traffic during night, more at day, most during the peek hours.

We will now address the issues one by one.

Probability to use a Road

When using a random demand, we choose a small road with the same probability to be a starting/ending road as we do with a big one. One could argue that the capacity for standing vehicles of big roads are similar to those of small roads - may be, but I personally would assume that smaller roads are statistically less frequented during work time, mainly only during the rush hour where persons are leaving their homes or coming back. Shopping places are found rather on the major roads what makes them more probably to be used as origins/destinations during the day. Also, there are parking places etc, also mostly accessed from the major roads. Of course, this may be not true for shopping areas, where people try to find parking places in covered small streets. Nonetheless, this is supposed to be seldom - and, you don't know this from just looking at the road network.

As a conclusion, it is rather not assumed to choose minor and major roads for origins and sources with the same frequency. Still, we have no solution for this. But read further...

Transit Traffic

Choosing randomly an edge from within the network completely ignores the fact that the smaller the simulated area is, the more traffic is just transiting the network - enters the network at its boundaries and leaves the network at its boundaries.

Of course, for our highway scenario, this is probably the largest source of mismodelling - no one starts or ends his trip in at a place in the middle of the highway.

So for improving our method for route generation, we have to determine the network's boundaries and the edges which are incoming into it or outgoing from it. We can then try to use those as the major origins/destination of our routes.

Turning Directions

Some people who tried the random routes generator may have noticed that many vehicles were turning. This is BTW also the case with our first "sophisticated" approach. The reason is very simple: if we choose two roads randomly, the probability to choose an origin which points to the opposite direction then the destination is located in is about 50%; sure, in detail this depends on the network topology. The same counts for the destination; the probability that the route ends at the road into the opposite direction is also high, maybe not 50%, but still high.

In real life, you have to turn at least once, too. Nonetheless, due to not having transit traffic, the overall number of turnings is much larger in random routes than in reality.

ImportingWithJtrRouter

Routing by Turn Probabilities

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

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 has 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%. 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%.

The definitions of the flow is the same as 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 defined, one can support a list of edges to be declared as sinks using --sinks <EDGE_ID>[,<EDGE_ID>]*. You may also add your sink definitions to a turn-file (XML only):

```
<turn-defs>
   ... some further turning definitions as above ...

   <sink><EDGE_ID></sink>
   ... further sink definitions ...

</turn-defs>
```

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.

DfRouterUsage

Generating routes from observation point measures using 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 induction 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 induction 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

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 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 values: "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.

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. --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.

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 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.

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, 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/induction loops, see "E1-Detectors (Induction 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.

ActivityGenUsage

ActivityGen generates demand from a description of the population in the net. To do so, it uses a simple activity-based traffic model. It supports the activities work, school, and free time and the conveyances walking, bike, car, and bus. Cars may have their start or stop location outside the map.

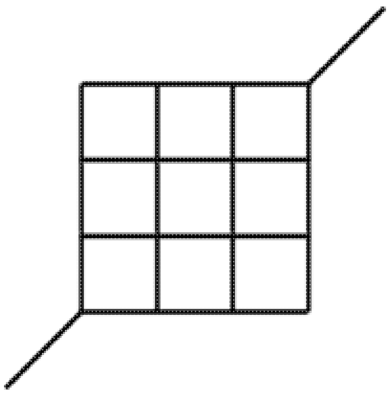
Typical Command Line

```
activitygen --net-file <NET> --stat-file <STATISTICS> --output-file <ROUTES> --abs-rand
```

<NET> is a map in form of a SUMO net file, <STATISTICS> contains the description of the population (described below), and <ROUTES> is the generated SUMO routes file (the demand).

You can give ActivityGen a first try with the following example.

Example



While activitygen has been developed mainly to generate traffic demand for larger networks, this example features the small network shown in the image on the right. You can download all the files of this example in the ZIP archive File:Activitygen-example.zip.

To run this example, use the following commands:

```
netconvert --configuration-file activitygen-example.netc.cfg

activitygen --net-file activitygen-example.net.xml \
            --stat-file activitygen-example.stat.xml \
            --output-file activitygen-example.tmp.rou.xml
            --abs-rand

duarouter --net-file activitygen-example.net.xml \
          --sumo-input activitygen-example-tmp.rou.xml \
          --output-file activitygen-example.rou.xml \
          --remove-loops --continue-on-unbuild --repair

sumo --configuration-file activitygen-example.sumo.cfg
```

The first command generates the net file from the node and edge description. The second command generates the routes file (the demand) from a description of the population in the stat file. The generated routes are incomplete though. This will be changed in the coming weeks, but it allows you to choose the router you want for linking departure and destination edges. To use the SUMO's standard Dijkstra algorithm, you need to *repair* the route file executing the third command. Finally, you can run a traffic simulation with the SUMO main executable.

The Statistics File

General information

First of all we need general information over the city.

```
<city>
    <general
        inhabitants="1000"
        households="500"
        childrenAgeLimit="18"
        retirementAgeLimit="65"
        carRate="0.58"
        unemploymentRate="0.05"
        footDistanceLimit="350"
        incomingTraffic="200"
        outgoingTraffic="50"
    />
</city>
```

The meanings of all these attributes are described in the following table:

Attribute Name	Mandatory	Value Type	Description
inhabitants	yes	Integer	Total number of inhabitants
households	yes	Integer	Total number of households (must be between 0.5 and 1 times the number of adults of the city)
childrenAgeLimit	yes	Integer	Age under which people are children. This is the first year of adulthood
retirementAgeLimit	yes	Integer	Age under which people can have children home and work
carRate	yes	Float[0;1]	Probability for an adult to own a car
unemploymentRate	yes	Float[0;1]	Probability for an adult in age of work to be unemployed
footDistanceLimit	yes	Integer	Maximum distance one would go by foot instead of another transportation mean.
incomingTraffic	yes	Integer	Number of people coming everyday into the city for they work
outgoingTraffic	yes	Integer[0,inhabitants]	Number of inhabitants working outside the city

Parameters

These entries are different from the general information element in what they describe. They depend much more of the special context or state of the city (events, behaviour of the population...). They can also be useful for optimization: the mean speed can be observed afterwards; all values can be changed in order to meet validation criteria on the traffic.

```
<city>
    <general ... />
    <parameters
        carPreference="0.50"
        meanTimePerKmInCity="360"
        freeTimeActivityRate="0.15"
        uniformRandomTraffic="0.20"
        departureVariation="120"
    />
</city>
```

The meanings of all these attributes are described in the following table:

Attribute Name	Mandatory	Value Type	Default Values	Description
carPreference	no	Float[0;1]	0.0	Probability that an adult prefers to take

				his car instead of a public transportation mean (when both available)
meanTimePerKmInCity	no	Integer(sec)	360	Estimation of the time needed to drive one kilometer (bird's eye) on the map
freeTimeActivityRate	no	Float[0;1]	0.15	Probability that a given household, a given day, has a free time activity using a car
uniformRandomTraffic	no	Float[0;0.999]	0.0	Proportion of the random traffic demand in the whole traffic demand
departureVariation	no	Float(sec)	0.0	Variance of the normal distribution introduced for slight variations in departure time (human natural variation in relation to schedules)

Population's Age Brackets

In order to distribute the population among households and in the city coherently, we need a precise age distribution of this population.

```
<city>
    <general ... />
    <parameters ... />

    <population>
        <bracket beginAge="0" endAge"4" peopleNbr="1745" />
         ...
        <bracket beginAge="66" endAge"90" peopleNbr="978" />
    </population>

</city>
```

Follows the description of all bracket attributes.

Attribute Name	Mandatory	Value Type	Description
beginAge	yes	Integer	Beginning age of the interval (incl.). If this is not the first bracket, this age is greater or equal to the previous bracket's ending age
endAge	yes	Integer	End age of the interval (excl.). This age is greater to the current bracket's Beginning age

peopleNbr	yes	Integer	Number of people in the interval age of the bracket ([beginAge,endAge)). This is an relative value, it will be normalized with the number of inhabitants of the city

Work Hours

We need to specify the opening and closing hours of all city's work positions.

```
<city>
    <general ... />
    <parameters ... />
    <population> <bracket ... /> ... </population>

    <workHours>
        <opening hour="30600" proportion="0.30" />
         ...
        <closing hour="43200" proportion="0.20" />
         ...
    </workHours>

</city>
```

Here are descriptions of all attributes of opening and closing elements:

Attribute Name	Mandatory	Value Type	Description
hour	yes	Integer(sec)	Possible beginning or ending time of work.
proportion	yes	Float	Proportion of work positions having this time as a beginning (resp. end) hour. It's a relative value: the probability of occurrence is computed by normalizing all opening (resp. closing) proportion values by one.

Population and Work Position Distribution

Now, we have to specify the density of people and work in each street of the city. (by street I meant edges which are even more precise)

```
<city>
    <general ... />
    <parameters ... />
    <population> <bracket ... /> ... </population>
    <workHours> <opening ... /> ... <closing ... /> ... </workHours>

    <streets>
        <street edge="abc123" population="2.5" workPosition="10.0" />
         ...
    </streets>

</city>
```

Here are the corresponding attribute descriptions:

Attribute Name	Mandatory	Value Type	Description
edge	yes	String	Edge's id
population	yes	Float	Number of people per meter street. Relative value (normalized with the total number of inhabitants)
workPosition	yes	Float	Number of work positions per meter street. Relative value (normalized with the total number of city's work demand)

City Gates

In order to generate incoming and outgoing traffic, we need to specify gates to the city. Every gate will generate the same number of incoming (resp. outgoing) cars (the total number divided by the number of gates).

```
<city>
    <general ... />
    <parameters ... />
    <population> <bracket ... /> ... </population>
    <workHours> <opening ... /> ... <closing ... /> ... </workHours>
    <streets> <street ... /> ... </streets>

    <cityGates>
        <entrance edge="abc123" pos="243.67" incoming="1.5" outgoing="2.2"/>
        <entrance edge="abc234" pos="0.00" incoming="1.0" outgoing="0.5"/>
         ...
    </cityGates>

</city>
```

The corresponding attribute descriptions:

Attribute Name	Mandatory	Value Type	Description
edge	yes	String	Edge's id
pos	yes	Float (m)	Exact position in the street (edge), in meters, from the beginning of the given edge (the maximum value is the length of the edge)
incoming	yes	Float	Proportion of the incoming vehicles, coming through this gate (relatively to the incoming values of the other gates)
outgoing	yes	Float	Proportion of the outgoing vehicles, leaving the city through this gate (relatively to the outgoing values of the other gates)

Schools

Children don't go to work but to school. The particularity of schools is that they are exactly positioned and receive many pupils every day.

```
<city>
```

```
    <general ... />
    <parameters ... />
    <population> <bracket ... /> ... </population>
    <workHours> <opening ... /> ... <closing ... /> ... </workHours>
    <streets> <street ... /> ... </streets>
    <cityGates> <entrance ... /> ... </cityGates>

    <schools>
        <school edge="123abc" pos="23.0" beginAge="12" endAge="18" capacity="400" opening="32400"
         ...
    </schools>

</city>
```

The corresponding attribute descriptions:

Attribute Name	Mandatory	Value Type	Description
edge	yes	String	Edge's id
pos	yes	Float (m)	Exact position in the street (edge), in meters, from the beginning of the given edge (the maximum value is the length of the edge)
beginAge	yes	Integer	Age of the youngest pupils of the school (lower bound, included in the bracket of acceptance)
endAge	yes	Integer	Age of children not accepted in the school any more (higher bound, excluded from the bracket of acceptance)
capacity	yes	Integer	Maximum number of pupils accepted
opening	yes	Integer (sec)	Time of class beginning (school time)
closing	yes	Integer (sec)	Time of class ending (home time)

Bus Lines

People have a public bus line system to their disposition. This public transportaion system is described by stations, different bus lines having their corresponding station for both directions and schedules.

```
<city>
    <general ... />
    <parameters ... />
    <population> <bracket ... /> ... </population>
    <workHours> <opening ... /> ... <closing ... /> ... </workHours>
    <streets> <street ... /> ... </streets>
    <cityGates> <entrance ... /> ... </cityGates>
    <schools> <school ... /> ... </schools>

    <busStations>
        <busStation id="1" edge="abc123" pos="456" />
        <busStation id="2" edge="123cba" pos="324" />
         ...
    </busStations>
    <busLines>
        <busLine id="601" maxTripDuration="3000">
            <stations>
                <station refid="1" />
                 ...
            </stations>
            <revStations>
                <station refid="2" />
                 ...
            </revStations>
            <frequencies>
                <frequency begin="10000" end="25000" rate="1500" />
                 ...
            </frequencies>
        </busLine>
    </busLines>

</city>
```

The corresponding attribute descriptions of all elements:

Attribute Name	Mandatory	Value Type	Description
id	yes	String	Bus line's id
maxTripDuration	yes	Integer (sec)	Maximum time needed for a bus to do the end-to-end trip.
refid	yes	Integer	Reference to the id of the chosen station (refers to a busStation object's id)
begin	yes	Integer (sec)	Time of the beginning of a new frequency
end	yes	Integer (sec)	End time of the frequency
rate	yes	Integer (sec)	Time between tow buses, this is the inverse of the mathematical frequency. This bus rate is performed between the beginning and end values described above

Final Aspect

Here is a complete example of a stat file:

```
<city>
    <general inhabitants="45774" households="28200" childrenAgeLimit="18" retirementAgeLimit="65"
        unemploymentRate="0.05" footDistanceLimit="500" incomingTraffic="1500" outgoingTraffic="3

    <parameters carPreference="0.50" meanTimePerKmInCity="360" freeTimeActivityRate="0.15"
        uniformRandomTraffic="0.20" departureVariation="120" />

    <population>
        <bracket beginAge="0" endAge="30" peopleNbr="1765" />
        <bracket beginAge="30" endAge="75" peopleNbr="1290" />
    </population>

    <workHours>
```

```
        <opening hour="30600" proportion="0.30" />
        <opening hour="32400" proportion="0.70" />
        <closing hour="43200" proportion="0.20" />
        <closing hour="63000" proportion="0.20" />
        <closing hour="64800" proportion="0.60" />
    </workHours>

    <streets>
        <street edge="-2779#2" population="4.0" workPosition="2.0" />
        <street edge="-2776#0" population="3.5" workPosition="2.5" />
        <street edge="-2950#15" population="1.3" workPosition="0" />
    </streets>

    <cityGates>
        <entrance edge="-2950#15" pos="0.0" />
    </cityGates>

    <schools>
        <school edge="-2779#2" pos="23" beginAge="12" endAge="18" capacity="400" opening="32400"
        <school edge="-2779#2" pos="23" beginAge="3" endAge="12" capacity="150" opening="30600" c
        <school edge="-2776#0" pos="765" beginAge="0" endAge="6" capacity="245" opening="32400" c
    </schools>

    <busStations>
        <busStation id="1" edge="-2779#2" pos="456" />
        <busStation id="2" edge="-2776#0" pos="324" />
        <busStation id="3" edge="-2950#15" pos="233" />
    </busStations>

    <busLines>
        <busLine id="601" maxTripDuration="3000">
            <stations>
                <station refid="2" />
                <station refid="1" />
            </stations>
            <revStations>
                <station refid="1" />
                <station refid="2" />
            </revStations>
            <frequencies>
                <frequency begin="21600" end="64800" rate="1200" />
                <frequency begin="64800" end="84600" rate="1800" />
            </frequencies>
        </busLine>
    </busLines>
</city>
```

Further Ways to import VISUM Demand Definitions

VISUM stores its demand in OD-matrices which can be imported using OD2TRIPS. Though, it may be interesting for using the computed assignments without performing a dynamic user assignment. The SUMO package contains some scripts which allow to process other VISUM data and are discussed in the following.

Importing Turn Percentages

VISUM can save the defined/computed turning percentages at junctions. The format differs from turning probabilities format used by JTRROUTER. The script ***visum_convertTurnPercentages.py*** converts VISUM turning percentages into JTRROUTERs turning definitions. The tool requires the SUMO-network converted from VISUM, the turning probabilities from VISUM, and the name of the file into which the converted turning probabilities shall be written:

```
visum_convertTurnPercentages.py <SUMO_NET> <VISUM_TURNINGS> <OUTPUT>
```

The script is located in ***<SUMO>***/tools/import/visum. It is written in Python.

Usability

It seems as using turning ratios for large areas would not make any sense. The resulting routes are very unrealistic as they contain many loops.

See Also

- JTRROUTER - page on the turning ratios router
- ImportingWithJtrRouter - further documentation on this tool's usage

Importing Routes

VISUM can save the routes it computes during the assignment. The format differs from Definition of Vehicles, Vehicle Types, and Routes used by SUMO and GUISIM. The script ***visum_convertRoutes.py*** converts VISUM routes into SUMO routes. The tool requires the SUMO-network converted from VISUM, the routes exported from VISUM, and the name of the file into which the converted turning probabilities shall be written. Additional options are shown in the following table:

Option	Description
--net-file *<FILE>* -n *<FILE>*	Read the SUMO-network to map the routes onto from <FILE>; mandatory
--visum-routes *<FILE>* -r *<FILE>*	Read VISUM-routes to map from <FILE>; mandatory
--output *<FILE>* -o *<FILE>*	Write generated routes to <FILE>; mandatory
--begin *<INT>* -b *<INT>*	Define the begin of the interval the vehicles are emitted within [s]; default: 0s
--end *<INT>* -e *<INT>*	Define the end of the interval the vehicles are emitted within [s]; default: 3600s
--prefix *<STRING>* -p *<STRING>*	Add <STRING> as prefix to the IDs of generated vehicles; optional, default: no prefix
--type *<STRING>* -t *<STRING>*	Set the vehicle type to <STRING>; optional, default: no type

--uniform -u	Vehicle departure times will be spread across the interval uniformly; optional, default: false
--timeline *<STRING>* -l *<STRING>*	Applies a daily time line. The time line must be given as a list of 24 floats, each describing the percentage of emissions from the original number for each hour of a day; optional, default: no timeline

Example call:

```
visum_convertRoutes.py -n <SUMO_NET> -r <VISUM_ROUTES> -o <OUTPUT> --uniform
```

The script is located in ***<SUMO>***/tools/import/visum. It is written in Python.

Usability

The routes can be directly used within SUMO / GUISIM.

See Also

- Examples of daily time lines

Further Ways to import Vissim Demand Definitions

Besides using OD-matrices which can be imported using OD2TRIPS as VISUM does, Vissim also allows to define vehicle flows as a combination of in-flows and route decisions. The following tool allows to import these.

vissim_parseRoutes.py

Parses routes stored in the given Vissim file (first parameter) as (in-)flows and route decisions. The read information is saved twice; the read flows are saved as *<OUTPUT_PREFIX>.flows.xml*, and the read routes are saved as *<OUTPUT_PREFIX>.rou.xml*. (Starting?) edges of the route may be renamed by setting them within "edgemap" variable (within the tool).

```
vissim_parseRoutes.py <VISSIM_NETWORK> <OUTPUT_PREFIX>
```

The script is located in ***<SUMO>***/tools/import/vissim. It is written in Python.

vissim_parseBusStops.py

Parses bus stops and bus routes stored in the given Vissim file (first parameter). The read bus lines are saved as *<OUTPUT_PREFIX>_busses.rou.xml*. The read routes are saved as *<OUTPUT_PREFIX>_stops.add.xml*. (Starting?) edges of the route may be renamed by setting them within "edgemap" variable (see below).

```
vissim_parseRoutes.py <VISSIM_NETWORK> <OUTPUT_PREFIX>
```

The script is located in ***<SUMO>***/tools/import/vissim. It is written in Python.

AutomaticRouting

There is another approach to vehicle routing which may be adequate in the following situations:

- there is not enough time / computing power to wait for the dynamic user equilibrium
- changes to the net occur while the simulation is running
- vehicles need to adapt their route while running

In this case SUMO may be used directly for routing with either routes or trip files (or a mix) as input. The options related to this routing are:

Option	Mandatory y/n	Description
--device.routing.probability *<FLOAT>*		The probability for a vehicle to have a routing device
--device.routing.knownveh *<STRING>*		Assign a device to named vehicles
--device.routing.deterministic		The devices are set deterministic using a fraction of 1000
--device.routing.period *<STRING>*		The period with which the vehicle shall be rerouted
--device.routing.pre-period *<STRING>*		The rerouting period before emit
--device.routing.adaptation-weight *<FLOAT>*		The weight of prior edge weights.
--device.routing.adaptation-interval *<STRING>*		The interval for updating the edge weights.
--device.routing.with-taz		Use zones (districts) as routing end points

If the routing is enabled for selected vehicles (either with the ".probability" or the ".knownveh" option), the average travel times in the net are collected for all edges. If a vehicle needs to be routed (either because it gets inserted or because a repeated route choice was enabled via the ".period" option) it chooses the shortest route to its destination edge (or district) according to the present edge weights (travel times). The update of the edge weights does not simply overwrite the old value but gives it a certain weight which may be modified with the ".adaption-weight" option. Since updating the weights of all edges in each simulation step means a major slowdown for the simulation this interval may be altered using the ".adaption-interval" option.

Simulation

Basic Simulation Definitions

In the following, the inputs needed by the simulation modules SUMO and GUISIM are described.

Road Network

For a simulation, a SUMO Road Network must be given using the option --net-file *<NETWORK_FILE>* (or -n *<NETWORK_FILE>*). The network is normally built using NETCONVERT or NETGEN.

Traffic Demand

The vehicles to simulate must be given. Their description normally includes vehicle types, vehicles, and vehicle routes. Routes are normally given to the simulation modules using the option --route-files *<ROUTES_FILE>*[,*<ROUTES_FILE>*]* (or -r *<ROUTES_FILE>*[,*<ROUTES_FILE>*]*).

Defining the Time Period to Simulate

Each simulation requires the definition about the time period to be simulated. This is given to SUMO or GUISIM using the options --begin *<INT>* (or -b *<INT>* for short) and --end *<INT>* (-e *<INT>*). Please note that whether the option --end was given influences the simulation's behavior. The details are described below.

The simulation starts at the time given in --begin, which defaults to 0. All vehicles with a departure time (depart) lower than the begin time are discarded.

The simulation performs each time step one-by-one.

The simulation ends in the following cases:

- The final time step was given using --end and this time step was reached (time after a step is >= end).
- No value for --end has been given and all vehicles have been simulated. The state of the simulation is the one in which the last vehicle has left the simulated area. If a TraCI connection is active, the simulation will continue even after the last vehicle (potentially "forever").
- A close command has been received via TraCI

There are two more command line options influencing ending of the simulation:

- --too-many-vehicles *<INT>* Quit simulation if the given number of vehicles is exceeded

---

Changes

- before 7.5.2009 the end step was assumed to be inclusive - was simulated, see http://apps.sourceforge.net/mediawiki/sumo/index.php?title=SUMO_BasicDefinition&oldid=985

Advanced Traffic Lights Simulation

Normally, NETCONVERT and NETGEN 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 real traffic light programs, it is possible to load additional programs. Also, SUMO/GUISIM allow to load definition which describe when and how a set of traffic lights can switch from one program to another. Both will be discussed in the following subchapters.

Loading new TLS-Programs

You can load new definitions for traffic lights as a part of additional files. When loaded, the last program will be used. Switching between programs is possible via WAUTs and/or TraCI. Also, one can switch between them using the GUI context menu. A definition of a traffic light program looks like this:

```
<tl-logic id="0" programID="my_program" offset="0" type="static">
   <phase duration="31" state="GGggrrrrGGggrrrr"/>
   <phase duration="5"  state="yyggrrrryyggrrrr"/>
   <phase duration="6"  state="rrGGrrrrrrGGrrrr"/>
   <phase duration="5"  state="rryyrrrrrryyrrrr"/>
   <phase duration="31" state="rrrrGGggrrrrGGgg"/>
   <phase duration="5"  state="rrrryyggrrrryygg"/>
   <phase duration="6"  state="rrrrrrGGrrrrrrGG"/>
   <phase duration="5"  state="rrrrrryyrrrrrryy"/>
</tl-logic>
```

The following attributes/elements are used:

Attribute Name	Mandatory	Value Type	Description
id@tl-logic	y	id (string)	The id of the traffic light
type	y	enum (static, actuated, agentbased)	The type of the traffic light
programID	y	id (string)	The id of the traffic light program; Please note that "off" is reserved, see below.
offset	y	int	The initial time offset of the program

Each phase is defined using the following attributes:

Attribute Name	Mandatory	Value Type	Description
duration@phase	y	time (int)	The duration of the phase
state@phase	y	list of signal states	The traffic light states for this phase, see below

Each character within a phases' state describes the state of one signal of the traffic light. Please note, that a single lane may contain several signals - for example one for vehicles turning left, one for vehicles which move straight. This means that a signal does not control lanes, but links - each connecting a lane which is incoming into a junction and one which is outgoing from this junction. In SUMO, a one-to-n dependency between signals and links is implemented, this means each signal may control more than a single link - though networks generated by NETCONVERT or NETGEN usually use one signal per link. Please note also, that a traffic light may control lanes incoming into different junctions. The information about which link is controlled by which traffic light signal may be obtained using the "show link tls index" option within GUISIM's visualisation settings or from the according linkno@succlane attribute of the network.

The following signal colors are used:

Character	Description
r	'red light' for a signal - vehicles must stop
y	

	'amber (yellow) light' for a signal - vehicles will start to decelerate if far away from the junction, otherwise they pass
g	'green light' for a signal, no priority - vehicles may pass the junction if no vehicle uses a higher priorised foe stream, otherwise they decelerate for letting it pass
G	'green light' for a signal, priority - vehicles may pass the junction

After having defined a tls program as above, it can be loaded as an additional file; of course, a single additional file may contain several programs. It is possible to load several programs for a single tls into the simulation. The program loaded as last will be used (unless not defined differently using a WAUT description). All subkeys of the additional programs must differ if they describe the same tls.

It is also possible to load a program which switches the tls off by giving the programID the value "off".

```
<tl-logic id="0" type="static" programID="off"/>
```

Tools for TLS programs import

Description from real-world traffic light systems do not arrive us in form of SUMO-traffic light descriptions normally. For an easier import than editing them by hand, a tool named "tls_csv2SUMO.py" exists which parses a csv-file which describes the program and builds an according SUMO-traffic light description. The tool can be found in ***<SUMO>***/tools/tls. This tool requires the program definition and the SUMO-network it shall be converted to:

```
tls_csv2SUMO.py <TLS_CSV> <NET>
```

It prints the generated TLS definition on stdout (you can pipe it to a file).

The format of the CSV description is as following. At first, three header lines must be given, which name the tls, the program's subkey, and the offset at which the program shall start in seconds - normally 0:

```
key;<KEY>
subkey;<SUBKEY>
offset;<OFFSET>
```

Then, it is defined which signal (number) in the program is responsible for which link. The link may be described either using the incoming edge only, or the incoming lane only, or also incorporating the outgoing edge/lane. This means the following lines are all valid:

```
link;<LINK_NUMBER>;<FROM_EDGE>;;0
link;<LINK_NUMBER>;<FROM_LANE>;;0
link;<LINK_NUMBER>;<FROM_EDGE>;<TO_EDGE>;0
link;<LINK_NUMBER>;<FROM_EDGE>;<TO_LANE>;0
link;<LINK_NUMBER>;<FROM_LANE>;<TO_EDGE>;0
link;<LINK_NUMBER>;<FROM_LANE>;<TO_LANE>;0
```

It is also possible to assign more than one link to a single signal.

Then, the signals' states are given:

```
<LINK_NUMBER>;<STATES>
```

The states are encoded using the signal colors (please note that only lower-case letters are used, see below) described above, separated by ';'. An example signal phase definition (for signal 1) could be:

```
1;g;g;g;g;y;r;r;r;r;r
```

Please not that the number of states must be the same for all defined links.

Now, we only have to define the phase times:

```
time;<TIMES>
```

The times are given in seconds, again separated using ';'. An example could be:

```
time;18;33;3;6;3;3;9;15;90
```

A complete CSV-description could look like:

```
key;102
subkey;utopia
offset;0
link;1;4643;;0
link;1;3078;;0
link;2;3074;;0
link;2;-6494;;0
1;g;g;y;r;r;r;r;r
2;r;r;r;r;g;g;y;r
3;r;r;r;r;g;y;y;r
4;g;y;y;r;r;r;r;r
min;21;3;3;2;20;9;3;2
time;45;3;3;2;36;9;3;2
max;78;3;3;2;62;9;3;2
```

Defining Program 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. It is possible to load a definition of switch times between the programs using a WAUT (short for "Wochenschaltautomatik" ~ weekly switch automatism).

Given 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, and already defined programs, named "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:

Attribute Name	Mandatory	Value Type	Description
id	y	string id	The name of the defined WAUT
refTime	y	int	A reference time which is used as offset to the switch times given later (in simulation seconds)
startProg	y	string id	The program that will be used at the simulation's begin

and the fields in wautSwitch:

Attribute Name	Mandatory	Value Type	Description
time	y	int	The time the switch will take place
to	y	string id	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 switch steps must be sorted by their execution time.

Additionally, a definition about which tls shall be switched by the WAUT must be given, as following:

```
<wautJunction wautID="myWAUT" junctionID="RCAS" [procedure="Stretch"] [synchron="t"]/>
```

Here, the attributes have the following meaning:

Attribute Name	Mandatory	Value Type	Description
wautID	y	string id	The id of the WAUT the tls shall be switched by
junctionID	y	string id	The name of the tls to assign to the WAUT
procedure	y	string enum	The switching algorithm to use; If none is given, the programs will switch immediately (default)
synchron	y	string enum	Additional information whether the switch shall be done synchron (default: false)

It is possible to assign several tls to a single WAUT. It is also possible to assign several WAUTs to a single junction in theory, but this is not done in reality.

Evaluation of Traffic Lights Performance

Tools for automatic Detector Generation

Some tools are available which help generating detector definitions for the evaluation of traffic lights. All are located in ***<SUMO>***/tools/output.

- ***generateTLSE2Detectors.py*** generates a file which includes areal detectors. All lanes incoming into an intersection are covered with these detectors. The offset from the intersection may be given using the option --distance-to-TLS *<FLOAT>* (or -d *<FLOAT>*), the default is .1m. The generated detectors end either after a given length, defined using --detector-length *<FLOAT>* (or -l *<FLOAT>*) where the default is 250m, or at the lane's end if the lane is shorter than this length.
- ***generateTLSE3Detectors.py*** generates a file which includes multi-entry/multi-exit detectors. Detectors are built for each edge incoming to the traffic light. All lanes of each of these edges are covered with exit points. These point's offset from the intersection may be given using the option --distance-to-TLS *<FLOAT>* (or -d *<FLOAT>*), the default is .1m. The incoming edges are followed upstream, either until a given length, defined using --detector-length *<FLOAT>* (or -l *<FLOAT>*) where the default is 250m, or another traffic light is reached or no further upstream edge exists. Entry points are generated at these points.

In both cases, the network must be given using the option --net-file *<FILE>* (or -n *<FILE>*). The file including the detector definitions to generate may be given using the option --output *<FILE>* (or -o *<FILE>*), default is "e2.add.xml" for areal detectors, and "e3.add.xml" for multi-entry/multi-exit detectors. Per default, the areal detectors generated by ***generateTLSE2Detectors.py*** are writing their measures to "e2output.xml", the multi-entry/multi-exit detectors generated by ***generateTLSE2Detectors.py*** to "e3output.xml". The output file name can be changed for both scripts using the option --results-file *<FILE>* (or -r *<FILE>*). The frequency of generated reports is 60s per default. It can be changed using the option --frequency *<INT>* (or -f *<INT>*).

SUMO PublicTransport

Simulation of Public Transport

It is possible to 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: <busStop id="<BUS_STOP_ID>" lane="<LANE_ID>" startPos="<STARTING_POSITION>" endPos="<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:

Attribute Name	Mandatory	Value Type	Description
id	y	id (string)	The name of the bus stop; must be unique
lane	y	string-id	The name of the lane the busstop shall be located at
startPos	y	float	The begin position on the lane (the lower position on the lane) in meters
endPos		float	The end position on the lane (the higher position on the lane) in meters
line		string list	A list of names separated by spaces (' ') meant to be the names of the bus lines that stop at this bus stop. This is only used for visualisation

			purposes.

Caution:
Please note that bus stops must be added to a config via the *--additional-files* parameter
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>/examples/sumo/busses):

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

    <vehicle id="0" type="BUS" depart="0" color="1,1,0">
        <route edges="2/0to2/1 2/1to1/1 1/1to1/2 1/2to0/2
             0/2to0/1 0/1to0/0 0/0to1/0 1/0to2/0 2/0to2/1"/>
        <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 is defined 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 short definition of a vehicle's stop is: <stop bus_stop="<BUS_STOP_ID>" | (lane="<LANE_ID>" endPos="<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. For a complete list of attributes for the "stop"-element of a vehicle see Specification#Stops.

SUMO VariableSpeedSigns

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: <variableSpeedSign id="<VSS_ID>" lanes="<LANE_ID>[<LANE_ID>]*" file="<DEF_FILE>"/>.

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' may be found in the SUMO directory under "examples/sumo/variable_speed_signs".

SUMO Rerouter

Rerouter

Rerouter change the route of a vehicle as soon as the vehicle moves onto a specified edge.

A rerouter is set into the simulated network by adding the following declaration line to an "additional file": <rerouter id="<REROUTER_ID>" edges="<EDGE_ID>[;<EDGE_ID>]*" file="<DEFINITION_FILE>" [probability="<PROBABILITY>"]/>. Rerouter may be placed on several edges, at least one edge is necessary. Furthermore, it is possible to define the probability for rerouting a vehicle by giving a number between 0 (none) and 1 (all) already within the definition. The declaration values are:

Attribute Name	Mandatory	Value Type	Description
id	y	id (string)	The id of of the rerouter
edges	y	float	An edge id or a list of edge ids where vehicles shall be rerouted
file	y	float	The path to the definition file
probability		float	The probability for vehicle rerouting (0-1)

In addition to this declaration a definition file (stored in <DEFINITION_FILE>) must be given which describes the behaviour of the rerouter over time. Each description 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 within the rerouter's definition file in detail.

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 edges-attribute of the rerouter's declaration. a closing_reroute 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:

Attribute Name	Mandatory	Value Type	Description
id	y	id (string)	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 that pass one of the edges defined in the edges-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 new route will be the fastest route in the empty network.

The attributes used within a dest_prob_reroute are:

Attribute Name	Mandatory	Value Type	Description
id	y	id (string)	The id of the new destination; mandatory string, the id must be the id of an edge within the network
probability	y	float	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 (but this is not necessary)

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 edges-attribute of the rerouter's declaration. In this case, the id of a complete route must 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:

Attribute Name	Mandatory	Value Type	Description
id	y	id (string)	The id of a new route to assign; mandatory string, the id must be the id of a previously loaded route
probability	y	float	The 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

SUMO Output

Output generated (optionally) be the Simulation

The number of possible outputs that are available in SUMO is enormous. Below, the available outputs are listed, joined into groups of topic/aggregation type. Further information about each output can be found by following its link.

- vehicle-based information, unaggregated
    - raw vehicle positions dump: all vehicle positions over time
      contains: positions and speeds for all vehicles for all simulated time steps
      used for: obtaining movements of nodes (V2V, for ns-2)
    - vehicle type probe: positions of vehicles over time for a certain vehicle type (or all vehicles)

- simulated detectors
    - e1-detectors: simulated induction loops
    - e2-detectors: lanearea detectors - simulated vehicle tracking cameras
    - e3-detectors: simulated entry/exit detectors

- values for edges or lanes
    - edgelane traffic: edge/lane-based network performace measures
    - edgelane hbefa: edge/lane-based vehicular pollutant emission; based on the HBEFA database
    - edgelane harmonoise: edge/lane-based vehicular noise emission; based on Harmonoise

- vehicle-based information
    - trip information: aggregated information about each vehicle's journey
    - vehicle routes information: information about each vehicle's routes over simulation run

- simulation(network)-based information
    - emissions statistics: information about the current state of the simulation

- traffic lights-based information
    - traffic light states: information about the state (lights) of a traffic light
    - stream-based traffic light switches: information about the switches of a traffic light signal responsible for a certain link
    - traffic light states, by switch: information about the states (lights) of a traffic light signal, written only when changed
    - e2-detectors coupled to tls: simulated vehicle tracking cameras triggered by tls

TraCI

Introduction to TraCI

TraCI is the short term for "**Tra**ffic **C**ontrol **I**nterface". The basic idea behind is to give access to a running road traffic simulation. If you have any questions or helpful suggestions, you may use the [sumo-devel] mailing list.

SUMO startup

TraCI uses a TCP based client/server architecture to provide access to SUMO. Thereby, SUMO acts as server that is started with additional command-line options:

```
--remote-port <int>   The port SUMO listens on for incoming connections
--penetration <float> The fraction of vehicles, that should be accessed via TraCI.
                      This value has to be between 0 and 1. It defaults to 1.
```

When started with the `--remote-port` switch, SUMO only prepares the simulation and waits for an external application, that takes over the control.

To let sumo run a set up phase, the following command line parameters can be used:

```
--begin <int> Sumo will run up to this time in seconds, before listen for a TraCI-Client
--end <int>   When reaching this time in seconds, sumo closes
--step-length <float> Length of each simulation step in seconds, its default value is 1, e.g. to
```

Please note, that the `--begin` option results in an offset between the simulation times of sumo and the TraCI-Client, i.e., time=0 on TraCI-Client side means time=begin_time in sumo.

Protocol specification

Please see the TraCI Protocol Specification (including Basic Flow, Messages, Data Types).

TraCI Commands

There are two "generations" of **TraCI** commands. The first one mainly uses an internal mapping between the string-typed IDs used in SUMO and an external representation of these which is int-based. The mapping is done internally (within **TraCI**).

This was found to raise different problems. Additionally, it bloats the code and it is assumed to be not a part of a traffic simulation to map the IDs between an internal and an external representation. Due to this, the APIs are moved towards a "second generation" which uses string-IDs equal to those SUMO reads. The "drawback" in the case you couple SUMO with ns-2 which uses integer ids, you have to translate them somewhere. Nonetheless, it is assumed that translation of IDs between simulators is not a duty of the simulators themselves - independent which simulators you couple. Instead, this should be done by a "middleware" instance "in between" those.

We highly advice to use the second generation APIs. If needed, the mapping should be done by a middleware. We will move the functionalities of the first generation to the second.

Note:
For those still struggling with revision 5499 just for use it with TraNS: you are limited to the old API.

- Control-related commands: perform a simulation step, close the connection

Using internal Mapping

The following APIs use an internal mapping between the IDs read by SUMO and the ones used during the transmission.

- Mobility-related commands set a maximum speed of a vehicle, stop a vehicle, slow a vehicle down, force lane changing, set a new route, set a new destination edge
- Subscription-related commands subscribe for retrieval of an object's values
- Environment-related commands convert positions between different formats, compute routes, ask for values of objects within the scenario
- Traffic Lights retrieve information about the status of a traffic light

Using plain SUMO IDs

For the following APIs, the ID is equal to the ID defined in SUMO's input files. Here, you find their general structure.

- Artifact-based Value Retrieval
    - Induction Loop Value Retrieval retrieve information about induction loops
    - Multi-Entry/Multi-Exit Detectors Value Retrieval retrieve information about multi-entry/multi-exit detectors
    - Traffic Lights Value Retrieval retrieve information about traffic lights
    - Lane Value Retrieval retrieve information about lanes
    - Vehicle Value Retrieval retrieve information about vehicles
    - Vehicle Type Value Retrieval retrieve information about vehicle types
    - Route Value Retrieval retrieve information about routes
    - PoI Value Retrieval retrieve information about points-of-interest
    - Polygon Value Retrieval retrieve information about polygons
    - Junction Value Retrieval retrieve information about junctions
    - Edge Value Retrieval retrieve information about edges
    - Simulation Value Retrieval retrieve information about the simulation

- State Changing
    - Change Lane State change a lane's state
    - Change Traffic Lights State change a traffic lights' state
    - Change Vehicle State change a vehicle's state
    - Change PoI State change a point-of-interest's state (or add/remove one)
    - Change Polygon State change a polygon's state (or add/remove one)
    - Change Edge State change an edge's state
- TraCI/Value Retrieval Subscription
- TraCI/Add Vehicle

TraCI/Vehicle Signalling explains which signals, etc. are modeled and can be retrieved.

Example use

- There is a tutorial on using TraCI for adaptive traffic lights (using Python).
- The CityMobil example uses TraCI for assigning new routes to vehicles (using Python).

Resources

- Axel Wegener, Michal Piorkowski, Maxim Raya, Horst HellbrÃ¼ck, Stefan Fischer and Jean-Pierre Hubaux. TraCI: A Framework for Coupling Road Traffic and Network Simulators. Proceedings of the 11th Communications and Networking Simulation Symposium, April 2008. Available at ACM Digital Library
- Axel Wegener, Horst HellbrÃ¼ck, Christian Wewetzer and Andreas LÃ¼bke: VANET Simulation Environment with Feedback Loop and its Application to Traffic Light Assistance. Proceedings of the 3rd IEEE Workshop on Automotive Networking and Applications, New Orleans, LA, USA, 2008. Soon available at IEEEXplore
- TraCI Server in Sumo is accessible via SUMOs source SVN. The relevant sources are located in the folder `src/traci-server`.

Interface Implementations

- Python: traciControl.py allows to interact with SUMO using Python

Implementation Status

- Other TraCI Servers aside from SUMO
    - **AnSim**: On request: TraCI server, that enables the Ad-Hoc Network Simulator AnSim to act as a mobility generator. Ansim includes traditional mobility models like random waypoint.
- TraCI Clients
    - **ns2**: The ns2 patch containing the TraCI client is now available for download. You can also checkout the anonymous readable SVN repository at `https://svn.itm.uni-luebeck.de/traci/trunk/ns2-patch`.
    - **Shawn**: TraCI client is included in the algorithmic centered sensor network simulator Shawn.
    - **OMNeT++** 3.4b2 / INET 20061020: SUMO Traffic Control Interface (TraCI) modules for OMNeT++ 3.4

- ♦ **OMNeT++** 4.0rc2 / INET Framework: SUMO Traffic Control Interface (TraCI) modules for OMNeT++ 4.0 / INET
- ♦ **OMNeT++** 4.1b3 / MiXiM Framework: SUMO Traffic Control Interface (TraCI) modules for OMNeT++ 4.0 / MiXiM
- ♦ **JiST / SWANS** 1.0.6: SUMO Traffic Control Interface (TraCI) modules for JiST / SWANS 1.0.6

The table below reflects the current implementation status of TraCI commands in the listed servers and clients.

command	SUMO (release 0.10.1)	SUMO (latest svn)	ns2	Shawn	OMNeT++	JiST/SWANS
Simulation Step	X	X	X	O	X	X
Set Maximum Speed	X	X	X	O	X	X
Stop Node	X	X	O		O	O
Change Route	X	X	X		X	
Change Target	X	X				
Change Lane	X	X				
Slow Down	X	X				
Position Conversion	X	X				
Distance Request	X	X			O	
Scenario	X	X				
Subscribe Lifecycles	O	O			X	X
Subscribe Domain	O	O			X	X
Add Vehicle		O			X	

- O = command is implemented, but was not tested yet
- X = command is implemented and was tested (however, there might still be undiscovered errors or side effects)

Why Vehicles are teleporting

When running a simulation, one may encounter the following warning:

```
Warning: Teleporting vehicle '...'; waited too long, lane='...', time=....
```

What does it mean?

Reasons

The following circumstances may force the simulation to "teleport" a vehicle:

- the vehicle stood too long in front of an intersection (message: "*...'; waited too long, lane='...*")
- the vehicle has collided with his leader (message: "*...'; collision, lane='...*")
- the vehicle has not left the lane but the one behind did (message: "*...'; false leaving order, lane='...*")
- the vehicle landed beyond it's lane length (message: "*...'; beyond lane (X), lane='...*")

Waiting too long, aka Grid-locks

Grid-locks, jamming a simulated scenario are unfortunately something normal to traffic simulations. You can solve this only by assigning different routes. Some further approaches may be invoked in the future to make them occure more seldom.

In the case a vehicle is standing at the first position in front of an intersection, SUMO counts the number of steps the vehicle's velocity stays below 0.1m/s. These steps are the "waiting time". In the case the vehicle moves with a larger speed, this counter is resetted. In the case the vehicle waited longer than a certain swell, the vehicle is teleported, assuming a grid-lock occuren on the intersection.

The swell can be modified using a command line option. The option is --time-to-teleport *<INT>*. In the case a value lower than 0 is given, no teleportation is done, otherwise the given value will be interpreted as seconds to wait before teleporting a vehicle.

Also, besides plain grid-locks, the imperfection of the lane-change model sometimes yields in a situation where two vehicles try to get to the other lane, and each vehicle is blocking the other one. The simulation behaves as described in prior. An additional possibility to "solve" this is to allow vehicles to be swapped - they are exchanged. To enable this possibility, use the option --lanechange.allow-swap.

Collisions

Though SUMO uses a collision-free model, collisions have beed detected. As they yield in an undefined state of the simulation, a vehicle teleportation is performed for solving them.

Note:
Collisions are assumed to occur due to bugs in the simulation. Please report if you encounter one.

False Leaving Order

A vehicle has driven over his leader and wants to get to the next lane, though his leader has not yet left his one.

Note:
Leaving the lane in a false order is assumed to occur due to bugs in the simulation. Please report if you encounter one.

Landing beyond the Lane's End

This can only occur if a vehicle's right-of-way has changed from what the vehicle assumed. An example: the vehicle though it could pass a junction, but a second vehicle (surprisingly) disallows it from passing the junction. In order not to brake with a value larger than the vehicle's deceleration ability, the vehicle continues his ride, but is not able to enter the next lane - in some cases this yields in a position beyond the lane's end.

Note:
Landing beyond the Lane's End is assumed to occur due to bugs in the simulation. Please report if you encounter one.

What's happening

A teleported vehicle is removed from the network. It is then moved along its route, but no longer being on the street. It is reinserted into the network as soon as this becomes possible. While being teleported, the vehicle is moved along its route with the average speed of the edge it was removed from or - later - it is currently "passing". The vehicle is reinserted into the network if there is enough place to be placed on a lane which allows to continue its drive.

Note:
Please note that up to version 0.12, the teleporting speed was higher and that vehicles were reinserted into the network if there was free place on any of the lanes of the edge the vehicle was "above".

Simulation Loop

Each simulation time step, the following procedures are performed. Please note that this is description for the current (since 12.01.2010) SVN version; earlier versions differ.

- Process TraCI Commands (if TraCI is not disabled)
- Save Simulation State if needed (currently only in mesosim)
- Execute "Begin of Time Step" Events
- Check for Collisions
- Reset Junction Requests
- Check Traffic Light Changes
- Set Traffic Light Signals on Links
- Assure all not-free Lanes are known
- Compute Vehicle Movement (iterating over all not-free lanes)
- Determine which Vehicles have Right-of-way
- Move Vehicles
- Change Lanes
- Check for Collisions
- Progress Vehicles Loading
- Emit Vehicles
- Execute "End of Time Step" Events
- Save Output
- Increment current Time Step

Compute Vehicle Movement (iterating over all not-free lanes)

A vehicle's (**EGO'**s) speed during the next time step is constrained by: a) the leading vehicle (**LEADER**), b) a vehicle on the lane left to **EGO'**s lane in some cases, c) right-of-way rules at approached intersections, d) speed limits at approached lanes.

Interaction with the leading Vehicle

Using a car-following model, and having an **EGO** vehicle, we need its **LEADER** which parameter are used within the model's equations. Its pretty easy to determine one on a circular road, but things get more complicated when moving to real-world road networks:

- we have to consider leading vehicles on the next lanes that may be approached

Interaction with LEADER on same lane

We go through the (sorted) list of a lane's vehicles and use the next one as **LEADER**. This is the first check in computing the next speed. We do this only for the next vehicle in front (**LEADER**), discarding further vehicles in fron (**LEADER+n**) - assuming the **LEADER'**s model would be responsible for keeping **EGO** collision free by using a safe velocity in respect to his (**LEADER'**s) **LEADER**.

Obviously, the first vehicle on the lane (the one nearest to the lane's end) does not have a leading vehicle on this lane. Now, one assumption is that if a vehicle is on a lane which does not allow to continue its route, it brakes so that it does not leave it. This means that for the last vehicle we do not need a **LEADER** if this last vehicle is on a false lane. Nonetheless, an additional problem may occur for the last vehicle: a vehicle which is only partially at the lane; vehicles are administrated by the lanes and the lane stores and moves only vehicles which front is on the lane - the vehicle's back may nonetheless be (still) on a predecessing lane. We have to check this even in the case a vehicle will not leave the lane.

Interaction with LEADER on consecutive lanes

In the case **EGO** is on a "correct" lane, we have to look into consequent lanes for **LEADER**s. The reason is quite simple: assume **EGO** is turning left, but his **LEADER** - being on the same lane - goes straight and has no **LEADER** itself. The **LEADER** may progress at high speed, but **EGO** has to watch out for the situation on subsequent lanes.

Advices on selected Research Topics

TopicVehicleRerouting

Vehicle Rerouting

Within the next subchapter, we want to try to give some advices on how investigations on vehicle rerouting should be prepared and evaluated. Though most of these are based on prior work, any further ideas or processes and other information is appreciated.

Basic Description

The research descibed in here deals with methods for computing vehicle routes, mainly in the case of unforeseenable events. Normally, we assume traffic is in an "equlibrium state" - a route change does not yield in a better performance of the road network. Nonetheless, the traffic may change due to:

- accidents and resulting jams
- additional event traffic
- street closures
- changing destinations (new shopping center)

In such cases, certain edges of the road network will be either completely unsuable or the time needed to pass them raises, either directly, or because of the additional demand of vehicles that want to pass them.

Things to keep in Mind

- **The simulated traffic should be in equilibrium mode before the incident is introduced**

  Explanation: Using fastest routes in an empty network, like obtained when using a simple Dijkstra (done when converting trips to routes via DUAROUTER) is not realistic. In this case, all vehicles would use the fastest edges *in an empty network*, which would yield in jam situation on these edges. I such cases, it is easy to find a new best route during simulation - but the initial situation would be not realistic. Having a proper traffic assignment is needed for realistic evaluations.

Evaluation Advices

TrafficOnline

Missing:
This text is currently under work

GSM-based Surveillance

TrafficOnline (1) is a traffic surveillance system based on tracking GSM cellular phones located in vehicles. This system was developed by a consortium within the "Verkehrsmanagement 2010" initiative. One of TrafficOnlineâ s sub-topics dealt with a simulation-based evaluation of the system. A simulation was used because it allows modeling and reproduction of certain traffic situations at well-defined time and space. It was decided to use SUMO for this purpose due to its availability as open source what allows to extend it easily.

Below, it is described what steps have been done to perform the simulation. The telephony model originally implemented in SUMO was removed from the code after a while.

How Vehicles can be tracked using GSM

GSM-telephony takes place within a network of stationary cells and each "connected", meaning currently used, cellular phone is assigned to a "serving cell", mostly the cell with the best connection to the phone. Additionally, a list of the next best serving cells is hold within the phone. Evaluations done during the TrafficOnline project have shown that one can find distinct combinations of these listâ s entries at certain areal positions when the quality of each connection between a cellular phone and a cell is taken into account. These additional points which allow determining a cellular phoneâ s position will be called *TOL-points* in the following.

Especially in the case of a moving call, so-called "handovers" of calls between two cells get necessary when the call moves from one serving cell to another one. Handovers do not take place for unused phones. Used and unused phones are assigned to a "location area" (LA), a structure made up of several GSM-cells. Due to the larger size of location areas in comparison to cells, handovers of unused cellular phones are more seldom than those of used ones.

Handovers at cell- and LA-boundaries are tracked by the GSM-system. Additionally, it is possible for the system to obtain and store the list of best serving cells of each cellular phone. All this information allows tracking a moving, cellular phone through the network at certain positions, made up of the cell and la-boundaries and of TOL-points. The density of these positions is larger for used phones, where not only crossing of LA-boundaries can be observed by the system. When projected on the road network, one can compute traveling times for a cellular phone between the positions knowing their distances. Additional information about the GSM-network and how the TOL-systems works can be found in (2, 3).

Usage of SUMO in TrafficOnline

Five areas located in Berlin, Germany, as shown in figure 1, were chosen for which partners supplied information about road traffic amounts by means of induction loop and infrared sensor data, and about the utilization of the corresponding GSM network in the form of log-files containing anonymous data of calls. The areas were chosen in order to cover a large-most number of different road types and possible disturbances. For these areas, the locations of the corresponding TOL-points were prepared, encoded into geo-coordinates, and assigned to the road sections.

Traffic Simulation

The digital road networks for the specified areas were extracted from a digital road map bought from NavTeq (6). Only those roads near to those concerned by TOL-points were extracted for the simulation, other roads were discarded. The resulting maps were converted into the SUMO-format using a tool from the SUMO-package, NETCONVERT. This tool automatically computes right-of-way rules at junctions and the necessary connections between lanes across the junctions. Both are missing within most digital networks. After this prior conversion, the obtained SUMO-networks were compared with satellite photos from GoogleEarth (7) in order to find mismatches in the number of lanes between digital and real roads and to locate junctions equipped with traffic lights. The original digital road data was adjusted in order to eliminate the mismatches found using GoogleEarth. Afterwards the conversion was redone. Finally, further work was done in order to eliminate falsely computed connections between lanes and to make the simulated traffic lightsâ programs fit more to reality.

The demand of an average day for each of the areas was modeled using point count data collected between June and August 2006 at weekdays from Tuesday to Thursday. Where possible, within highway areas where induction loops are located at all on- and off-ramps, vehicle routes could be built automatically using a tool from the SUMO-package named DFROUTER. This tool imports count data, determines whether a detector may be used as a source, sink, or none of both for the given area, and calculates the probabilities for using routes between each sink/source pair. In addition, DFROUTER can compute vehicle flows which shall be inserted at source positions in a format the simulation can read. In those scenarios in which sensors were less dense in reality, the routes and their distributions were built by hand. The traffic amount for average traffic was implemented into the simulation using so-called "emitters". In SUMO, these simulation structures can be placed at a certain position of a lane and can insert vehicles into the network. Each vehicle is inserted at a defined time. In our case these vehiclesâ routes were assigned from the previously computed route distributions. The vehiclesâ types were assigned to the vehicles using a vehicle type distribution. The flows used by the emitters were computed using DFROUTER in all cases.

Where possible, count data which was not used for emitting vehicles was used to validate the simulation. Figure 2 shows a set of comparisons between flows from real life and the simulation across a real and a simulated induction loop. A comparison of simulated and real speeds was not possible, because they were not in the provided sensor data set.

Simulation of Cellular Telephony

In order to keep the simulation fast, no attempt was done to determine the correct serving cell of a device by using the vehicleâ s position. Instead, the cell boundaries of real-world cells, given as a set of GIS-polygons, were laid over the simulated road network to get the positions both cross. At the resulting intersections, so called "cell-actors" were placed within the simulation. When a simulated cellular phone passes such a "cell-actor", it is informed about having entered a new cell and both the prior and the new serving cell are informed about the phoneâ s cell change. If a simulated phone in connected mode enters a cell or if a phone starts a call within a cell, this cell increments one of the internal "moving call" counters â either the one for incoming or the one for outgoing calls. Besides this, the information about the duration each phone was within the cell is computed, too. This information is used for calibrating the telephony model, as shown later on. The applied approach assumes that stationary calls are first of all made by slowly moving pedestrians or not moving occupants of the area. Due to this assumption and the fact that it is not yet possible to simulate single persons with SUMO, the stationary calls are not generated by the simulation using a telephony model. Instead, the amounts of stationary calls, again split into incoming and outgoing calls, are once extracted from real-life data and later passed to the simulation. This is done for each cell in intervals of 5 minutes. These statistics are joined with the moving calls generated by the simulation when writing the output. In combination with the number of simulated in-vehicle devices this information represents the "COLLECTORCS" output. The TOL-points are modeled similar to the cell crossings. "TOL-actors" are placed on the road network at the positions of the TOL-points. They save the call id of each cellular phone that crosses them in â connected modeâ together with the id of the ROL-point and the time the vehicle has crossed it. This information is immediately saved into the data table "COLLECTORPOS", by what this output need by the TrafficOnline algorithm is completely implemented.

When starting their trips, the simulated in-vehicle cellular phones do not know the cell they are located in because they have not yet passed a "cell-actor". Their state is set to idle initially. As soon as a simulated cellular phone is assigned to a cell, it starts asking the simulation whether to switch into the connected mode, what is repeated every step of the simulation. In order to get the probability to start a call, a simulation of a normal day was performed at first for each of the simulated areas. Within these simulation runs, every vehicle was equipped with exactly one cellular phone device. The previously mentioned durations a device was located in the area were summed up over an interval of 30 minutes. This sumâ s unit is vehicle-seconds per 30 minutes. Based upon this value and the mean number of moving calls within the same period retrieved from real-life data, the probability to start a call within a cell i has been computed for a given time interval using the following formula:

(1)

where is the probability to start a call within cell i within the interval, is the number of measured cellular calls within this cell and interval, and is the duration vehicle j was within cell i within the interval.

The information about the GSM traffic (amount of calls, changeovers etc.) was not provided for every cell of all the simulation areas. Furthermore, it was assumed that there are only minor differences between the parameters of the telephony behavior regarding cells within a small area. Thus, a common, time-dependent starting probability of a call was calculated for each area. A generic probability for all areas was neglected because of the infrastructural differences, amongst others concerning the road types and the means of travel, between them. As the information about the direction of the connections (incoming â outgoing) is not yet used by the TOL algorithm, the percentage of moving calls was set up to 50% for all simulation runs. As the results of analyses of the real-life GSM data show, the percentage of incoming calls for moving calls is about 45%. In addition to the probability for a cellular phone to switch to connected mode, also the duration of a call had to be modeled. The available real-life data contains the information about the duration of each call, so a distribution function could be generated. For the right part of the distribution, starting at duration of 60 seconds, the curve was assumed to be a log-normal distribution. The parameters were obtained by fitting the data to the log-normal distribution function concerning the respective range of values. Consequently, the computation of the call durations was modeled using the following equation:

(2)

where is the duration of a call in ms, and are uniformly distributed random number in range (0, 1] and is a uniformly distributed random number in range [0, 1]. A comparison between the measured and the modeled call duration frequencies is shown in figure 3. Calibrated with the probability curves, validation simulations using normal day traffic were performed. Figure 4 shows a comparison between the amounts of moving calls for the days real data existed, their mean value, and the simulation results. It is evident that the model is accurate concerning the moving calls, though it fails within cells in which vehicles begin their route, shown in the diagram at the bottom right of figure 4.

The current Approach

As said, the original code used within TrafficOnline was removed from SUMO. There were several reasons for this

- The original code used hard-coded values for telephony probability
- No further usage was avised after the project's end
- The code was bloated by the telephony "simulation"

The current approach is to use TraCI for simulating GSM-based surveillance. We set up the scenarios as done in TrafficOnline, but determining if a vehicle is crossing a cell or a LA boundary is done in an external script which is also responsible for simulating the telephony behaviour.

Appendices

Glossary

A

application
: An application is a computer program, herein, an "application" mostly means one of the main, compiled programs that are included in the SUMO package.

additional-file
: A file that can be included in a sumo configuration. This is needed to load pois, shapes, bus stops and variable speed signs. Call `sumo --help` for additional information.

C

connection(s)-file
: An XML-file describing connections between edges or lanes, see connection descriptions for importing networks defined as XML

D

DELPHI
: Deutschlandweite Echtzeit Verkehrs-Lage und Prognose im Ereignisfall

destination
: The position of the end of a vehicle journey. Mostly a district processed by OD2TRIPs within which a "sink" edge lies

detector
: An artificial device for object (mainly vehicle) states recognition and/or logging

DFD
: Deutsches Fernerkundungsdatenzentrum des DLR

district	A collection of edges describing a part of a road network's area
DLR	Deutsches Zentrum fÃ¼r Luft- und Raumfahrt
DUA	Dynamic User Assignment
DUE	Dynamic User Equilibrium

E

edge	A single-directed street connection between two points (junctions/nodes). An edge contains at least one lane
edge(s)-file	A XML-file describing edges of a road network, see edge descriptions for importing networks defined as XML

F

FCD	Floating Car Data

I

induction loop	A detector placed on a certain position on a lane which recognizes vehicles passing it and logs their attributes (speed, size, etc.)

J

junction	The place an edge begins or ends at (same as node)
junction logic	The part of a junction which determines the vehicle behaviour at the according junction by using certain right-of-way - rules

L

link	A connection between two lanes within a junction. Within microsim, each lane has links (connections) to the following lanes. A link contains the information whether the vehicle has to decelerate in front of the junction.

M

macroscopic
: In a macroscopic traffic (flow) simulation, the atomic instances are roads; the flow is simulated directly (see microscopic)

mesoscopic
: In a mesoscopic traffic (flow) simulation, the streets are partitioned into sections which multiple vehicles may enter and leave at each simulation step

microscopic
: In a microscopic traffic (flow) simulation, the atomic instances are vehicles; the flow is simulated by simulating the vehicles (see macroscopic)

multimodal
: A multimodal traffic simulation is capable of processing different types of traffic (busses, trains, vehicles etc.)

N

(street) network
: In our terms, a network is the combination of junctions (nodes) and edges (streets)

node
: A node (junction) is a single point were at least one edge (road) starts or ends

node(s)-file
: A XML-file describing nodes of a road network, see node descriptions for importing networks defined as XML

O

O/D-matrix (od-matrix)
: Origin/Destination-matrix; describes how many vehicles are moving from each origin to each destination within a certain time period

option
: A parameter for the application which determines what or how an application shall do

origin
: The position of the begin of a vehicle journey. Mostly a district processed by OD2TRIPs within which a "source" edge lies

P

poi
: short for "point of interest"; A position of an object that may be interesting

R

route
: A route is a complete description of a vehicle's path over the network; it contains the information

when the vehicle departs (the route starts) and over which edges the vehicle shall drive

S

script
: A small computer program which normally doees not have to be compiled. The SUMO package includes several scripts, most written in Python

source
: A source means a place at which vehicles are inserted into the street network

source-tar-ball
: A file containing the sources (program code) for a project. The source code must be compiled in order to get the runnable application. It is called "tar-ball" because of using the application "tar" to join all the source files into a single file.

submicroscopic
: In a submicroscopic traffic (flow) simulation, the atomic instances are parts of the vehicle or the driver (gearing or behavior is modelled)

T

TLS
: traffic lights system

trip
: A reduced information about a vehicle's movement; contains only the departure time, the begin, and the end edge. Must be transformed into a route using a router for being used within the simulation

trip-table (trip list)
: A file containing several trips

TS
: Institute of Transportation Systems at the German Aerospace Center (DLR)

TS-BS
: TS Standort Braunschweig

TS-BA
: TS Standort Berlin-Adlershof

type(s)-file
: A XML-file describing types of streets, see type descriptions for importing networks defined as XML

X

XML
: "eXtensible Markup Language is a universal format for structured documents and data" as the w3c says. Further information may be found at http://www.w3c.org/XML/. Most of the data read and written by SUMO is stored as XML

FAQ

General

What is SUMO?

SUMO is a traffic simulation package. It is meant to be used to simulate networks of a city's size, but you can of course use it for smaller networks and larger, too, if your computer power is large enough.

What does "SUMO" mean?

"SUMO" is an acronym for "Simulation of Urban MObility".

What kind of a traffic simulation is SUMO?

SUMO is a microscopic, space-continuous road traffic simulation. In the next time it will be extended to be multi-modal.

Features

Does SUMO support traffic within the junctions?

Yes, SUMO supports inner-junction traffic since version 0.9.5.

Is it possible to connect SUMO to an external application (f.e. ns-2)?

There are several approaches to do this, see (http://sumo.sourceforge.net/wiki/index.php/Main_Page#Related_to_C2C)

Can SUMO simulate lefthand traffic?

No.

Is it planned to simulate lefthand traffic one day?

Would be definitely a nice feature and we would support anyone who wants to implement it. As we are not involved in projects were lefthand traffic is wished, we are not capable to do this work by ourselves.

Can SUMO generate movement traces?

Most of the C2C-extensions do this, see (http://sumo.sourceforge.net/wiki/index.php/Main_Page#Related_to_C2C)

Building / Installation

How do I access the Subversion repository?

There are plenty of subversion clients for all platforms. If you use the command line client, you can checkout sumo using the following command:

```
svn co https://sumo.svn.sourceforge.net/svnroot/sumo/trunk/sumo
```

For later updates go inside the sumo directory, which has been created, and simply type svn up.

Is there further documentation on Subversion?

There are the Sourceforge documentation and the Subversion book.

How to check out revision 5499 (or any other outdated sumo)?

You can use the SVN option "**-r <REVISION_NUMBER>**" together with the checkout on the command line. You have to consult your client's documentation if you use a graphical interface. Please be aware of the fact that we can only give very limited support for older versions.

Which platforms are supported?

We compile regularly under Windows XP using Visual Studio 2005 (aka MSVC 8) and have daily builds on Linux (openSUSE 10.3 and 11.1 with gcc4). We would be happy to hear about successful builds on other platforms. We already heard about successful builds (without GUI) on MacOS X, Solaris and Cygwin.

Troubleshooting

See LinuxBuild or WindowsBuild.

Basic Usage

What measures are used/represented?

All time values are given in seconds. All length values are given in meters. This means that speed should be given in m/s, etc.
Currently, each simulation time step is one second long (to be exact: in each simulation step one second real time is simulated).

What does the following error mean?

Warning: No types defined, using defaults... Error: An exception occurred! Type:RuntimeException, Message:The primary document entity could not be opened. Id=<PATH> Error: (At line/column 1/0). Error: Quitting (conversion failed).
Answer: Simply that the file you try to use (<PATH>) does not exist. This is xerces' way to say "file not found".

NETCONVERT

Segmentation Fault while building a Road Network

It was reported that NETCONVERT breaks with a segmentation fault under Ubuntu Linux in step 12 "Computing node shapes". It seems that the reason is due to a broken (?) version of the GDAL library. You probably have to install your own one or try to build NETCONVERT with no GDAL support.

Can I import the free network of OsnabrÃ¼ck "Frida"?

Yes and no. You import it using NETCONVERT, a description is available at http://sumo.sourceforge.net/docs/gen/sumo_moreon_arcview.shtml#more_arcview-frida. Still, this may be a good gis-network but lacks some needed information in order to be usable for simulations (see discussion at the link above).

Are there any other free networks available I can use?

Actually, we have not yet spent so much time for finding open source networks. Take a look at http://sumo.sourceforge.net/docs/gen/sumo_moreon_arcview.shtml in order to find some further information. That's all we know by now.

The application hangs after a while (few memory consumption, most of the system time) (Windows)

You are propably running a program compiled in the debug-mode. This yields in at least the triple of normal memory usage and your system may not be able to solve. Try to use the normal version, build in Release-mode (or buy more RAM:-)).

NETCONVERT aborts at step 12 (Computing node shapes)

Please check whether the network you try to import is encoded in geocoordinates. If so, try to convert it with the option --proj.utm (for UTM projection which is the most widely used) or --proj.dhdn (for "old" german data) or if you are familiar with the proj utility with a properly defined projection using --proj <PROJ_DEF>.

Simulation

How to simulate an accident

Currently, there are two possibilities to simulating something like an 'accident':
1. Let a vehicle halt on the lane for some time (see http://sumo.sourceforge.net/docs/gen/user_chp06.shtml#user_chp06-management-public). This works quite nice for simulating accidents.
2. Put a variable speed sign of the lane where the accident is meant to be and let it reduce the speed (see http://sumo.sourceforge.net/docs/gen/user_chp06.shtml#user_chp06-management-vss). This method will reduce the throughput on the lane, but further dynamics will rather not fit to what one would expect from an accident situation.
1+2. You may of course combine both approaches. Within TrafficOnline, we simulated traffic incidents by letting vehicles stop on one lane and reducing the speed on the other lanes.

I have changed my network and now SUMO does not load it.

Actually, SUMO-networks are not meant to be edited by hand. You have to describe everything within your input properly and let NETVCONVERT build your network. Editing networks by hand is very complicated and error-prone.

No vehicle appears

We encountered this problem under the following circumstances: As we imported a network where the maximum speed allowed on was given in km/h instead of the supposed m/s, no vehicle could be emitted due to a too large space needed for oncoming traffic. Check whether your network description also defines speed in km/h. If so, use the SUMO-NETCONVERT - switch "speed-in-kmh" to recompute the speed limits. If this does not help, contact us.

Vehicle jumps backward

When I try to evaluate my vehicle dump, sometimes vehicle seem to jump backward (their position is smaller than in the previous step)

To avoid jams due to an inappropriate lane changing behaviour, vehicles may swap their lanes. In this case each vehicle obtains the position and speed of the other vehicle. As the position of the vehicles may differ a bit, one of the vehicles may seem to jump backwards.

This should not happen from Version0.8 on.

How do I change the duration of cycles and phases?

If you have a network that contains traffic lights, you can open it with a text editor. Then search for "tl-logic". You will find the definition(s) of your traffic light(s). Now, you can see a set of "phase"-definitions within each of the traffic lights definitions. You can change the time for each

phase directly in here.
See also the documentation on traffic light (Traffic Lights) and (SUMO - More on... Traffic Lights).

I can not see a vehicle moving in my simulation

There may be two reasons why you do not see the cars.
1. If your simulation area is too big, cars will not be displayed unless you zoom into the net. The reason is that cars are displayed if their visible size is larger than 1 pixel in order to improve the visualization speed by avoiding drawing all the streets and cars even if the cars would not be visible. You may also change the settings by clicking to the color wheel in guisim, selecting the "vehicles" panel and changing the value of "minimum size to show" to zero.
2. The simulation is too fast so that all vehicles disappear before being seen. To avoid this, you may increase the "Delay"-value what makes the simulation wait between simulation steps.

Different departure times with different time step size

Vehicles are entering the network earlier with decreasing time step lengths.
The reason for this behaviour is that a vehicle is emitted at the end of a time step. For one second time steps, this means that a vehicle which has the depart time of 0 will be inserted into the network at the end of the time step between 0 and 1, this means almost on second 1 (0.99...). If time steps of 0.1 seconds are used, the same vehicle is inserted into the network at the end of the time step between 0 and 0.1, this means almost on 0.1 (0.099...).

Visualisation

GUISIM breaks

Sometimes guisim terminates with no reason. In most cases, an update of the opengl-driver solves this problem.

Display flickers in the area of the mouse pointer (Windows)

Newer Windows-Versions seem to cache the area under the mouse to apply the mouse shadow afterwards. To avoid this, go to your Systemmenu, then Mouse->Pointers and disable the mouse shadows. That's the only solution so far. (Origin: Till Oliver Knoll, via QT Interest List)

Loading of a jpeg/png fails

Currently (version 0.9.7), GUISIM is not capable to load jpegs and pngs. We will try to add a support for these formats in the next time.

GUISIM finishs with: FATAL: exception not rethrown (Linux)

This is a known bug in the pthreads library. The last link also shows how to patch fox in order to avoid this.

GUISIM windows and buttons appear but no net/cars are visible

You have most definitely an opengl problem. Even recent (as of 01/2009) software implementations (for instance mesa together with the radeon driver on X) are buggy. Some workarounds are mentioned at the end of this thread.

XML Schema Definitions

XML Schema Definitions (xsd) define the structure of XML files. As all files used by SUMO natively are written in XML, they can be verified against according XML schemas. The work on the schema definitions is currently ongoing. Schema definitions exist for the following file types:

- nodes files (used by NETCONVERT): http://sumo.sourceforge.net/xsd/nodes_file.xsd
- edges files (used by NETCONVERT): http://sumo.sourceforge.net/xsd/edges_file.xsd
- types files (used by NETCONVERT): http://sumo.sourceforge.net/xsd/types_file.xsd
- connections files (used by NETCONVERT): http://sumo.sourceforge.net/xsd/connections_file.xsd

These file define the format used by the current (SVN) version.

SUMO

From 30.000 feet

SUMO is the simulation itself; it is a microscopic, space-continuous, and time-discrete traffic flow simulation.

Purpose: Simulates a defined scenario
System: portable (Linux/Windows is tested); runs on command line
Input (mandatory):
A) a road network (as generated via NETCONVERT or NETGEN, see also building the networks),
B) a set of routes (as generated by DUAROUTER, JTRROUTER, or DFROUTER, see also building the demand)
Input (optional): Additional definitions of traffic lights, variable speed signs, output detectors etc.
Output: SUMO allows to generate a wide set of outputs; visualization is done using GUISIM
Programming Language: c++

Subtopics

- Using additional Polygons and POIs within the Simulation

Category: ApplicationDescription

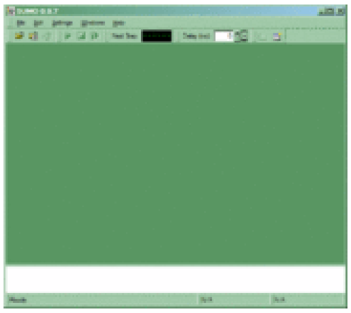
GUISIM

From 30.000 feet

GUISIM is basically the same application as SUMO, just extended by a graphical user interface.

> **Purpose:** Simulates defined scenarios
> **System:** portable (Linux/Windows is tested); opens a window
> **Input (mandatory):** A SUMO-configuration file (see SUMO)
> **Output:** GUISIM generates the same output as SUMO
> **Programming Language:** c++

Brief Usage Description



Initial appearence of GUISIM; may differ from what you see
Being a window-based application, GUISIM is started by a double click with the left mouse button on Windows, on Linux probably with a single click. After this, an empty window should show up, similar to the one shown in the image.

Using either the "File->Open Simulation..." menu entry or by using the "open"-icon (📂), you should be able to load an existing SUMO configuration file, if it has the proper extension *".sumo.cfg"*. If the SUMO configuration file is errornous, the errors are reported, otherwise your network referenced within the configuration file should be shown. Now you can start to simulate by pressing the "play" button (▶). The simulation works as if being started on the command line. The simulation can be halted using the "stop" button (■) and continued by pressing the "play" button again. When stopped, also single steps may be performed by pressing the "single step" button (⏯).

If the simulation is running, the current simulation step is shown in the "digital digits" field, right to "next step" (Next Step: 000 150).

Besides loading simulation configurations, it is also possible to load networks by using either the "File->Open Network..." menu entry or by using the "open network"-icon (). Please note, that normally **GUISIM** assumes networks have the extension *".net.xml"*, but also accepts other extensions.

Both, the loaded simulation or the loaded network may be reloaded using the "reload" button () or the menu entry "File->Reload".

If a network or a simulation are loaded, the navigation through the network is possible with the mouse only. One can drag the network with the left mouse button pressed into all directions and zoom either by using the mouse wheel or by pressing the right mouse button and moving the mouse up and down. For fine grained zooming (half zoom speed) press the "Control" key while using the mouse wheel, for double speed use "Shift".

Subtopics

- Changing the appearance/visualisation of the simulation
- Using additional Polygons and POIs within the Simulation
- Using decals within GUISIM
- Selecting net elements in GUISIM

Category: ApplicationDescription

NETCONVERT

From 30.000 feet

NETCONVERT imports digital road networks from different sources and generates road networks that can be used by other tools from the package.

Purpose: Road networks import and conversion
System: portable (Linux/Windows is tested); runs on command line
Input (mandatory): Definition of a road network
Output: A generated SUMO-road network; optionally also other outputs
Programming Language: c++

Usage Description

NETCONVERT is a command line application. It assumes at least one parameter - the combination of the name of the file type to import as parameter name and the name of the file to import as parameter value. So, for importing a network from OpenStreetMap one could simply write:

```
netconvert --osm my_osm_net.xml
```

and for importing a VISUM-network:

```
netconvert --visum my_visum_net.net
```

In both cases, as no output name is given, the SUMO network generated from the imported data is written into the file "net.net.xml". To write the network into a different file, use the option **-o *<OUTPUT_FILE>***. If you want to save the imported VISUM-network into a file named "my_sumo_net.net.xml", write:

```
netconvert --visum my_visum_net.net -o my_sumo_net.net.xml
```

Many further parameter steer how the network is imported and how the resulting SUMO-network is generated.

Supported Formats

NETCONVERT is able to import road networks from the following formats:

- "SUMO native" XML descriptions (*.edg.xml, *.nod.xml, *.con.xml)
- Shapefiles (.shp, .shx, .dbf), e.g. ArcView and newer Tiger networks
- OpenStreetMap (*.osm.xml), including shapes
- Robocup Rescue League, including shapes
- VISUM, including shapes and demands
- Vissim, including demands
- a DLR internal variant of Navteq's GDF (Elmar format)
- SUMO-networks (*.net.xml)

Options

Import Options

Option	Mandatory y/n	Description
--xml-node-files <FILE>[,<FILE>]* --xml-nodes <FILE>[,<FILE>]* -n <FILE>[,<FILE>]*		Reads node definitions from the given file(s)
--xml-edge-files <FILE>[,<FILE>]* --xml-edges <FILE>[,<FILE>]* -e <FILE>[,<FILE>]*		Reads edge definitions from the given file(s)
--xml-connection-files <FILE>[,<FILE>]* --xml-connections <FILE>[,<FILE>]* -x <FILE>[,<FILE>]*		Reads connection definitions from the given file(s)
--xml-type-files <FILE>[,<FILE>]* --types <FILE>[,<FILE>]* -t <FILE>[,<FILE>]*		Reads edge type definitions from the given file(s)

Projection

Most data of real world networks has locations and shapes of streets and junctions specified with geocoordinates. This is especially true for OSM, Tiger networks (and most of the other shapefiles) and Navteq

GDF and derivatives. Since SUMO works with cartesian coordinates only, you need to perform a projection of the geo coordinates into the plane. Unfortunately there are plenty of projections available but fortunately there exists the Proj library (which is used by NETCONVERT) to handle almost all of them. There are several options to use geo projection (and proj) together with NETCONVERT:

Option	Mandatory y/n	Description
--proj.simple		Use a simple projection method
--proj.shift *<INT>*		Number of places to shift the decimal point to the right in geo coordinates
--proj.utm		Determine the UTM zone (for a universal transversal mercator projection based on the WGS84 ellipsoid)
--proj.dhdn		Determine the DHDN zone (for a transversal mercator projection based on the bessel ellipsoid)
--proj *<STRING>*		Uses string as proj.4 definition for projection
--proj.inverse		Inverses projection

In case you (or your network supplier) are already familiar with the use of proj you can pass the correct parameters directly to NETCONVERT using --proj. Sometimes also the EPSG number of a projection is known, which together with this list (coming also with most proj installations) helps to find the correct proj parameters. Most of the networks have however WGS84 data, which is usually converted using UTM. In order to support this often used projection NETCONVERT provides another option --proj.utm which determines the correct UTM zone to use from the longitude of the first point in the network. --proj.dhdn does essentially the same for data referring to the "Deutsches HauptDreiecksNetz" which used to be the basis for most of the administrative data in germany.

If you want to "undo" a projection for instance if you need to modify an existing sumo network, you can do an inverse projection by giving --proj.inverse. If you don't have proj available but desperately need a projection of some kind, you can try --proj.simple. This should be a last resort only, if you have proj, you should always prefer --proj.utm.

Net Translation Options

Normally, both **NETCONVERT** and NETGEN translate the read network so that the left- and down-most node are at coordinate (0,0). The following options allow to disable this and to apply different offsets for both the x- and the y-axis. If there are explicit offsets given, the normalization is disabled automatically (thus there is no need to give --disable-normalize-node-positions if there is at least one of the offsets given).

Option	Mandatory y/n	Description
--disable-normalize-node-positions		Disables network translation to (0,0)
--x-offset-to-apply *<FLOAT>*		Applies an offset to the x-axis (in m)
--y-offset-to-apply *<FLOAT>*		Applies an offset to the y-axis (in m)

Category: ApplicationDescription

NETGEN

From 30.000 feet

NETGEN generates abstract road networks that may be used by other SUMO-applications.

Purpose: Abstract road network generation
System: portable (Linux/Windows is tested); runs on command line
Input (mandatory): Command line parameter
Output: A generated SUMO-road network; optionally also other outputs
Programming Language: c++

Options

Net Translation Options

Normally, both NETCONVERT and **NETGEN** translate the read network so that the left- and down-most node are at coordinate (0,0). The following options allow to disable this and to apply different offsets for both the x- and the y-axis. If there are explicit offsets given, the normalization is disabled automatically (thus there is no need to give --disable-normalize-node-positions if there is at least one of the offsets given).

Option	Mandatory y/n	Description
--disable-normalize-node-positions		Disables network translation to (0,0)
--x-offset-to-apply *<FLOAT>*		Applies an offset to the x-axis (in m)
--y-offset-to-apply *<FLOAT>*		Applies an offset to the y-axis (in m)

Category: ApplicationDescription

OD2TRIPS

From 30.000 feet

OD2TRIPS imports O/D-matrices and splits them into single vehicle trips.

Purpose: Conversion of O/D-matrices to single vehicle trips
System: portable (Linux/Windows is tested); runs on command line
Input (mandatory): O/D-Matrix, a set of districts
Output: A list of vehicle trip definitions
Programming Language: c++

Category: ApplicationDescription

DUAROUTER

From 30.000 feet

DUAROUTER imports different demand definitions, computes vehicle routes that may be used by SUMO using shortest path computation; performs a DUA.

> **Purpose:** Building vehicle routes from demand definitions
> **System:** portable (Linux/Windows is tested); runs on command line
> **Input (mandatory):**
> > A) a road network (as generated via NETCONVERT or NETGEN, see also building the networks),
> > B) a demand definition
>
> **Output:** Vehicle routes
> **Programming Language:** c++

Category: ApplicationDescription

JTRROUTER

From 30.000 feet

JTRROUTER computes routes that may be used by SUMO using different amount definitions and junction turning percentages.

> **Purpose:** Building vehicle routes from demand definitions using junction turning percentages
> **System:** portable (Linux/Windows is tested); runs on command line
> **Input (mandatory):**
> > A) a road network (as generated via NETCONVERT or NETGEN, see also building the networks),
> > B) a demand definition
>
> **Input (mandatory):**
> > Junction turning definitions
>
> **Output:** Vehicle routes
> **Programming Language:** c++

Category: ApplicationDescription

DFROUTER

From 30.000 feet

DFROUTER uses induction loop values to compute vehicle routes that may be used by SUMO.

Purpose: Building vehicle routes from induction loop counts
System: portable (Linux/Windows is tested); runs on command line
Input (mandatory):
A) a road network (as generated via NETCONVERT or NETGEN, see also building the networks),
B) induction loop definitions
C) induction loop measures
Output: Vehicle routes; different other
Programming Language: c++

Category: ApplicationDescription

POLYCONVERT

From 30.000 feet

POLYCONVERT imports geometrical shapes (polygons or points of interest) from different sources, converts them to a representation that may be visualized using GUISIM.

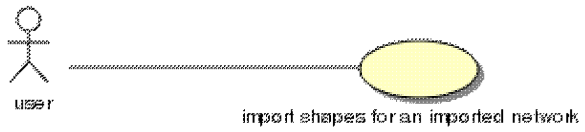
Purpose: Polygon and POI conversion
System: portable (Linux/Windows is tested); runs on command line
Input (mandatory): polygons or pois stored in different formats
Output: SUMO-shape file
Programming Language: C++

Features

- Imports OSM, VISUM, Elmar, XML, ArcView shape files
- per-type import settings
- projections using a given proj.4-definition or via a matching network
- Writes simulation shape files usable within GUISIM and SUMO

Usage Description

Use Cases



PC01: import shapes for an imported network

Use Case ID	PC01
Application	**POLYCONVERT**
Use Case Name	import shapes for an imported network
Use Case Description	For a network imported from a non-SUMO-native format using NETCONVERT, the shapes (POIs/Polygons) stored in the same format and same projection shall be imported.
Primary Actor	The user
Precondition	The road network must have been imported using NETCONVERT
Trigger	N/A
Basic Flow	The user starts **POLYCONVERT** on the command line, giving it the reference to the converted SUMO road network using the option --net-file *<FILE>*. Additionally, the input file to parse is given using the proper option. In order to apply the projection defined within the network, the option --use-projection must be given. The output file name is defined by the option --output *<FILE>*.
Alternate Flows	N/A
Example Call	polyconvert --net-file converted_network.net.xml --visum-files orig.net --use-projection --output mypolys.poi.xml

Options

Import Options

POLYCONVERT is able to import shapes from different file types. Normally, for importing data of a certain type, the type name is used as option name and the value indicates the position of the file. So

```
polyconvert --visum mynet.net -o converted.poi.xml
```

imports from a VISUM-net file.

Option	Description
--elmar-poly-files <FILE>[,<FILE>]*	**POLYCONVERT** shall read polygons stored in an Elmar polygon file(s) <FILE>
--elmar-poi-files <FILE>[,<FILE>]*	**POLYCONVERT** shall read POIs stored in an Elmar POI file <FILE>
--xml <FILE>[,<FILE>]*	Forces **POLYCONVERT** to read POIs and/or polygons stored in an XML file
--osm-files <FILE>[,<FILE>]* --osm <FILE>[,<FILE>]*	**POLYCONVERT** shall read POIs and/or polygons stored in an OSM file <FILE>
--osm.keep-full-type	When importing, the shape type will be constructed from the value/key pair; valid only if importing OSM shapes
--visum-files <FILE>[,<FILE>]* --visum <FILE>[,<FILE>]*	**POLYCONVERT** shall read POIs and/or polygons stored in a VISUM file <FILE>
--shape-file <FILE> --shape <FILE>	**POLYCONVERT** shall read POIs and/or polygons stored in a shape files <FILE>
--arcview.guess-projection	Tries to guess the projection of the read shape file; valid only if importing shapes from shape files
--shape-file.id-name <ID>	Tells **POLYCONVERT** from which db-table column the id of the shape shall be read; valid only if importing shapes from shape files

Output Options

All imported shapes that have not been discarded are written into a file which has to be defined using --output <FILE>.

Option	Description
--output <FILE> -o <FILE>	Defines the file into which the converted shapes shall be written

Type-Dependent Import

POLYCONVERT is capable to apply different attributes to the imported shapes in dependence of their "type". Not all imported formats have a type information. When using shape files, for example, all instances of an artifact type are normally stored in a distinct shape file.

Option	Description
--typemap <FILE>	Defines from which type definitions shall be read

Projection Options

One of the major uses of **POLYCONVERT** is to apply a projection on the read shapes. Normally, one wants the shapes to be aligned in accordance to a previously imported road network. In this case, the network should be given using --net-file *<FILE>*. But it is also possible to use a different projection. In any case, if the read coordinates shall be changed, --use-projection must be given.

Option	Description
--use-projection	Whether the read data shall be projected
--net-file *<FILE>* --net *<FILE>* -n *<FILE>*	Defines which network file shall be used as the projection source (--use-projection must be set)
--proj *<PROJ_DEFINITION>*	Uses the given proj.4 definition string to initialise the projection
--proj.simple	Uses a simple projection procedure
--proj.inverse	Inverses he projection

Pruning Options

Sometimes, shapes cover a much larger area than the network. In order to reduce the amount of data, one can force **POLYCONVERT** to prune the imported data on the network's or a given boundary. Read shapes which are completely outside this boundary are discarded in these cases.

Option	Description
--prune.on-net	Forces **POLYCONVERT** to discard shapes which are completely outside the network's area
--prune.on-net.offsets *<2D-BOUNDING_BOX>*	Defines the offset to apply to the network boundary
--prune.boundary *<2D-BOUNDING_BOX>*	Forces **POLYCONVERT** to discard shapes which are completely outside the given boundary
--prune.ignore *<ID>[,<ID>]**	Shapes with the given ids will not be discarded, even if being outside the pruning / network boundary
--remove *<ID>[,<ID>]**	Shapes with the given ids will be discarded

Options for Setting Defaults

When importing shapes for which no type-dependent attributes have been given, the following default values are used which can be changed on the command line.

Option	Description
--color *<COLOR>*	Per default, the imported shapes will have this color
--prefix *<ID>*	Per default, this prefix will be prepended to the shape ids
--type *<ID>*	Per default, the imported shapes will get the given type
--layer *<INT>*	Per default, the imported shapes will be located in this layer

Reporting Options

These option change the reporting behaviour of **POLYCONVERT**.

Option	Description
--verbose	**POLYCONVERT** reports what it is doing
--suppress-warnings	**POLYCONVERT** will not write warnings
--print-options	**POLYCONVERT** shows the options set
--help -?	**POLYCONVERT** shows the help screen
--log-file *<FILE>*	**POLYCONVERT** writes on the console and into this file

See Also

- Using additional polygons and POIs within the Simulation
- Using configuration files

Known Issues

Category: ApplicationDescription

AdditionalTools

Overview

Tools can be found in the SUMO-distribution under <SUMO_DIST>/tools. Most of them are tiny - they were written for a certain purpose and worked well under certain input, but may be not verified for other cases.

The tools are divided into the following topics:

- *assign* - traffic assignment tools
- *build* - tools used for code styling and by the building subsystems
- *detector* - some tools for dealing with real life induction loop data
- *district* -
- *examples* -
- *import* - allow to import data from formats which can not be processed "natively"
  - *osm* - some helpers for accessing/using OpenStreetMap data
  - *visum* - some helpers for using VISUM data
- *net* - tools for working with networks (mainly SUMO-networks)

- *projects* - additional tools used by our projects
    - *TaxiFCDKrieg_tools* - validation Simulation<->FCD (Diploma Thesis by Sascha Krieg, running)
- *route* - tools for working with routes
- *Shapes Tools* - tools for working with shapes (PoIs and polygons)
- *tls* -
- *traceExporter* -
- *trip* - Trip generation and modification without OD2TRIPS
- *visualization* - graphical evaluation of SUMO-outputs

Translations

Navigation

- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help

.

- This page was last modified on 14 September 2010, at 09:44.
- This page has been accessed 12,921 times.