Vissim Kernel for Linux

A full manual for PTV Vissim is distributed with the Windows package and is accessible here: https://cgi.ptvgroup.com/vision-help/VISSIM_2022_ENG. This document only describes the usage of the PTV Vissim Kernel for Linux.

Installation requirements

PTV Vissim Kernel is supported only on the 64-Bit Ubuntu long-term support releases 18.04 and 20.04. This manual assumes installation on Ubuntu 18.04.

However, the installation package for PTV Vissim Kernel is almost self-contained and should run on most reasonably modern Linux distributions. The only external references are:

- libc6 >= 2.27 (various shared libraries: pthread, libm, …)
- GCC support library >= 8.4 (libgcc_s.so)

For the actual installation, some other standard tools, for example tar, are required. In addition, to access the DrivingSimulator interface, a C/C++ compiler tool chain is required, as you need to write a program against the shared object library.

Non-Ascii filenames are only supported if a Unicode locale is active. Most distributions set a Unicode locale, but e.g. Docker containers may not. As an example, you can set the US Unicode locale with the command:

```
export LC_ALL=en_US.utf8
```

Activation of Licenses

There are two licensing options: software license or cloud license (CmCloud).

- Software license works with an activation code. You need to install your license as a CodeMeter License Server on a physical machine.
- Cloud license (CmCloud) works with a certificate (this is a *.wbc file). You do not need to set up a CodeMeter License Server.

Prerequisites:

- Your PTV Vissim Kernel license activation code or certificate file (depends on the license option)
- Network access from each (virtual) machine which runs PTV Vissim Kernel to the CodeMeter License Server (software license) or to the cloud (cloud license [CmCloud])

Setting up a CodeMeter License Server (only for software license with activation code)

Note: Running a CodeMeter License Server on a Virtual Machine is not supported with the software license. The CodeMeter License Server is usually not the machine on which PTV Vissim Kernel runs.

Download the "CodeMeter User Runtime" suitable for the CodeMeter License Server's operating system from https://www.wibu.com/support/user/user-software.html and install it. If the CodeMeter License Server runs on Ubuntu 18 you need the "CodeMeter User Runtime for Linux".

Install the package with the command:

```
sudo apt install <path_to_installation_package>
```

Enable the CodeMeter network server mode:

- You can use the CodeMeter WebAdmin interface (https://www.wibu.com/magazine/keynote-articles/article/detail/the-new-look-of-webadmin.html) It should be available at http://localhost:22352/ . Under `Configuration/Server/Server Access` enable `Network Server`. You can probably ignore the Network Port.
- Alternatively, on Linux you can modify the setting `IsNetworkServer=1` in the CodeMeter configuration file `/etc/wibu/CodeMeter/Server.ini` and then restart the CodeMeter service.

If needed, more details are provided in the CodeMeter Administrator Manual available at https://www.wibu.com/support/manuals-guides.html

Start a web browser on the CodeMeter License Server machine and activate your licenses with your activation code on the license activation web page. You should have received both the activation code and the address of the web page when buying your license. The activated licenses are bound to the CodeMeter License Server machine. If you want to activate the licenses on another machine you have to deactivate them first. You can do this also with the activation code on the license activation web page.

PTV Vissim Kernel machine

Each machine running PTV Vissim Kernel needs network access to a CodeMeter License Server.

On each PTV Vissim Kernel machine:

- Download the 64-Bit "CodeMeter User Runtime for Linux" from https://www.wibu.com/support/user/downloads-user-software.html
- Install it with the command: `sudo apt install <path_to_installation_package>`
- If you use a software license and host your own CodeMeter License Server, on which you activated your license, add it to the search list:
  `cmu --add-server <CodeMeter License Server IP Address>`
- If you use a cloud license (CmCloud) activate your certificate:
  `cmu --import --file <certificate>.wbc`

PTV Vissim Kernel will automatically use a license from the CodeMeter License Server or from the cloud.

In case of problems with your software license, you can verify that your CodeMeter License Server and its hosted licenses are found with `cmu --list-server --list-content`.

Installation

Download

Download the PTV Vissim Kernel installation package. It consists of a compressed archive.

Extract

We recommend to extract the archive to the folder `/opt`:

```
cd /opt
sudo tar xf <path_to_installation_package>
```

This unpacks the whole distribution into a sub-folder of `/opt`.

Verify

You should now be able to start the command line client under `/opt/<vissim_subfolder>/bin/vissim-kernel`.

Vissim calculates the location of all installed files relative to the location of the executable or the driving simulator shared object, so no further setup is required.

Usage of PTV Vissim Kernel

The Vissim command-line client is generally started with the command line

```
vissim-kernel <options> <inpx-filename>
```

If the path to the `bin/` directory has not been set, you must use the absolute path name to the installed executable.

`<inpx-filename>` is the absolute or relative path to the Vissim input file to simulate.

The following options are supported:

- `-h / --help`

gives a command summary of all options

- `-c <filename> / --config <filename>`

Uses the given filename for configuration parameters instead of the configuration file in the user's home directory (see below for details)

- `-s <no> / --simRunNo <no>`

Sets the number of the simulation run. If not set, Vissim automatically selects this number depending on already existing simulation runs.

- `-r <seed> / --randSeed <seed>`

Uses the supplied integer as the seed for the random number generator. If this parameter is not set, the seed is taken from the network file.

- `--recordANI`

Records animations during the simulation which can be replayed with the Windows GUI. Network objects for the animation recordings must be configured in the input file.

- `-t <numThreads> / --threads <numThread>`

Uses up to the specified number of threads for the simulation.

- `-v / --verbose`

Increases verbosity of the output.

- `-o / --output_folder`

Overrides the simulation result output folder defined in the inpx. Causes all

simulation results, including evaluation result databases to be written directly to the provided directory.

- `--version`

Prints version information and exits.

The examples directory contains a network file which is configured to write a network performance evaluation into the database in the `*.results` folder next to the input file. When simulated with `--recordANI`, an animation of the vehicles in the network is also written to `<filename>.ani` in the folder of the input file. Both outputs can be copied to a Windows computer to be analyzed with the Vissim GUI.

To configure some settings, PTV Vissim Kernel uses configuration files. Settings are first read from the installation-wide configuration file under `etc/vissim.cfg` in the installation directory. These can be further overridden by user-specific configurations in a file `$HOME/.config/ptv-vision/vissim-kernel-<version>/vissim.cfg` (e.g. with `<version>` = `2022`) or by the configuration file given with the `-c / --config` command-line parameter. The installation-wide configuration file comes with documentation on the meaning of the individual configuration options.

Note that PTV Vissim Kernel has some limitations as of now:

- Simulation results may differ between simulations run on Linux and Windows. This is a fundamental limitation due to the different runtime environments (libc).
- Several extension interfaces have not been implemented under Linux yet. In particular, signal controllers and the emission interface do not work. Driver model should work, but is not yet supported.
- Internal scripting is not supported.
- Scenario management support is not supported.
- Playing sound files when triggering detectors is not supported under Linux.

DrivingSimulator interface

In addition to running simulations from the command line, you can also use the DrivingSimulator (DS) interface. This offers a C interface that is identical to the interface supported by the full PTV Vissim package for Windows. The interface is accessible through the shared object `libDrivingSimulatorProxy.so`.

Example

An example program for the DrivingSimulator interface can be found in `examples/DrivingSimulatorInterface`. To run this example, install the GNU C++ compiler and perform the following steps:

- Make a copy of the `examples/DrivingSimulatorInterface` directory and cd into that copy.
- Run `env VISSIM_INSTALL_DIR=<path to vissim> bash build.sh` to build the example.
- Run `./DrivingSimulatorTextClient 10 10` to run the example.

Architecture

To use the DrivingSimulator interface, include the `DrivingSimulatorProxy.h` header located in the `include` folder of your Vissim installation and link against the `libDrivingSimulatorProxy.so` file in the `lib` folder.

After linking against `libDrivingSimulatorProxy.so`, you need to ensure that the dynamic linker finds the library on program start. This can be done by setting the environment variable `LD_LIBRARY_PATH` to the `lib/` directory inside the Vissim installation.

Alternatively, you can use the RUNPATH mechanism to insert the library search path into the build result. See the build.sh script for an example. Calling Vissim_ConnectToKernel() starts a PTV Vissim Kernel instance, and initiate communication via shared memory.

The DrivingSimulator automatically records an ANI file if this has been enabled in the network file, which is the case for the demo example.

Additional optional parameters allow to set the frequency of the simulator, a visibility radius, and the maximum number of data sets for vehicles, pedestrians, signal groups and detectors to be passed in one or both directions. In the Vissim network settings, the option "driving simulator" needs to be activated and a default vehicle type must be selected to be used for DS vehicles with unspecified type. After loading the network, Vissim establishes communication with the proxy library and waits for the starting location(s) of the DS vehicle(s) present at the start of the simulation run (in world coordinates, fitting with the Vissim network).

Vissim determines the most probable location for each DS vehicle in the Vissim network by selecting the link with the driving direction closest to the orientation of the DS vehicle from all links at the passed world coordinates.

After the DS has placed its vehicle(s) at their locations in the Vissim model, the first time step of the Vissim simulation is executed. After that time step, Vissim passes the world coordinates of all vehicles in the network (including the DS vehicle(s)) to the DS and waits for the new location of the DS vehicle(s). The simulator can now read the new locations of the Vissim vehicles, visualize them and then pass back a new set of locations for the DS vehicle(s). After this, Vissim immediately calculates the next time step, and so on. When the DS passes the new location of the simulator vehicle(s) to Vissim, the DS can also add new DS vehicles to the network and/or remove DS vehicles from the network and/or pass the control over former DS vehicles to Vissim and/or take control over former Vissim vehicles. Because of this, all vehicles (including the DS vehicles) need to be identified by their Vissim vehicle ID. When a new DS vehicles is to be created, the DS passes a CreateID to Vissim which is then returned (only in the next time step) with the vehicle data including the actual Vissim vehicle ID, so the simulator can identify multiple DS vehicles and can subsequently use the Vissim vehicle IDs to designate all DS vehicles to Vissim.

Before the DS passes the new location of the simulator vehicle(s) to Vissim, it can optionally set detector states. Before or after retrieving the new locations of the Vissim vehicles, the DS can optionally retrieve the signal states passed from Vissim and/or a list of all vehicles which have entered the Vissim network in this time step, a list of all vehicles which have left the Vissim network in this time step and a list of all vehicles which have changed their location in this time step.

Similar as for vehicles, data of Vissim (Viswalk) pedestrians and/or DS pedestrians can be exchanged if desired. Unlike vehicles, DS pedestrians can be created only at the start of a simulation run. Furthermore, Vissim passes only data of Vissim pedestrians to the DS.

This data exchange between Vissim and the DS must be executed at least once per Vissim simulation time step. If the simulator runs at a higher frequency (frame rate) than Vissim, the interface provides automatic interpolation of the positions of the Vissim vehicles and pedestrians between Vissim simulation time steps.

The timing of the co-simulation is controlled by the DS. It can run faster than real time (if this is possible for the DS) but not faster than a stand-alone Vissim simulation of the same network. (The Vissim simulation of a time step can take much less or much more than real time, depending on the used hardware and the size of the network. A big network might not be possible to be simulated with very short time steps in real time.) The DS can slow down the co-simulation by waiting between DLL calls as much as desired (e.g. to reach exact real time) without hurting the Vissim simulation at all.