VeloView Developer Guide

Software developer documentation for the VeloView application and libraries

Version: 3.5.0

Table of Contents

1 Build instructions

1.1 Superbuild overview

VeloView uses a CMake *superbuild* system. This is a cmake project that defines dependencies and third party projects as CMake targets, such that their tested versions are automatically downloaded, compiled and linked to. The superbuild will give you the option to use system installations of third party projects instead of compiling themas a superbuild step. Some dependencies, on certain platforms, must be compiled by the superbuild, and there is no option to use a system version.

1.2 Getting the source code

The VeloView software is hosted in a git repository on github.com Use the following command line to checkout the source code:

```
git clone git@github.com:Kitware/VeloView.git
```

1.3 Veloview dependencies

The VeloView application and libraries have several external library dependencies. As explained in the Superbuild overview, the dependencies will be downloaded and compiled automatically during the build step of the Superbuild. Exact project urls and versions can be found in Superbuild/versions.cmake. See Configure and build instructions.

1.3.1 pcap

The required pcap version is 1.4. On Mac/Linux, only libpcap is required. On Windows, we use the Winpcap project which includes libpcap but also includes Windows specific drivers. Since the winpcap project only provides Visual Studio project files, which may be out dated, the superbuild does not attempt to compile winpcap. Instead, a git repository containing headers and precompiled .lib and .dll files is used. The repository url can be found in Superbuild/versions.cmake

1.3.2 Boost

The required boost version is 1.50. Boost is used for threading and synchronization, and for network communication. The boost thread and asio libraries are used.

1.3.3 Qt

The required Qt version is 4.8. Qt is a desktop widget library that is used to provide user interface elements like windows and menus across the supported platforms Windows, Mac, and Linux.

1.3.4 Python

The required Python version is 2.7. VeloView uses libpython to embed a Python interpreter in the VeloView application. The core VeloView features are implemented in C++ libraries, and the libraries are wrapped for Python using VTK's Python wrapping tools.

1.3.5 PythonQt

PythonQt is used to build Qt applications from Python. PythonQt has support for wrapping types derived from Qt objects and VTK objects. PythonQt is used from a specific git branch, to mach the Qt version.

1.3.6 VTK and ParaView

The required VTK version is 7.0. The required ParaView version is 5.1. The ParaView repository includes VTK, so the superbuild only needs to checkout and build ParaView in order to satisfy both dependencies. A specific git commit sha1 is used instead of a specific released version. The commit sha1 is very similar to the mainstream release version but it has a few commits from the ParaView master branch cherry-picked onto it. The commits added are those that resolve some issues with the Python console and add the PythonQtPlugin for ParaView. The PythonQtPluginis a small plugin that initializes the PythonQt library and makes it available in the ParaView Python console.

1.4 Configure and build instructions

The superbuild requires cmake version 2.8.8. The build will be performed in a separate, out-of-source build directory. Start a terminal in the parent directory that contains your veloview checkout. Create a new build directory and then use cmake to configure the superbuild:

mkdir build cd build cmake -DENABLE_veloview:BOOL=ON ../veloview/SuperBuild

This will configure the veloview superbuild with the VeloView application enabled, and all other options left at default. For most builds, the default options are satisfactory. Some users may wish to configure custom cmake options in order to select system versions of certain third party dependencies such as Qt, Boost, or Python. You should not use a system version of pcap, instead, let the superbuild checkout the correct version of pcap or winpcap.

You can use cmake, ccmake (a curses interface to cmake) or cmake-gui.

You can set the CMAKE_BUILD_TYPE to Release, Debug, or RelWithDebInfo to set the build type. Most users will want to select Release.

You can set the CMAKE_INSTALL_PREFIX to determine where the VeloView binaries are installed when you run make install.

After cmake has generated the build files, you can compile the superbuild using:

cmake --build .

1.4.1 Windows build instructions

The superbuild has been tested on Visual Studio 11 (2012). It might work with other versions of Visual Studio, but be warned that they have not been tested with VeloView.

You can build VeloView for 32bit or 64bit. The target architecture is decided at CMake configure time, when CMake is launch from the MS Visual Studio "Cross-tool command prompt" environment. When selecting the generator in cmake-gui on Windows, you should select the appropriate version.

After generating the Makefiles, just run *make* or *cmake --build .* to run the superbuild. NMake and Visual Studio does not support parallel builds, so the build can take quite some time to complete on Windows, especially when compiling Qt instead of using a system install of Qt.

1.4.2 Mac build instructions

For Mac builds, it is best to use system installs of Qt and Python. You can use a package manager like Homebrew or macports to install these libraries system wide prior to compiling VeloView. The system version of pcap on Mac is too old to be used with VeloView, so the superbuild will always download and compile the correct version of pcap. You can choose to build Boost with the superbuild or use a system version of Boost, as long as the static Boost archive libraries are available (the libraries with the .a extension). If you are unsure, it is better to let the superbuild build Boost for you.

cmake -DENABLE_veloview:BOOL=ON -DCMAKE_OSX_ARCHITECTURES:STRING="x86_64" -DCMAKE_OSX_SYSROOT:STRING=/path/to/XcodeSDK/Developer/SDKs ../SuperBuild

1.4.3 Linux build instructions

These steps are for Ubuntu 16.04.4 LTS. First, install the following dependencies using the apt-get command. This is the full list used in the tested setup:

```
git
cmake-curses-gui
build-essential
libboost-all-dev
libxt-dev
libbz2-dev
libqt4-dev
qt4-default
qt4-dev-tools
zlib1g-dev
```

By default, Ubuntu packages version 3.5.1 of CMake, so you should be fine.

On Linux, libpcap can either be installed as a package or built from source. If you wish to build it from source you will need to apt-get install flex and byacc. If you want to use the packaged version, apt-get install libpcap-dev. If you're unsure, build it from source.

Clone the git repository, and from a separate build directory configure using CMake by pointing it at the Superbuild directory:

```
ccmake <VeloView>/Superbuild
```

Enable the following options in the CMake configuration:

```
ENABLE_veloview=ON
USE_SYSTEM_boost=ON
USE_SYSTEM_python=ON
USE_SYSTEM_qt=ON
```

Also be sure to set the *USE_SYSTEM_pcap* to the appropriate value, depending on whether you chose to use the system package or build it from source.

If you have both Qt4 and Qt5 on your system, you will need to choose Qt4 before building. To do that, run the following in bash:

```
export QT_SELECT=qt4
```

Then run *cmake --build .*, *make* or *make -jN* as usual to run the superbuild.

1.5 Packaging

After building VeloView, the application will be runnable on your system. In order to distribute VeloView to other users you must generate a VeloView package. The packaging process is different for different platforms.

1.5.1 Packaging for Windows

Packaging on Windows requires NSIS. Visit the NSIS website to download and install the latest version. NSIS is used to generate a standard Windows installer executable which will install VeloView to the Program Files directory. Make sure you install NSIS before configuring VeloView with CMake. After the superbuild has completed (you ran make and it completed without errors) you are ready for packaging.

Before packaging, you might want to test the VeloView install tree. You can run the make install command (make sure you have set the CMAKE_INSTALL_PREFIX to a writable location) and then cd to the install directory and open bin/VeloView.exe. If there are any issues, you should debug them at this point before continuing with the packaging. Make sure you open the VeloView Python console to make sure there are no issues with Python initialization.

To generate a Windows installer, run the package command:

    make package

The output will be a .exe installer in the current directory.

1.5.2 Packaging for Mac

Packaging on Mac will generate a .dmg image file. Opening the .dmg file will mount a volume that contains the VeloView.app bundle. There is already a VeloView.app bundle in your build tree, but it only contains the veloview binary and not any dependent libraries. A real app bundle contains library files for all the veloview dependencies. After copying the dependent library files into the app bundle, a script runs the Mac tool called install_name_tool to rewrite the library dependency locations using relative paths. The script is in the veloview repo named fixup_bundle.py and it is executed automatically during installation and packaging.

Before packaging, you might want to test the VeloView install tree. You can run the make install command (make sure you have set the CMAKE_INSTALL_PREFIX to a writable location) and then cd to the install directory and open VeloView.app. If there are any issues, you should debug them at this point before continuing with the packaging. Make sure you open the VeloView Python console to make sure there are no issues with Python initialization.

To generate a Mac installer, run the package command:

```
make package
```

The output will be a .dmg file in the current directory.

2 Developer Guide

2.1 Source code organization

The VeloView source code is a mixture of VTK classes and Qt classes. The source code files with the *vtk* prefix are VTK classes that do not have any Qt dependencies. The classes with the *vv* or *pq* prefixes are Qt classes that depend on VTK, Qt, and ParaView's Qt libraries. The core VTK classes in VeloView are compiled into a plugin library named *libVelodyneHDLPlugin* that can be loaded into ParaView. The VeloView app is implemented using a mixture of the C++ Qt classes and Python code. The Python code is mostly organized in the file *applogic.py* in the veloview Python module.

2.2 ParaView plugin

The *libVelodyneHDLPlugin* library depends on VTK, ParaView, Qt, PythonQt, Boost, and libpcap. The plugin can be loaded into the main ParaView application using ParaView version 4.0. The build specifies the static version of the boost libraries, so the plugin's only dependencies beyond ParaView are libpcap and PythonQt library.

On Windows, the plugin can be loaded as long as the libpcap and PythonQt library dll files are in the same directory. On Mac, you should use the install_name_tool to fix the library locations of these dependencies to be relative to @loader_path, then place the libpcap and PythonQt library files relative to the libVelodyneHDLPlugin library.

In ParaView, the Velodyne pcap reader and Velodyne network source plugin are available in the *Sources* menu.

2.3 VTK readers

VeloView, and the VelodyneHDL Plugin for ParaView included two readers/sources. The Velodyne pcap reader is implemented in the C++ class vtkVelodyneHDLReader.{cxx,h}. When reading a pcap file, the reader first scans the file and looks for frame splits when the azimuth resets from 360 degrees to 0 degrees. The pcap file position is recorded for each split so that the reader can jump to frames using file seeking.

The network source reader receives UDP data packets from a Velodyne sensor using the Boost asio library. The network source is implemented by vtkVelodyneHDLSource.{cxx,h}. The source manages multiple threads in a producer/consumer model, and uses an instance of the vtkVelodyneHDLReader to convert data packets into VTK point cloud data.

2.4 VeloView application

The VeloView application is implemented using Qt in C++ and Python. The PythonQt library is used to access the C++ layer from Python. The majority of the application logic is implemented in Python in the *applogic.py* file. The Python code also uses Qt's uitools library to load user interface *.ui* files at runtime. Qt designer can be used to edit the .ui files. The VeloView application can be extended using Python and .ui files.