**CuteDemo – Building the environment, application and install program**

# Introduction

For Windows version, the following examples will build a Win32 AWLCuteDemo application.

The example builds using a Visual Studio project.

There are tools available to build exclusively via CMake. (Documentation will follow).

In the examples, the use of “forward slash” character in paths is important, as the “backward slash” can affect some tools (such as CPack).

# Building the Libraries and Environment

Install CMAKE

* <https://cmake.org/download/>

Install version of Visual Studio

* (used in this document : Visual Studio 2019 Community Edition)

Download and install Boost Libraries

* Version of BOOST must be < 1.68 (Recommended 1.6.0)
* <https://sourceforge.net/projects/boost/files/boost-binaries/1.60.0/boost_1_60_0-msvc-14.0-32.exe/download>
* Build the Libraries
  + “Boostrap.bat”
  + ./b2 install --prefix=E:/DevTools/boost\_1\_60\_0/boost-build
* Set the Windows Environment Variable BOOST\_DIR=E:/DevTools/boost\_1\_60\_0

Download and install Open CV

* Version used in demo is 3.1.0
* Pre-built binaries at: <https://sourceforge.net/projects/opencvlibrary/files/opencv-win/3.1.0/opencv-3.1.0.exe/download>
* Set the Windows Environment Variable OPENCV\_DIR=E:\DevTools\opencv-3.4.6.0\build
* Version used in demo is 3.4.6
* Pre-built binaries at: <https://sourceforge.net/projects/opencvlibrary/files/3.4.6/opencv-3.4.6-vc14_vc15.exe/download>
* By executing the executable, the Libraries may end in a directory called plainly “OpenCV”. For the sake of clarity and long term maintenance, take the time to add the version number to the supplied name by renaming the newly created directory: “opencv-3.4.6.0”
* You may have to build the Win32 version of the Libraries, by following the instructions at <https://docs.opencv.org/3.4.6/d3/d52/tutorial_windows_install.html>
  + This creates a Visual Studio Project E:/DevTools/OpenCV-3.4.6.03build/OpenCV.sln
  + Build Open CV for the environments you need.
  + In CMake\_Gui, make sure you click the INSTALL\_CREATE\_DISTRIB option and the BUILD\_opencv\_world options.
    - This option is required to build OpenCV\_World
  + In CMake, click the WITH\_XIMEA option, if you want to build for older Phantom Intelligence units
    - which is used to interface with XIMEA camera used in some AWL sensors
* Set the Windows Environment Variable OPENCV\_DIR=E:/DevTools/OpenCV-3.4.6.0/build

Download and install Qt:

* Download the Open-source version under <https://www.qt.io/download>.
  + Current version as of this writing is Qt 5.12.4
* Follow the QT instructions provided in the downloadable executable for installation
* Set the Windows Environment Variable QTDIR=E:/DevTools/Qt/Qt5.12.4/5.12.4/msvc2017

Download and install QWT

* Download QWT from <https://sourceforge.net/projects/qwt/>
  + Current version as of this writing is QWT 6.1.4
* Unzip the Sources to your target directory (here: E:\DevTools. Unzip will create directory \qwt-6.1.4)
  + Make sure that your destination paths for the library file are set correctly to your liking, by editing the file qwt.config.pri. All files will be copied to an installation directory, that is configurable by editing qwtconfig.pri.
  + Its default settings, under Window, is:
    - C:/Qwt-6.1.4
  + Which, for our example, was modified to e:/Devtools/Qwt-6.1.4
    - win32 {
    - QWT\_INSTALL\_PREFIX = e:/Devtools/Qwt-$$QWT\_VERSION
    - }
* Create a makefile using Qt’s QMake tool:
  + E:/DevTools/Qt/Qt5.12.4/5.12.4/msvc2017/bin/qmake qwt.pro
* Make the libraries
  + nmake install
* Set the Windows Environment Variable QWTDIR=E:/DevTools/qwt-6.1.4

Download the Kvaser CANLib SDK

* <https://www.kvaser.com/developer/canlib-sdk/> or
* E:/DevTools/canlib.exe
* Drivers will be installed in your C:/Program Files/Kvaser/Drivers
* Set the Windows Environment Variable CANLIB\_DIR= C:/Program Files (x86)/Kvaser/Canlib

Download the XIMEA APIs for OpenCV

* XIMEA API is only required if you want to interface with XIMEA cameras (requires OpenCV)

Download LiBUSB

* <https://github.com/libusb/libusb/releases> (current version at this writing 1.023)
* Use the appropriate project from the “msvc” directory to build your platform.
* Set the Windows Environment variable LIBUSB\_DIR =E:/DevTools/libusb-1.0.23

Download and install the Wix Toolset to create installation programs

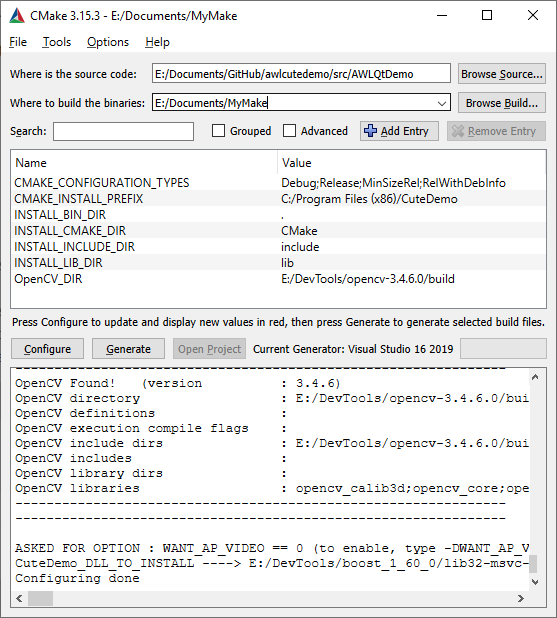
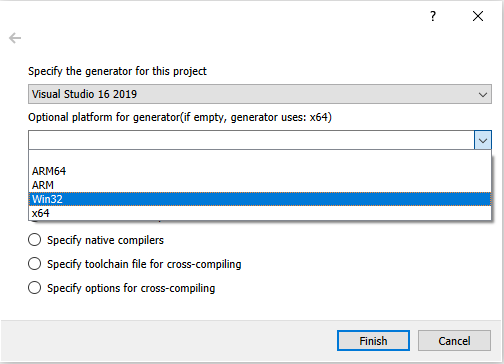
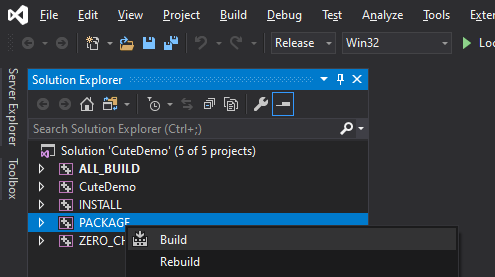
* <https://wixtoolset.org/releases/>

## Building the application

Building the application under Windows – Using Visual Studio

* Open the Visual Studio Solution located at src/AWLCuteDemo.sln
* Make sure that the path to Libraries are set up correctly in the project properties.
  + By default, the project refers to the pre-built binaries in E:/DevTools and assumes that the Windows environment variables are set as described above.

Building the application and installer

* Using CMake, open the src/AWLCuteDemo/CMakeLists.txt
  + Our current example uses the CMake-Gui, available at <https://cmake.org/download/>
  + (command line are also acceptable)
* Specify a separate target directory for “Where to build the binaries:”
  + The example here is using “E:/MyDocuments/MyMake”  
    
* Press on the “Configure” button.
  + Specify Win32 as a target platform for the build  
    
* Press “Configure” again, and then Generate
  + This step creates a full Visual Studio Project for the code, along with a Wix Installer project, in the directory that you have previously identified as your target for the binaries.
* Open the Visual Studio solution that was created under your target directory
  + In our example “E:/MyDocuments/MyMake/CuteDemo.sln”
* “Build All” the solution
  + First, make sure that the Target selected is “Release-Win32”
    - Because of different DLL structure, building the “Debug” targets will fail.
  + Menu “Build/Build Solution”
* Click on the “PACKAGE” project to select it. Right-click and select the “Build” menu item.  
  
* This should create an install package with the MSI extension under your target directory

# Notes:

Builds for Windows on Visual Studio should target the Win32 Platform

In the Visual Studio project, click on each of the individual projects. Then go the menu “Project / Properties”. Make sure that the following accurately represent your directory structure for the packages (do not forget to do it for each of the projects platforms -debug and release):

“C++/General/Additional Include Directories”

“Linker/General/Additional Library Directories”

“Linker/Input/Additional Dependencies”

For execution and debugging of the AWLQTDemo project from within Visual Studio, the path to DLLs must be set in the “Project / Properties”. The values indicated in the following are based on the current Pre-Built configuration:

“Configuration properties/Debugging/Environment”

PATH=$(PATH);$(QTDir)\bin;$(QTDir)\plugins\platforms\;$(OPENCV\_DIR)\bin\$(Configuration)\;$(QWTDIR)\lib\;$(XIMEA\_DIR)\API\x86\;$(CANLIB\_DIR)\bin\;$(LIBUSB\_DIR)\Win32\$(Configuration)\dll\;

When building the solution in Visual Studio, you may get warnings:

“Unknown compiler version - please run the configure tests and report the results”

This is caused by incomplete support of some later Visual Studio versions in Boost.