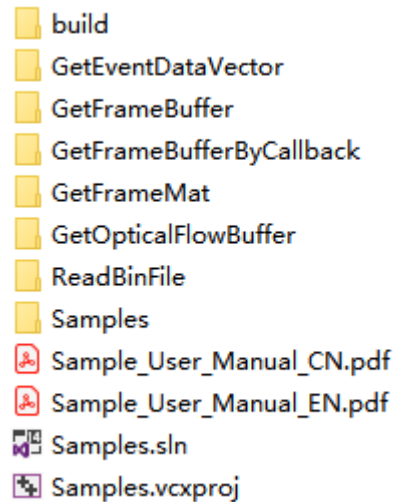


1 Introduction

The file directory of *Samples* is as follows:



There are eight sub directories in the *Samples*. The six directories are the main function instance code for the CeleX™ Sensor. The last directory mainly includes some necessary configure file, include files and library. More related content is introduced as follows:

1.1 GetEventDataVector

It shows how to get the (X, Y, A, T) information and use it to create an image frame.

1.2 GetFrameBuffer

It shows how to set the working mode of the CeleX™ Sensor and obtain the data that the CeleX™ Sensor works in different modes.

1.3 GetFrameBufferByCallback

It shows how to set the working mode of the CeleX™ Sensor and register to monitor the data that the CeleX™ Sensor works in different modes.

1.4 GetFrameMat

It shows how to set the working mode of the CeleX™ Sensor and obtain the data in cv::Mat form that the CeleX™ Sensor works in different modes.

1.5 GetOpticalFlowBuffer

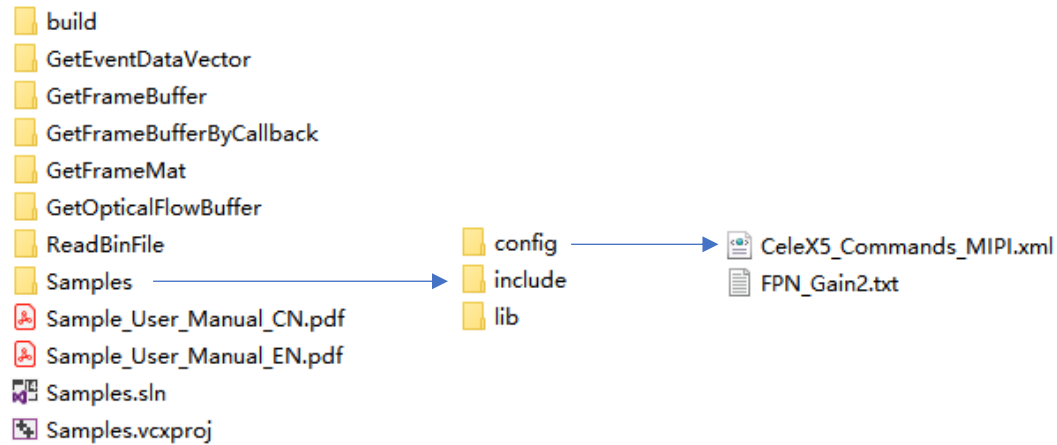
It shows how to get the optical flow data.

1.6 ReadBinFile

It shows how to read the recorded bin file and display it.

1.7 Samples

There are three folders in the *Samples* folder:



1.7.1 include

The header files of the API are placed in the *include* directory.

1.7.2 lib

The API libraries is included in the *lib* directory (including dynamic link library of 64-bit Windows and Linux).

1.7.3 config

There are a FPN file and a configuration file in *config* directory that CeleXTM Sensor will use when starting up. To run the program developed by CeleXTM Sensor library, you must copy the following configuration file to the directory where the executable file is located, or the program will not be executed. In these samples, these files are automatically copied to the build directory, and users do not need to manually copy these files.

Notes:

Users generally do not need to modify these configuration parameters. They can be adjusted by calling the APIs. For details, please refer to the API User Manual.

The OpenCV library is not included in the samples, so to use these, it need to install OpenCV first.

2 Compile Samples

You can compile and run these sample codes in Windows or Linux. There are some differences when compiling under Windows and Linux. The detailed description is as follows:

2.1 Windows

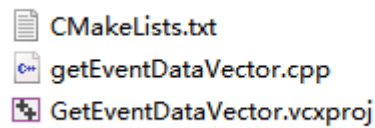
The samples are developed by the Visual Studio 2015 under Windows 10. Before compiling the sample code, you need to install OpenCV first. The OpenCV version used in samples is 3.3.0. The current OpenCV include directory and lib directory are under the local D:\Program Files\opencv. You need to specify the include and lib path to your OpenCV installation directory in the property list.

With Visual Studio 2015, you can open the project by opening the .sln or .vcxproj file. You can

compile and run any one sample by setting this as startup item.

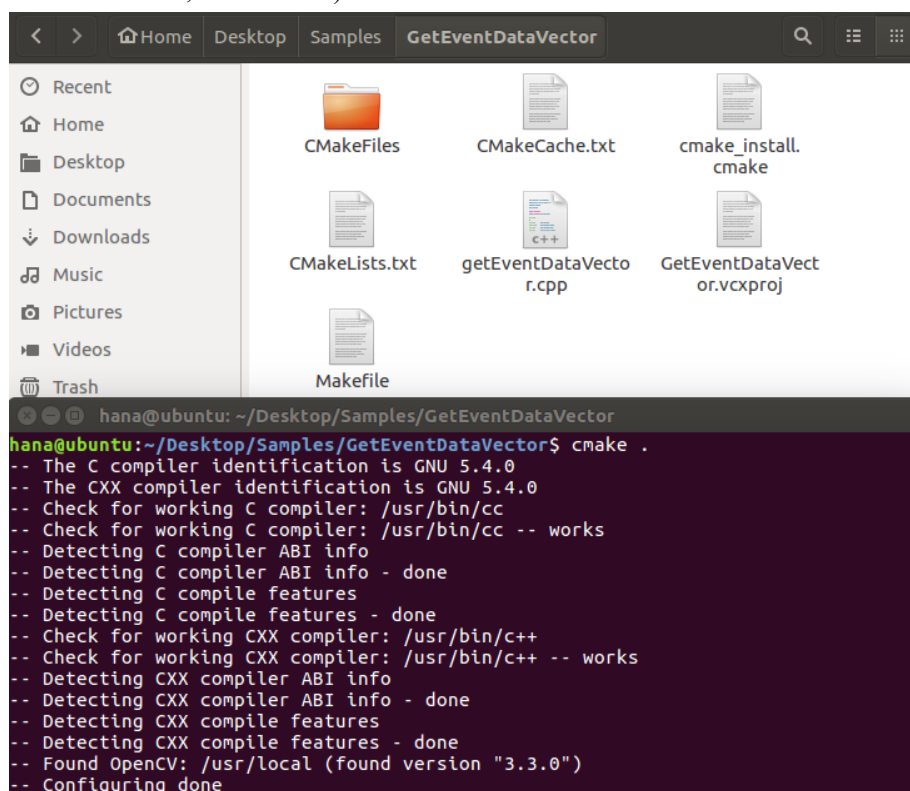
2.2 Linux

You can also compile and run these sample code in Linux. Each sample has the following three files:

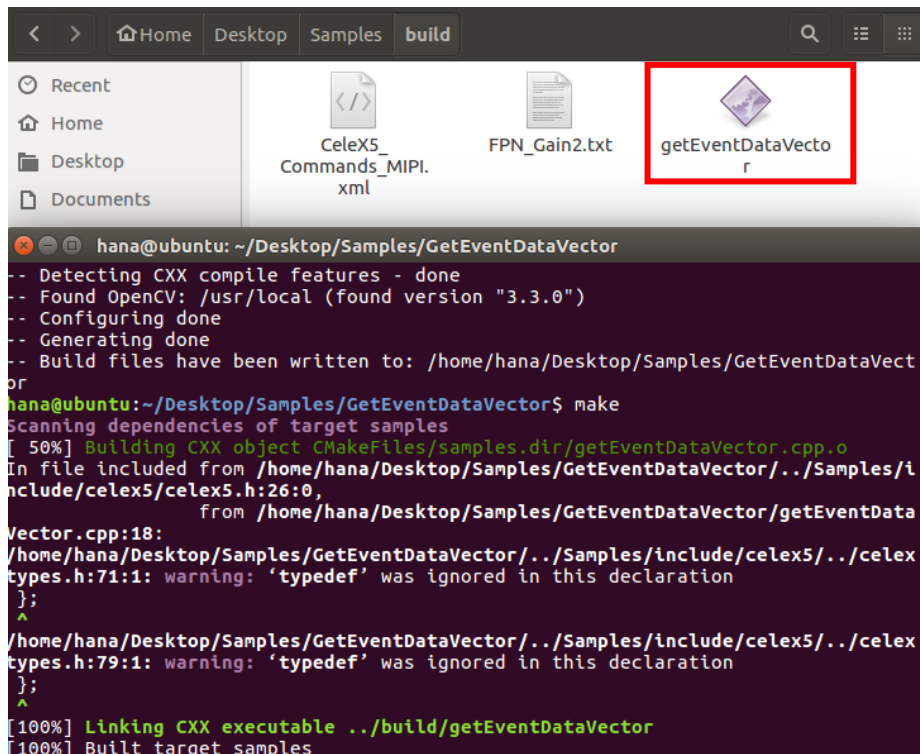


The CMakeList.txt is used in Linux, .cpp file is the source code and the .vcxproj file is used in Windows.

In Linux, you can use the CMakeList file to compile the samples. Note: Before compiling, check if there is a Linux library file in the **Samples\Samples\lib\Linux** directory (including dynamic link library: libCeleDriver.so, libCeleX.so).



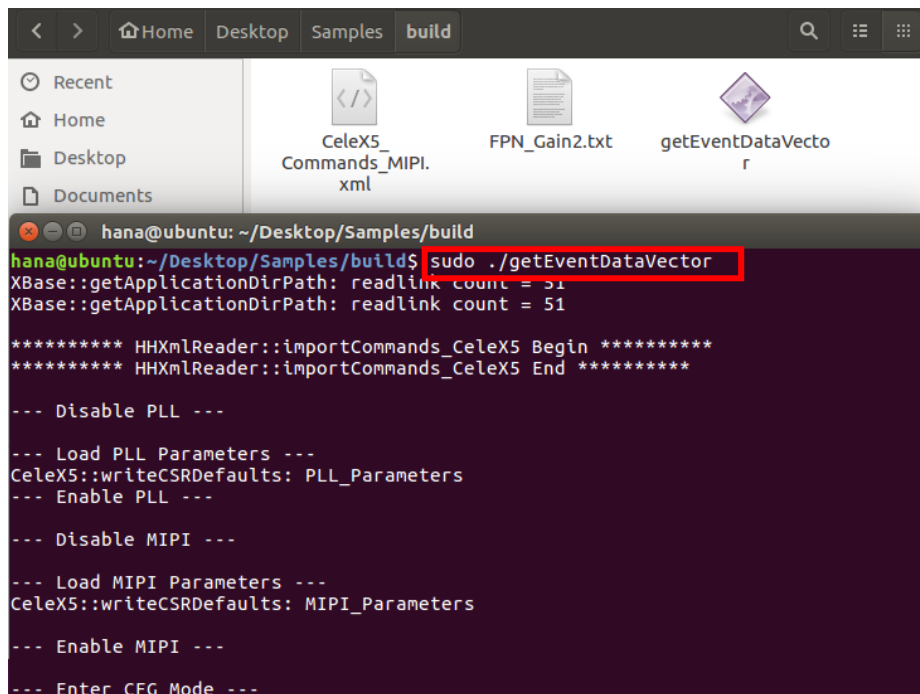
Then if there is no link error in the include and lib directories, you can use make to compile.



The screenshot shows a file manager window with the 'build' directory selected. It contains files 'CeleX5_Commands_MIPI.xml', 'FPN_Gain2.txt', and 'getEventDataVector'. Below the file manager is a terminal window showing the output of the 'make' command. The terminal output indicates that the build files have been written to the specified path, and the 'make' command has successfully built the target samples, including the 'getEventDataVector' executable.

```
hana@ubuntu: ~/Desktop/Samples/GetEventDataVector
-- Detecting CXX compile features - done
-- Found OpenCV: /usr/local (found version "3.3.0")
-- Configuring done
-- Generating done
-- Build files have been written to: /home/hana/Desktop/Samples/GetEventDataVector
hana@ubuntu:~/Desktop/Samples/GetEventDataVector$ make
Scanning dependencies of target samples
[ 50%] Building CXX object CMakeFiles/samples.dir/getEventDataVector.cpp.o
In file included from /home/hana/Desktop/Samples/GetEventDataVector/./Samples/include/celeX5/celeX5.h:26:0,
                 from /home/hana/Desktop/Samples/GetEventDataVector/getEventDataVector.cpp:18:
/home/hana/Desktop/Samples/GetEventDataVector/./Samples/include/celeX5/./celeXtypes.h:71:1: warning: 'typedef' was ignored in this declaration
};
^
/home/hana/Desktop/Samples/GetEventDataVector/./Samples/include/celeX5/./celeXtypes.h:79:1: warning: 'typedef' was ignored in this declaration
};
^
[100%] Linking CXX executable ../build/getEventDataVector
[100%] Built target samples
```

The executable file is generated to the **build** directory. There is a configuration file in **build** directory that CeleX™ Sensor will use when starting up. Then you can run the generated file. If the operation fails, check if the necessary files are included in the current running file directory. (Sensor needs to read and write to USB devices, so you need to use root permissions)



The screenshot shows the same file manager window as before, but now the 'build' directory is selected. The terminal window shows the command 'sudo ./getEventDataVector' being executed. The output of the command shows the application's initialization process, including disabling PLL and MIPI, loading parameters, and entering CFG Mode.

```
hana@ubuntu: ~/Desktop/Samples/build
hana@ubuntu:~/Desktop/Samples/build$ sudo ./getEventDataVector
XBase::getApplicationDirPath: readlink count = 51
XBase::getApplicationDirPath: readlink count = 51

***** HHXMLReader::importCommands_CeleX5 Begin *****
***** HHXMLReader::importCommands_CeleX5 End *****

--- Disable PLL ---

--- Load PLL Parameters ---
CeleX5::writeCSRDefaults: PLL_Parameters
--- Enable PLL ---

--- Disable MIPI ---

--- Load MIPI Parameters ---
CeleX5::writeCSRDefaults: MIPI_Parameters

--- Enable MIPI ---

--- Enter CFG Mode ---
```