OBSIDIAN

# Miramar Camera UVC Interface Application note

*Rev2 – Jun 2024*
*Rev1 – Sep 2023*

## Contents


INTRODUCTION ..... 2

UVC STREAMING ON WINDOWS ..... 2

OpenCV ..... 2
FFmpeg ..... 2

UVC STREAMING ON LINUX ..... 3

OpenCV ..... 3
GStreamer ..... 3

# Introduction

A Miramar camera outputs video through USB port using UVC protocol. The video can be viewed by camera apps on PC or cell phone. Data is perceived by the operating system as in 16-bit YUYV format. Raw 16-bit data can be sent if colormap is turned off. Any backend or framework compatible with UVC devices can be used to grab frames from a Miramar camera. For most of our examples we will be using OpenCV.

Depending on host hardware, OS or runtime environment the user will need to follow some steps. These issues and their solutions will be documented here.

# UVC Streaming on Windows

## OpenCV

When we last tested the Miramar camera in this environment, OpenCV for C++ had a documented bug which prevented the system from using the CAP_MSMF video backend. Until the bug is fixed, OpenCV needs to be built with specific parameters with these 5 lines (edit paths if different):

```
git clone --branch 4.6.0 https://github.com/opencv/opencv.git

mkdir build

cd build

cmake.exe -G "Visual Studio 17" -DCMAKE_INSTALL_PREFIX=%CD%\..\install -DBUILD_SHARED_LIBS=ON
-DBUILD_PERF_TESTS=OFF -DBUILD_TESTS=OFF -DBUILD_EXAMPLES=ON -
DOPENCV_INSTALL_DATA_DIR_RELATIVE=%CD%\..\install -DINSTALL_CREATE_DISTRIB=ON -DBUILD_DOCS=OFF
-DBUILD_opencv_python3=OFF -DBUILD_opencv_python2=OFF -DBUILD_opencv_java=OFF -
DVIDEOIO_PLUGIN_LIST= ../opencv
cmake.exe --build . --config Release --target install
%CD%\..\install\x64\vc17\samples\cpp\example_cpp_example.exe
```

After building OpenCV and linking the library dependencies in the development environment, running the C++ frame grabbing code in the SDK should run without issue. This bug has been fixed OpenCV 4.9.0: https://github.com/opencv/opencv/issues/23056
There are no issues grabbing frames on the Miramar camera using OpenCV for Python. The example frame grabbing code in the SDK should run without issue.

## FFmpeg

In OpenCV on Windows, the camera frames are captured using CAP_MSMF by default. However, virtually any proper UVC compatible backend can be used. For instance, the following line will use a local FFmpeg binary exe to grab 120 frames at 30fps from the Miramar camera as a DirectShow UVC device to a file “output.yuv”:

```
C:\PATH\TO\ffmpeg -f dshow -i video="Miramar Camera" -vframes 120 -r 30 C:\PATH\TO\output.yuv
```

# UVC Streaming on Linux

## OpenCV

The Linux video backend “V4L2” may cause errors where USB packets are dropped when working with some UVC devices, including our Miramar camera. This causes the host system to drop video frames. To fix this issue, the *uvcvideo* module needs to be reloaded with a *nodrop=1* option, as follows:

```
$ rmmod uvcvideo
$ modprobe uvcvideo nodrop=1
```

The module nodrop parameter can also be edited without reloading, as follows:

```
$ echo 1 > /sys/module/uvcvideo/parameters/nodrop
```

At this point, any UVC capturing should work. The example frame grabbing code in the SDK should run without issue (the C++ OpenCV default build should work, as CAP_MSMF is exclusive to Windows environments).

## GStreamer

To list all associated devices for the Miramar camera, run the following line: The module nodrop parameter can also be edited without reloading, as follows:

```
$ v4l2-ctl --list-devices | grep -A3 Miramar
```

Running the following line will grab frames and save them to an “mp4” file. The camera should be streaming with colormap mode enabled at this point to grab real images.

```
$ gst-launch-1.0 v4l2src device=/dev/video0 ! v4l2h264enc ! filesink location=test.mp4
```