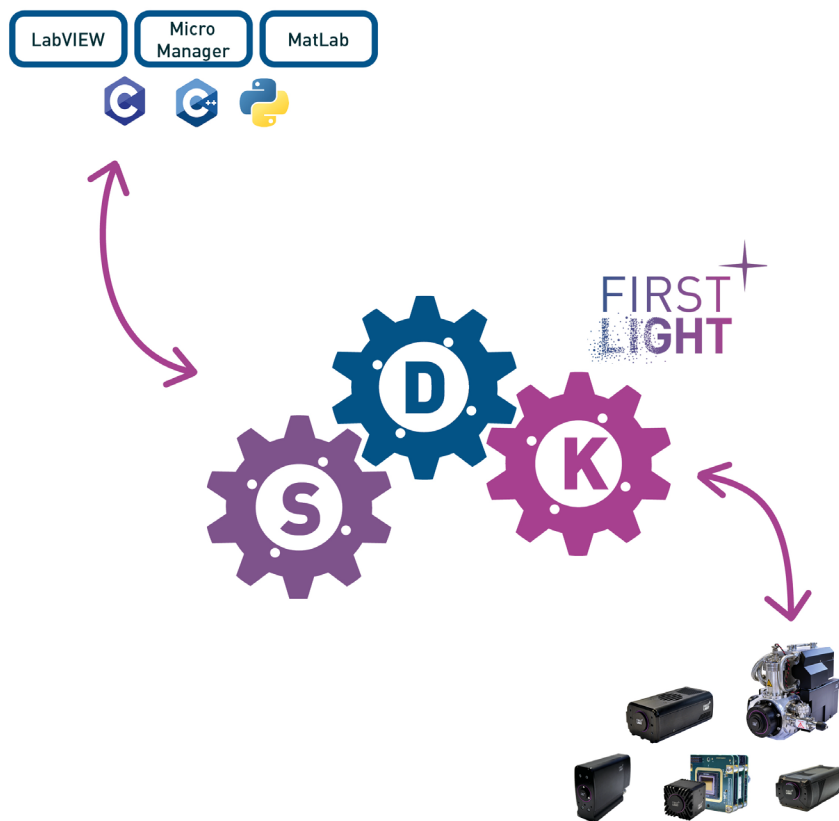


# FLI Software Development Kit

## User Manual

SDK User Manual\_20230207



### REVISION HISTORY

Issue	Date	Para	Details
<b>1.0.0</b>	21/11/2019		Initial release
<b>1.2.1</b>	21/04/2020		Addition of C-RED One, MATLAB and save formats.
<b>1.2.2</b>	14/09/2020		Image update
<b>1.3.1</b>	22/09/2020		MAJ SDK 1.3.1
<b>1.4.0</b>	13/01/2021		MAJ SDK 1.4.0
<b>2.1.1</b>	27/05/2021		MAJ SDK 2.1.1

# TABLE

1. Introduction	3
2. Installation	3
3. How to use the SDK	4
3.1. Setting the environment.....	4
3.2. Example project.....	4
3.3. C++ API.....	7
3.4. C API.....	7
3.5. Python (for Windows and Linux).....	7
3.6. Save buffer formats.....	8
3.7. Frame grabbing explained.....	9
3.7.1. Poll the frames	9
3.7.2. Observer	9
3.8. LabView (only for Windows).....	11
3.9. MATLAB (for Windows).....	12
4. Plugins	13
4.1. Micro Manager (for Windows and Linux).....	13

# 1. INTRODUCTION

First Light SDK is an SDK developed by First Light Imaging.

It allows the users to connect or develop their own GUI to control the First Light Imaging Cameras.

The SDK is provided with some development languages API (C/C++/Python/LabView/MATLAB) and some plugins to interface with other software (Micro Manager and more to come).

The SDK can be used with C-RED One, C-RED 2 and C-RED 2 ER, C-RED 3, OCAM<sup>2</sup> and C-BLUE One cameras on Windows 10, Linux Ubuntu 16.04 LTS and Linux Ubuntu 18.04 LTS (depending on grabber).

In addition, the SDK is also available for C-RED 2 and C-RED 3 for Jetson Tx2, Nano, Xavier NX and Xavier AGX. Please note, that only the USB3 interface is available on these **plat**forms.

# 2. INSTALLATION

During the installation of the SDK you will be able to choose what you want to install:

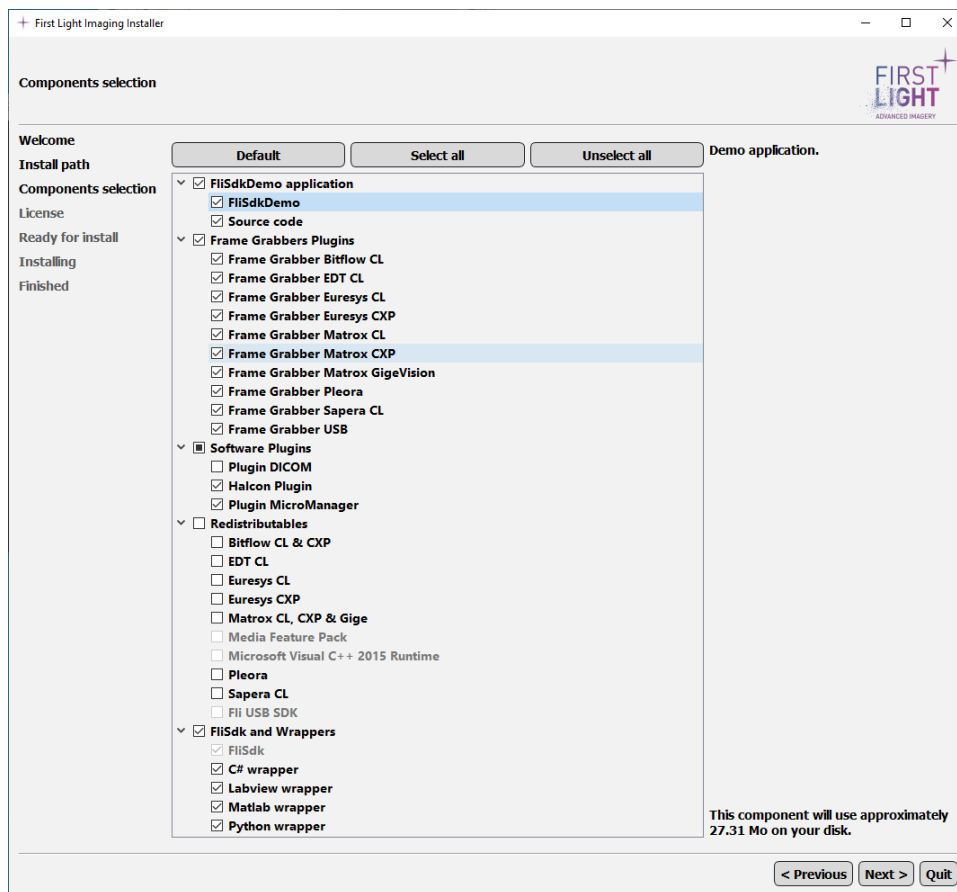


Fig. 1: Installation page

**Note:** For more detail on the installation, please use the relevant associated documentation. There are dedicated documents for the various cases.

## 3. HOW TO USE THE SDK

### 3.1. Setting the environment

To set the project environment you can use the environment variable FLISDK\_DIR.

This environment variable is set during the SDK installation and available after the reboot following the installation.

It points to the installation directory of the SDK and should never be changed.

- FLISDK\_DIR/lib/release for .dll and .lib
- FLISDK\_DIR/include for the .h

### 3.2. Example project

An example project is available in the installation folder.

Prior to using it, you have to install Qt Framework (Qt Creator included).

The demo example is known to work with Qt 5.15.2 LTS and you can get Qt from the following location:

<https://www.qt.io/download>.

Then, copy the Demo\_sources directory on a location in which you have writing rights (for example your desktop) and open "FliSdkDemo.pro".



### Configure Project

The following kits can be used for project **Demo\_sources**:

The project **FliSdkDemo** is not yet configured.

Qt Creator uses the kit **Desktop Qt 5.12.3 MSVC2015 64bit** to parse the project.

Type to filter kits by name...

Select all kits

<input checked="" type="checkbox"/>	<b>Desktop Qt 5.12.3 MSVC2015 64bit</b>	Détails ▲	
<input checked="" type="checkbox"/>	Debug	C:\Users\jtugnoli\Desktop\build-FliSdkDemo-Desktop_Qt_5_12_3_MSVC2015_64bit-Debug	Parcourir...
<input checked="" type="checkbox"/>	Release	C:\Users\jtugnoli\Desktop\build-FliSdkDemo-Desktop_Qt_5_12_3_MSVC2015_64bit-Release	Parcourir...
<input checked="" type="checkbox"/>	Profile	C:\Users\jtugnoli\Desktop\build-FliSdkDemo-Desktop_Qt_5_12_3_MSVC2015_64bit-Profile	Parcourir...
<b>Importer la compilation depuis...</b>			Détails ▼

Configure Project

Fig. 2 : Qt configuration page

## Configure your project:

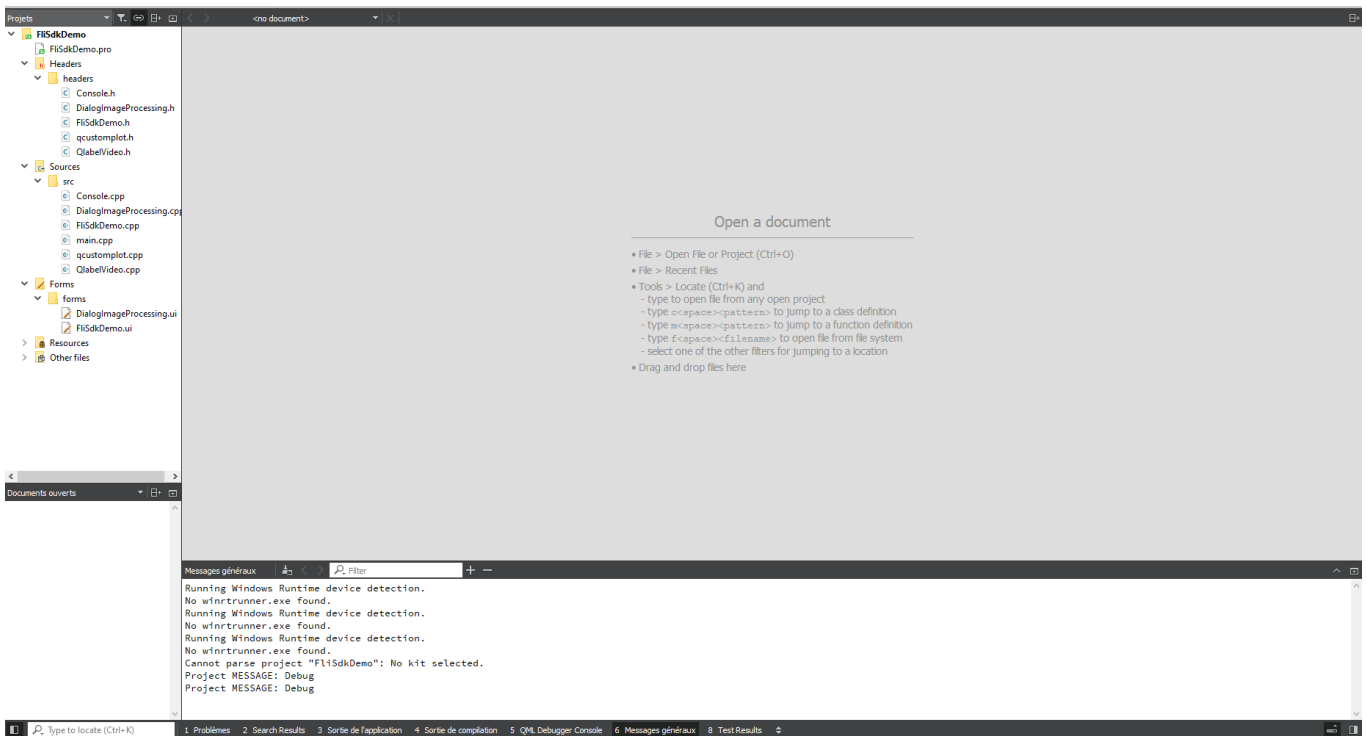


Fig. 3: Qt project view

## Compile the project:

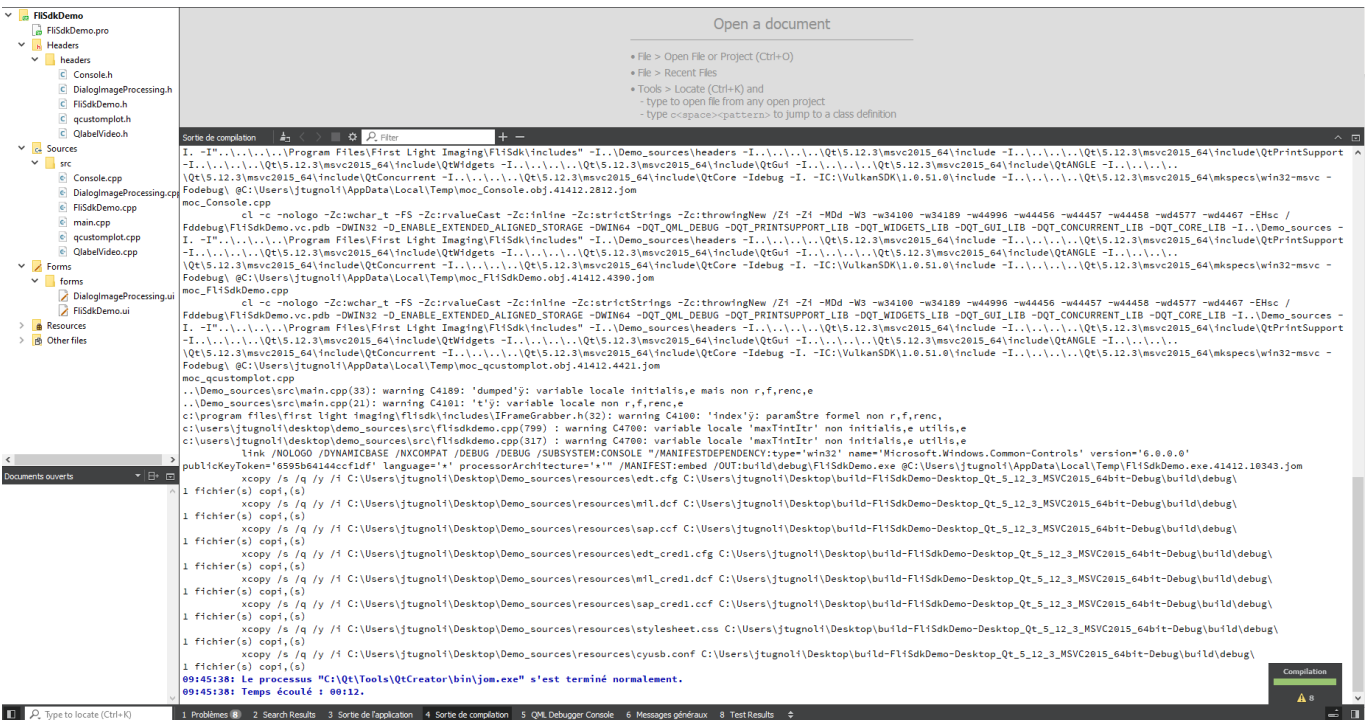


Fig. 4: Qt compilation view

## Connect a camera to the computer and start the software:

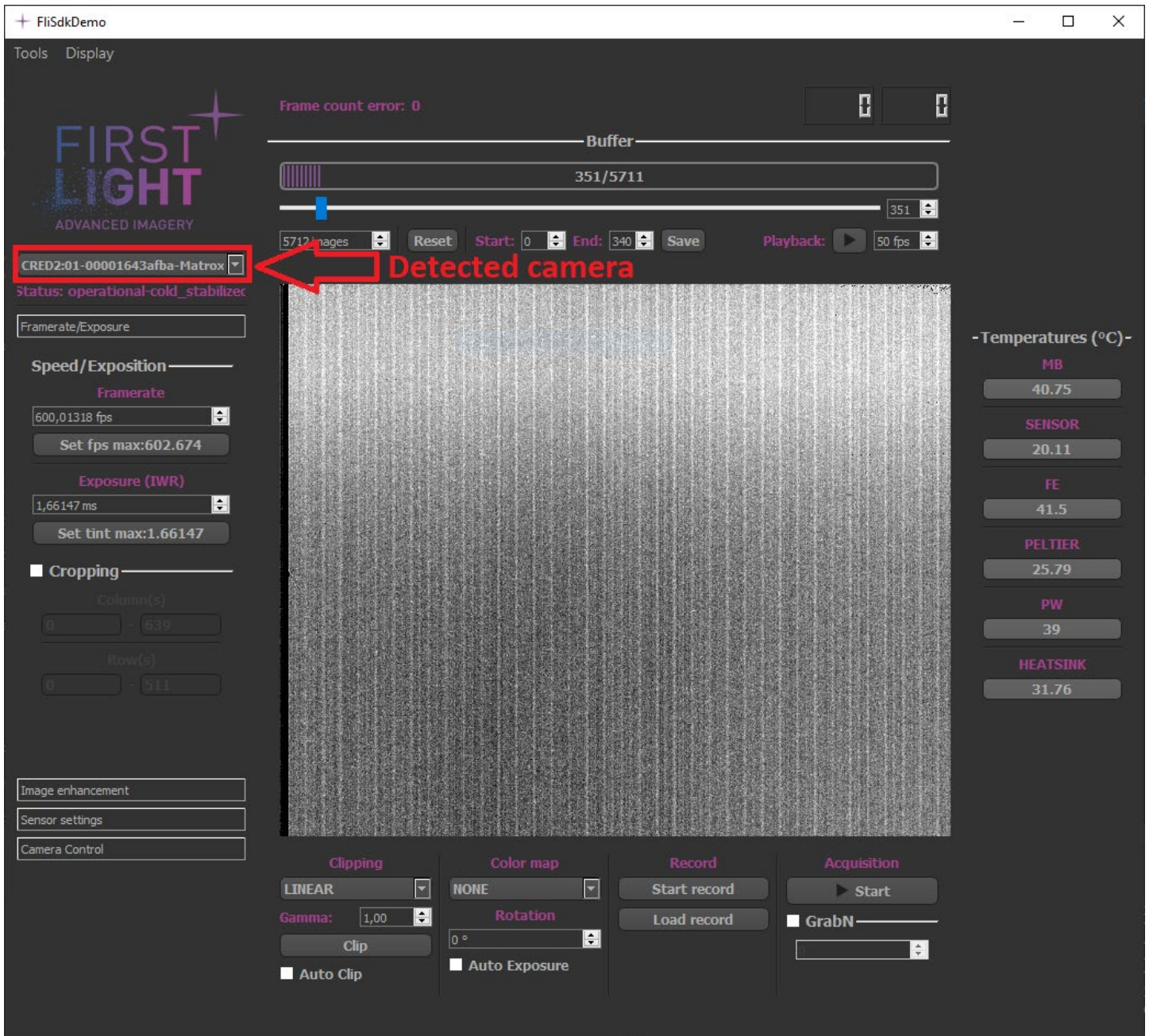


Fig. 5: Demo software

### 3.3. C++ API

Four example codes for the C++ API are available in "installDir/Example/API C++":

- An example to control one camera and grab the frames by polling the frame buffer.
- An example to control two cameras and grab the frames by polling the frame buffer.
- An example to control one camera and grab the frames by a callback.
- An example to control two cameras and grab the frames by a callback.

To resume, in order to use the SDK you have to:

1. Create an instance of the SDK
2. Detect the grabbers
3. Detect the cameras and keep the list
4. Set the camera to use, using the previous list
5. Configure the SDK (mode, ring buffer enable/disable, etc...)
6. Update the SDK
7. Use the right camera function

---

[Note: On windows, to compile with the C++ API visual studio >= 2015 is required.](#)

---

### 3.4. C API

Four example codes for the C API examples are available in "installDir/Example/API C":

- An example to control one camera and grab the frames by polling the frame buffer.
- An example to control two cameras and grab the frames by polling the frame buffer.
- An example to control one camera and grab the frames by a callback.
- An example to control two cameras and grab the frames by a callback.

The C API is a wrapper library based on C++ API

### 3.5. Python (for Windows and Linux)

In the install directory of the SDK you can find Python/lib/FLiSdk.py that you can import in your Python project in order to use it.

A code example is available in Python/demo, but you need to install some dependencies before starting it:

- PyQt5: pip install PyQt5
- Pillow: pip install Pillow
- Numpy: pip install numpy
- Matplotlib: pip install matplotlib

Then, start the demo with the command "python FLiSdkDemo.py".

An easier example is available for python in "installDir/Example/Python".

---

[Note: The SDK library is 64 bits only, so a 64 bits release of python is required.](#)

---

### 3.6. Save buffer formats

Different saving formats are available with the SDK (release > 1.2.0) to save the image buffer, the function is:

```
FliSdkError saveBuffer(std::string path, uint32_t start, uint32_t end, std::function<bool(int)> progressionCallback = nullptr, bool withMetadata = false, uint16_t offset = 0, bool forceUnsigned = false, uint16_t decimation = 1);
```

- "path" is the file path where the buffer will be saved.
- "start" is the starting index of the buffer.
- "end" is the end index of the buffer, so all the images between "start" and "end" will be saved
- "progressionCallback" is a callback to indicate the progression in the number of images.
- "withMetadata" to add some data of the camera.
- "offset" to add an offset to all the pixels before save.
- "forceUnsigned" to save pixels as unsigned.
- "decimation" to save each x images.

The different formats are available by calling the function "getAvailableSaveFormats" which returns the full name of the formats and the extension ("TIFF", ".tif").

The format is selected by the user thanks to the name of the file to save (path parameters).

Example: if the variable "path" ends with a ".tif", then the buffer will be saved in tiff format.

**".flf"** is a FirstLight format. It is a very simple proprietary format.

A fixed 2048 bytes header is added at the beginning of the file.

In this header, the camera configuration is written in ascii, so that it can be easily read.

The fields are separated by semicolons.

Offset 0	FirstLightFrame;Version:1.0;HeaderSize:xx;NbImages:xx;Camera Model:xx;Width:xx;Height:xx; CroppingEnabled:x;CroppingCol1:xx;CroppingCol2:xx; CroppingRow1:xx;CroppingRow2:xx;Cred1CroppingCols:xx;Cred 1CroppingRows:xx;
	Header padding
Offset 2048	Raw Image Data (16 bits Little Endian)

---

**Note:** Not all the fields are indicated in the array above. Since v1.4.0 all the formats have this header.

---

### 3.7. Frame grabbing explained

There are several ways to get the frames from the SDK.

First, you can use two different mechanisms: polling, or observer (improved callback). These two methods are detailed in the next sections.

Please note that when using polling `xxRawxxx` functions or observer you get raw frames which are read only. To get read write frames, polling processing function like `getImage16b()` must be used.

Second, you can get a frame whose memory has been allocated by the grabber library code or the `FliSdk` code.

However, in any case the registered observer is called from a `FliSdk` thread.

These various cases are explained below.

The SDK allocates a ring buffer from the RAM of the computer (by default, 20% of the available RAM). When a grabber receives a frame, the grabber library fires a callback with a pointer on a frame located in the memory allocated by the grabber library. The `FliSdk` provides an abstraction of the type of grabber, so that the grabber dependent callback is not seen by the user, and managed by the `FliSdk`.

The `FliSdk` is triggered by these dependent grabber callbacks, and calls the registered observers (`IRawImageReceivedObserver`).

The `FliSdk` copy the frame in its own ring buffer before or after calling the observers, depending on the flag provided to the function `addRawImageReceivedObserver`.

If the observer is registered to be called before the ring buffer, then it will be called with the pointer to the frame from the grabber buffer (less time to copy the frame but less latency). But if the observer is registered to be called after the copy in the ring buffer, then it will be called with a pointer to the frame from the `FliSdk` ring buffer (more time to copy/process image but more latency because of the copy in the ring buffer).

In the case of a critical latency application, the copy in the SDK buffer can be completely disabled with the function `enableRingBuffer`. In this case, the memory of the ring buffer will be released and no copy at all will occur.

#### 3.7.1. Poll the frames

The simplest way is to poll the frame from the SDK ring buffer using the function:

- `"const unsigned char* getRawImage(int64_t index=-1)"` which returns a pointer to the raw frame at position "index"
- `"uint8_t* getImage(int64_t index=-1, ProcessingId id = -1)"` which returns a 8bits/pixel processed image at position "index", this image is directly displayable.
- `"uint8_t* getImage16b(int64_t index = -1, ProcessingId id = -1)"` which returns a 16bits/pixel processed image at position "index".

If no index or `index = -1` then the last image of the buffer is returned.

#### 3.7.2. Observer

It is possible to get images by inheriting from the interface `IRawImageReceivedObserver`:

```

#include "FliSdk.h"

class Demo : public IRawImageReceivedObserver
{
    Demo();

    virtual void imageReceived(const uint8_t* image) override;
    virtual uint16_t fpsTrigger() override;

private:
    FliSdk* _fli;
}

//-----
void Demo::Demo()
{
    _fli = new FliSdk();
    //Do the init process...

    //Add this class in observer
    _fli->addRawImageReceivedObserver(this);
}

//-----
void Demo::imageReceived(const uint8_t* image)
{
    //here process the image
}

//-----
uint16_t Demo::fpsTrigger()
{
    return 100.0; //triggered at 100 fps
}

```

Each time an image is received, the SDK looks in a list of observers. For each observer, it reads the "fpsTrigger" value, to know at which speed it must call "imageReceived". If "fpsTrigger" returns 0, then "imageReceived" will be called at each image received.

The user can register an observer which will be called before or after the copy in the ringBuffer, using the beforeCopy flag. (cf. addRawImageReceivedObserver in doxygen API documentation for more details).

To get the pixel value, you need to cast the "uint8\_t\*" pointer to "uint16\_t\*" pointer or use the relevant pixel typedef. For example, "pixel\_cred2" for the C-RED 2 camera.

Cameras pixel types are available in the file "FliSdk\_utils.h".

### 3.8. LabView (only for Windows)

In the SDK install directory, you can find the file Labview/lib/FliSdk.lvlib. You can open it with LabView and take advantage of the functions provided to create a project.

An example project is available in Labview/Demo/FliSdkDemo\_Labview.vi. If you run it, click on "set camera" then "start", the video stream will be displayed.

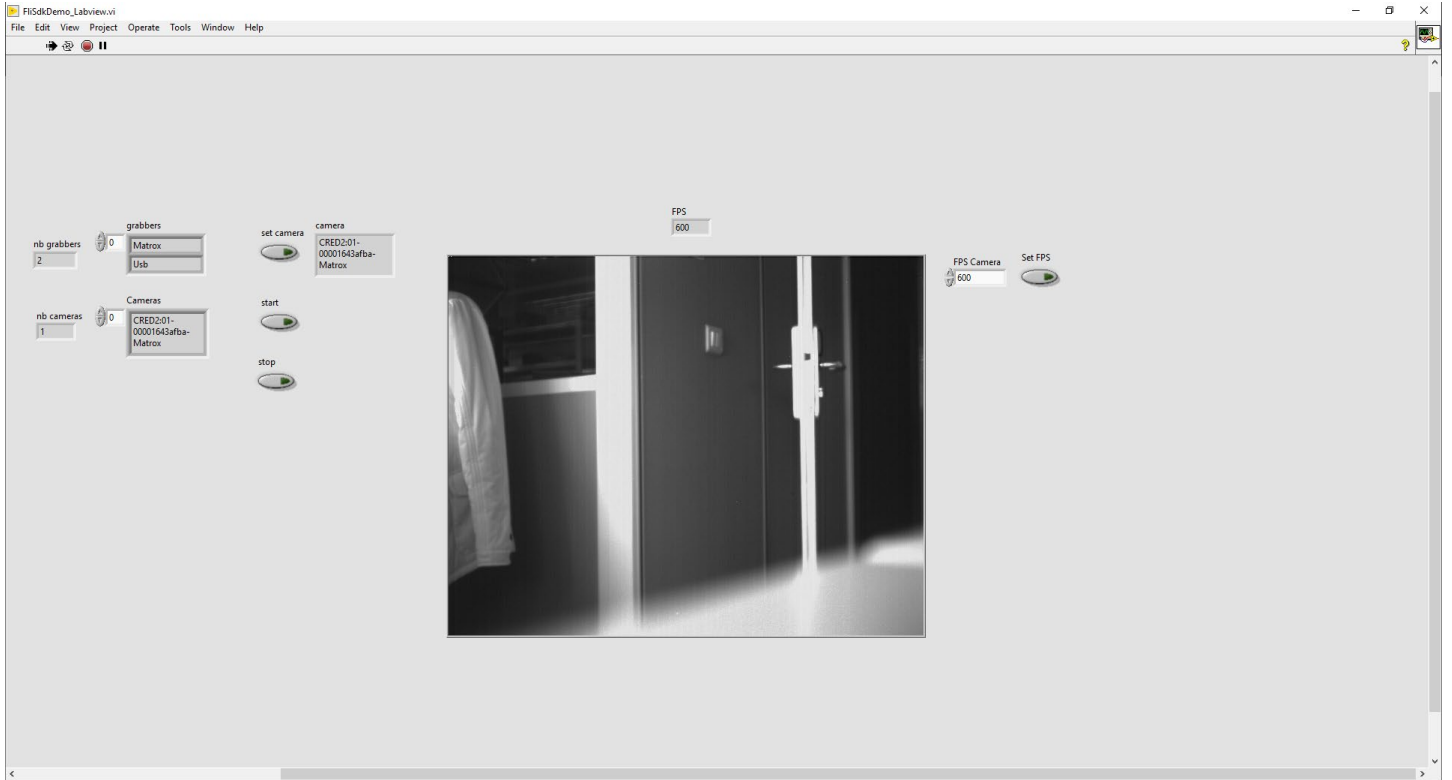


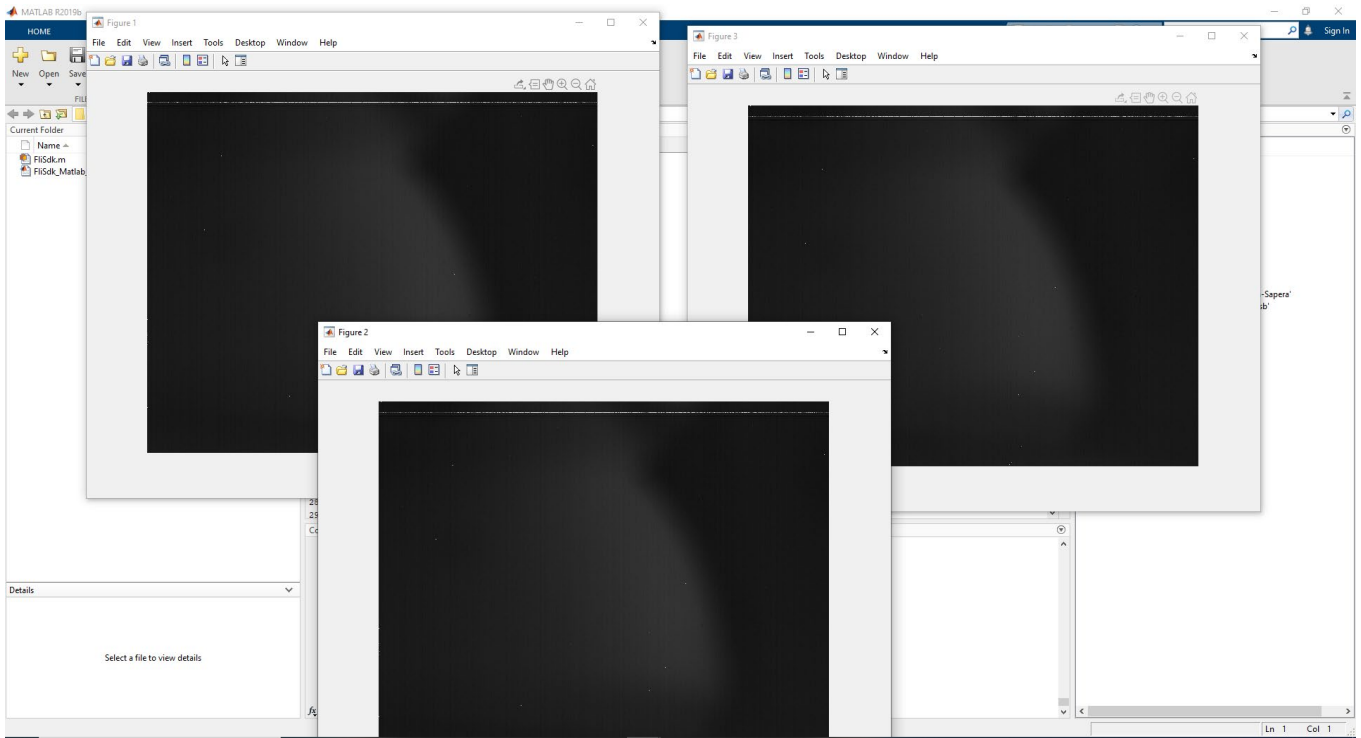
Fig. 6 : Demo of SDK with LabView

### 3.9. MATLAB (for Windows)

In the SDK installation directory, you can find a MATLAB folder containing the two following files:

- FliSdk.m which is a wrapper to use the C SDK with MATLAB.
- FliSdk\_Matlab\_Example.m which is an example of using the SDK.

If you open this example with MATLAB and start it, you will get the following display.



*Fig. 7: Figures generated by the MATLAB example script*

## 4. PLUGINS

### 4.1. Micro Manager (for Windows and Linux)

Micro Manager directly supports the First Light plugin, so it is directly available in the Micro Manager folder for recent releases. However, if you have issues with the native plugin, you can use the one distributed with the SDK. To do so, please copy the `MicroManager/mmgr_da1_FirstLightImaging.dll` in the install directory of MicroManager.

Start Micro Manager and go in "Devices" -> "Hardware Configuration Wizard":

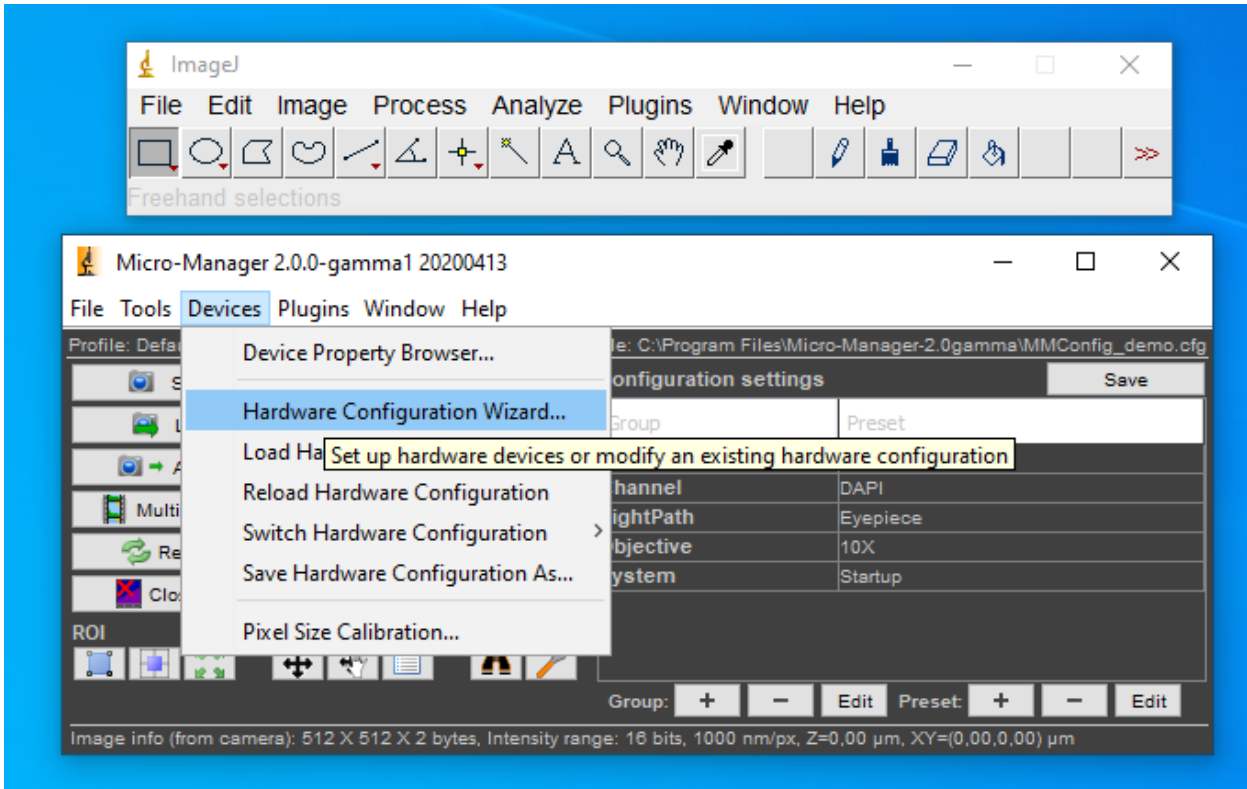


Fig. 8 : Micro Manager main page

Click "Next", then in the list in the bottom search for "FirstLightImaging" and double-click on "FliSdk: First Light Imaging camera".

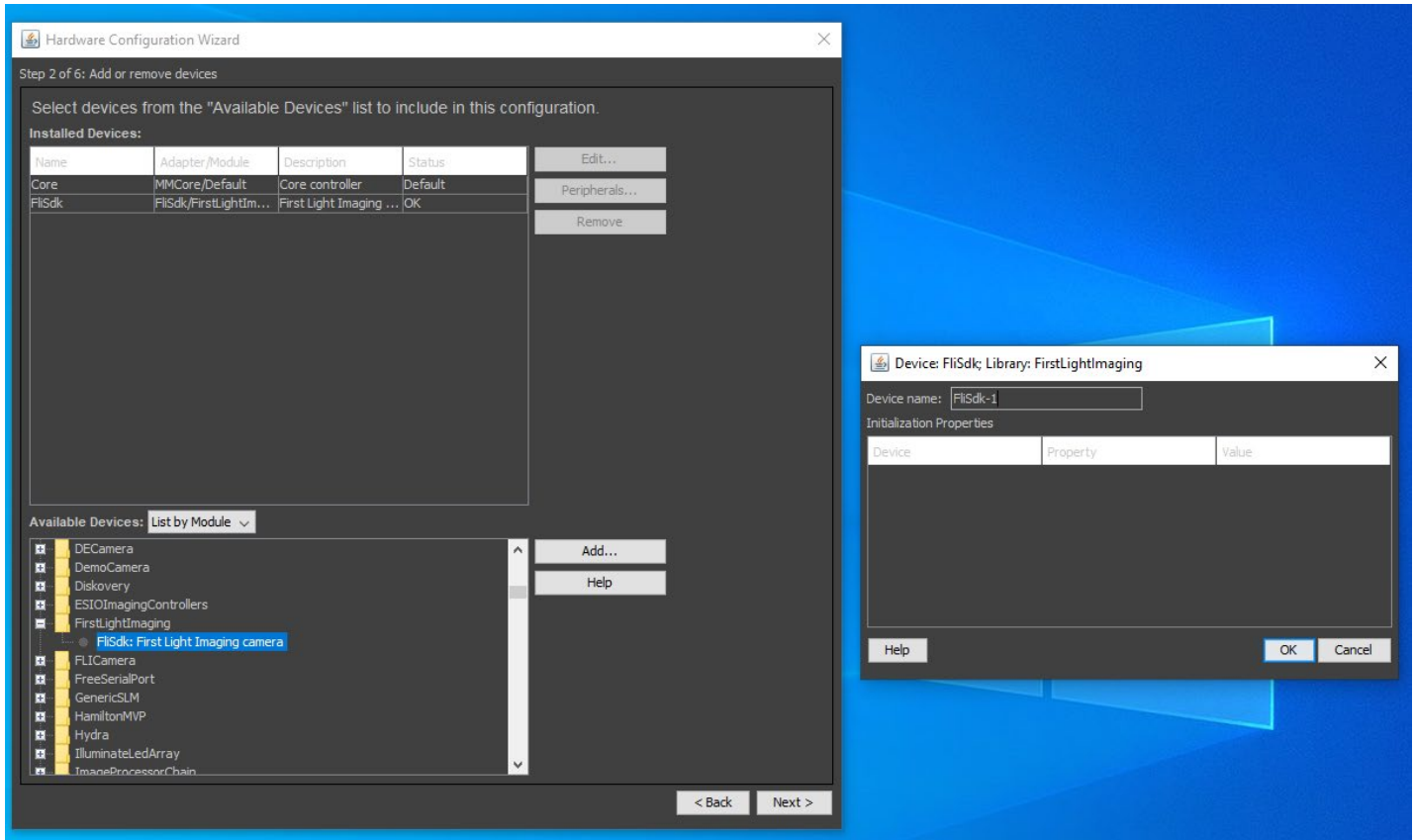


Fig. 9: Micro Manager configuration wizard

Click "Ok" in the window appeared and "Next" until the end of the wizard. Click on "Snap" or "Live" button to get images from camera.

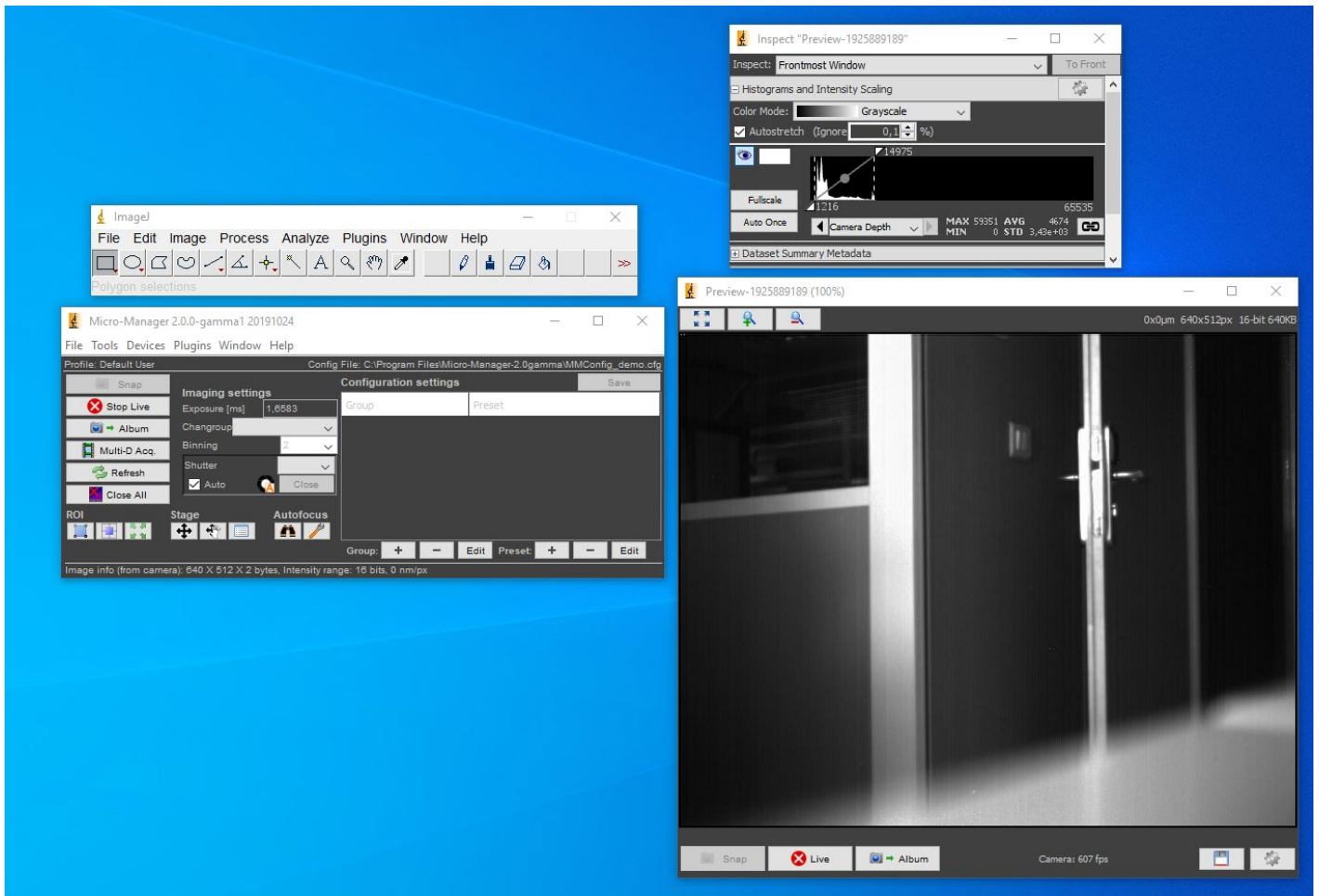


Fig. 10 : Micro Manager with image grabbed

You have access to the camera parameters by clicking on "Devices" -> "Device Property Browser".

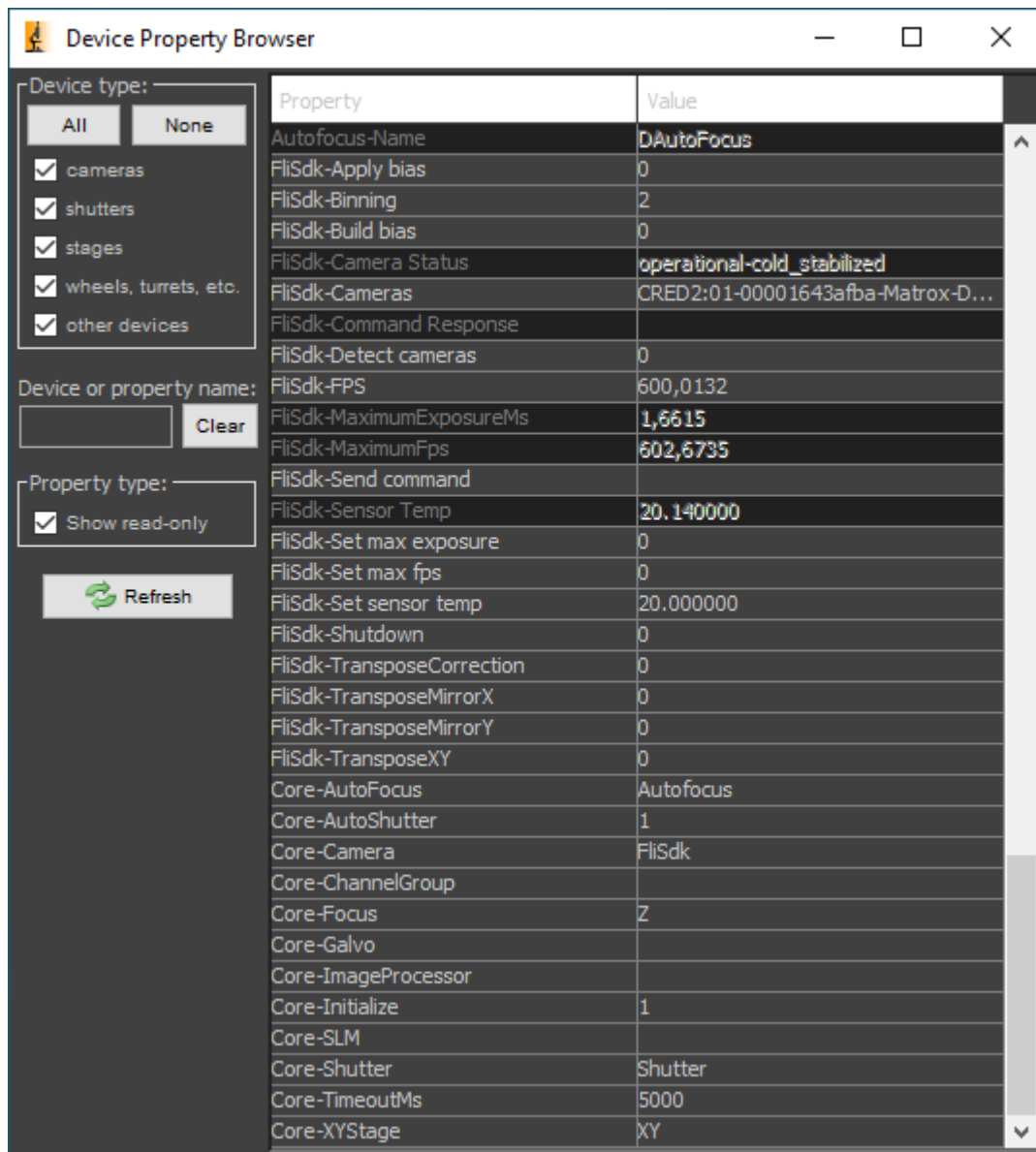


Fig. 11: Micro Manager camera config

The line "FliSdk-Send command" allows the user to send any commands to the camera.