Flc C++ library

v3.0.1

Table of contents

- Overview
- Versions
- Library files
- Flc class description
  - Flc class declaration
  - getVersion method
  - openLens method
  - initLens method
  - closeLens method
  - isLensOpen method
  - isLensConnected method
  - setParam method
  - getParam method
  - getParams method
  - executeCommand method
  - addVideoFrame method
  - decodeAndExecuteCommand method
  - encodeSetParamCommand method of Lens class
  - encodeCommand method of Lens class
  - decodeCommand method of Lens class
  - LensCommand enum
  - LensParam enum
- LensParams class description
  - LensParams class declaration
  - Serialize lens params
  - Deserialize lens params
  - Read and write lens params to JSON file
- Build and connect to your project

- Simple example

Overview

The **Flc** C++ library is a software controller for Fujinon lenses. These lenses provide control interface over serial port. The **Flc** library inherits Lens interface. It includes source code of libraries: Lens interface library (provides interface and data structures to control lenses, Apache 2.0 license), Logger logging library (provides function to print log information in console and files, Apache 2.0 license), SerialPort library (provides functions to work with serial ports, Apache 2.0 license) and **FujiProtocolParser** library (provides functions to encode control commands and decode responses from Fujinon lenses). The **Flc** library provides simple interface to be integrated in any C++ projects. The library repository (folder) provided by source code and doesn't have third-party dependencies to be specially installed in OS (excepts test application which depends on OpenCV](https://opencv.org/) open source library to provide user interface). It developed with C++17 standard and compatible with Linux and Windows.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	03.04.2023	First version.
2.0.0	10.01.2024	- Parser updated. - Documentation added.
3.0.1	02.02.2024	- Parser updated. - Documentation updated. - Code reviewed.

Library files

The library supplied by source code only. The user would be given a set of files in the form of a CMake project (repository). The repository structure is shown below:

```
CMakeLists.txt ----------------- Main CMake file of the library.
3rdparty ----------------------- Folder with third-party libraries.
    CMakeLists.txt ------------- CMake file which includes third-party libraries.
    FujiProtocolParser --------- Folder of the FujiProtocolParser library.
    Lens ----------------------- Folder of the Lens library.
    Logger --------------------- Folder of the Logger library.
    SerialPort ----------------- Folder of the SerialPort library.
test --------------------------- Folder with test application.
    3rdparty ------------------- Folder with third-party libraries.
        ConfigReader ----------- Folder of the ConfigReader library.
    CMakeLists.txt ------------- CMake file for test application.
    main.cpp ------------------- Source code file of test application.
```

```
src ---------------------------- Folder with source code of the library.
    CMakeLists.txt ------------- CMake file of the library.
    Flc.cpp -------------------- Source code file of the library.
    Flc.h ---------------------- Header file which includes Flc class declaration.
    FlcVersion.h --------------- Header file which includes version of the library.
    FlcVersion.h.in ------------ CMake service file to generate version file.
```

Additionally test application depends on OpenCV open source library to provide user interface.

Flc class description

Flc class declaration

Flc interface class declared in **Flc.h** file. The Flc class includes Lens interface. Class declaration:

```
class Flc : public cr::lens::Lens
{
public:

    /// Class constructor.
    Flc();

    /// Class destructor.
    ~Flc();

    /// Get class version.
    static std::string getVersion();

    /// Decode and execute command.
    bool decodeAndExecuteCommand(uint8_t* data, int size) override;

    /// Open controller serial port.
    bool openLens(std::string initString) override;

    /// Init lens by parameters structure.
    bool initLens(cr::lens::LensParams& params) override;

    /// Closes serial port and stops all threads.
    void closeLens() override;

    /// Get serial port status.
    bool isLensOpen() override;

    /// Get lens connection status.
    bool isLensConnected() override;

    /// Set lens parameter.
    bool setParam(cr::lens::LensParam id, float value) override;

    /// Get lens parameter.
    float getParam(cr::lens::LensParam id) override;
```

```
    /// Get the lens parameters.
    void getParams(cr::lens::LensParams& params) override;

    /// Execute lens command.
    bool executeCommand(cr::lens::LensCommand id, float arg = 0) override;

    /// Add video frame for auto focus purposes. Not supported.
    void addVideoFrame(cr::video::Frame& frame) override;
};
```

getVersion method

getVersion() returns string of **Flc** class version. Method declaration:

```
static std::string getVersion();
```

Method can be used without **Flc** class instance:

```
cout << "Flc class version: " << Flc::getVersion() << endl;
```

Console output:

```
Flc class version: 3.0.1
```

openLens method

openLens(...) opens serial port to communicate with Fujinon lens and run communication thread. If serial port already open the method will return TRUE. Lens parameters will be initialized by default. Method declaration:

```
bool openLens(std::string initString) override;
```

Parameter	Value
initString	Initialization string. Format : [serial port name];[baudrate];[timeout, msec]. For example: "/dev/ttyS0;9600;50"

Returns: TRUE if the controller initialized or FALSE if not.

initLens method

initLens(...) initializes controller and sets lens params (Lens interface). The method will set given lens params and after will call openLens(...) method. After successful initialization the library will run communication threads (thread to communicate with lens via serial port) if it not run before. Method declaration:

```
bool initLens(cr::lens::LensParams& params) override;
```

Parameter	Value
params	LensParams class object. LensParams class includes initString which used in openLens(...) method.

Returns: TRUE if the controller initialized and lens parameters were set or FALSE if not.

closeLens method

closeLens() closes serial port and stops all communication threads. Method declaration:

```
void closeLens() override;
```

isLensOpen method

isLensOpen() method returns controller initialization status. Open status shows if the controller initialized (serial port open) but doesn't show if controller has communication with equipment. For example, if serial port is open (opens serial port file in OS) but equipment can be not active (no power). In this case open status just shows that the serial port is open. Method declaration:

```
bool isLensOpen() override;
```

Returns: TRUE is the controller initialized (serial port open) or FALSE if not.

isLensConnected method

isLensConnected() shows if the controller receives responses from lens. For example, if serial port open but equipment not active (no power). In this case method isLensOpen(...) will return TRUE but **isLensConnected()** method will return FALSE. Method declaration:

```
bool isLensConnected() override;
```

Returns: TRUE if the controller has data exchange with equipment or FALSE if not.

setParam method

setParam(...) intended to set Lens parameter value. Method declaration:

```
bool setParam(LensParam id, float value) override;
```

Parameter	Description
id	Lens parameter ID according to LensParam enum.
value	Lens parameter value. Valid values depend on parameter ID.

Returns: TRUE if the parameter is set or FALSE if not.

getParam method

getParam(...) intended to obtain Lens parameter value. Method declaration:

```
float getParam(LensParam id) override;
```

Parameter	Description
id	Lens parameter ID according to LensParam enum.

Returns: parameter value or -1 of the parameters not supported.

getParams method

getParams(...) intended to obtain Lens parameters structure. Method declaration:

```
void getParams(LensParams& params) override;
```

Parameter	Description
params	Reference to LensParams class object.

executeCommand method

executeCommand(...) intended to execute Lens action command. Method declaration:

```
bool executeCommand(LensCommand id, float arg = 0) override;
```

Parameter	Description
id	Lens command ID according to LensCommand enum.
arg	Lens command argument. Valid values depend on command ID.

Returns: TRUE is the command is executed (accepted by controller) or FALSE if not.

addVideoFrame method

addVideoFrame(...) copies video frame data to lens controller to perform autofocus algorithm. **It is not supported** in **FIc** library. Method declaration:

```
void addVideoFrame(cr::video::Frame& frame) override;
```

Parameter	Description
frame	Frame class object.

decodeAndExecuteCommand method

decodeAndExecuteCommand(...) method decodes and executes command on controller side. Method will decode commands which encoded by encodeCommand(...) and encodeSetParamCommand(...) methods of Lens interface class. If command decoded the method will call setParam(...) or executeCommand(...) methods for lens. This method is thread-safe. This means that the method can be safely called from any thread. Method declaration:

```
bool decodeAndExecuteCommand(uint8_t* data, int size) override;
```

Parameter	Description
data	Pointer to input command.
size	Size of command. Must be 11 bytes for SET_PARAM and COMMAND.

Returns: TRUE if command decoded (SET_PARAM or COMMAND) and executed (action command or set param command).

encodeSetParamCommand method of Lens class

encodeSetParamCommand(...) static method of Lens interface class designed to encode command to change any remote lens parameter. To control a lens remotely, the developer has to develop his own protocol and according to it encode the command and deliver it over the communication channel. To simplify this, the **Lens** class contains static methods for encoding the control command. The **Lens** class provides two types of commands: a parameter change command (SET_PARAM) and an action command (COMMAND). **encodeSetParamCommand(...)** designed to encode SET_PARAM command. Method declaration:

```
static void encodeSetParamCommand(uint8_t* data, int& size, LensParam id, float value);
```

Parameter	Description
data	Pointer to data buffer for encoded command. Must have size >= 11.
size	Size of encoded data. Will be 11 bytes.

Parameter	Description
id	Parameter ID according to LensParam enum.
value	Parameter value.

SET_PARAM command format:

Byte	Value	Description
0	0x01	SET_PARAM command header value.
1	Major	Major version of Lens class.
2	Minor	Minor version of Lens class.
3	id	Parameter ID **int32_t** in Little-endian format.
4	id	Parameter ID **int32_t** in Little-endian format.
5	id	Parameter ID **int32_t** in Little-endian format.
6	id	Parameter ID **int32_t** in Little-endian format.
7	value	Parameter value **float** in Little-endian format.
8	value	Parameter value **float** in Little-endian format.
9	value	Parameter value **float** in Little-endian format.
10	value	Parameter value **float** in Little-endian format.

encodeSetParamCommand(...) is static and used without Lens interface class. This method used on client side (control system). Example:

```
// Buffer for encoded data.
uint8_t data[11];
// Size of encoded data.
int size = 0;
// Random parameter value.
float outValue = (float)(rand() % 20);
// Encode command.
Lens::encodeSetParamCommand(data, size, LensParam::AF_ROI_X0, outValue);
```

encodeCommand method of Lens class

encodeCommand(...) static method of Lens interface class designed to encode lens action command. To control a lens remotely, the developer has to develop his own protocol and according to it encode the command and deliver it over the communication channel. To simplify this, the **Lens** class contains static methods for encoding the control command. The **Lens** class provides two types of commands: a parameter change command (SET_PARAM) and an action command (COMMAND). **encodeCommand(...)** designed to encode COMMAND command (action command). Method declaration:

```
static void encodeCommand(uint8_t* data, int& size, LensCommand id, float arg = 0.0f);
```

Parameter	Description
data	Pointer to data buffer for encoded command. Must have size >= 11.
size	Size of encoded data. Will be 11 bytes.
id	Command ID according to **LensCommand enum**.
arg	Command argument value (value depends on command ID).

COMMAND format:

Byte	Value	Description
0	0x00	SET_PARAM command header value.
1	Major	Major version of Lens class.
2	Minor	Minor version of Lens class.
3	id	Command ID **int32_t** in Little-endian format.
4	id	Command ID **int32_t** in Little-endian format.
5	id	Command ID **int32_t** in Little-endian format.
6	id	Command ID **int32_t** in Little-endian format.
7	arg	Command argument value **float** in Little-endian format.
8	arg	Command argument value **float** in Little-endian format.
9	arg	Command argument value **float** in Little-endian format.
10	arg	Command argument value **float** in Little-endian format.

encodeCommand(...) is static and used without **Lens** class instance. This method used on client side (control system). Encoding example:

```
// Buffer for encoded data.
uint8_t data[11];
// Size of encoded data.
int size = 0;
// Random command argument value.
float outValue = (float)(rand() % 20);
// Encode command.
Lens::encodeCommand(data, size, LensCommand::ZOOM_TO_POS, outValue);
```

decodeCommand method of Lens class

decodeCommand(...) static method of Lens interface class designed to decode command on lens controller side. o control a lens remotely, the developer has to develop his own protocol and according to it decode the command on lens controller side. To simplify this, the **Lens** interface class contains static method to decode input command (commands should be encoded by methods **encodeSetParamsCommand(...)** or **encodeCommand(...)**). The **Lens** class provides two types of commands: a parameter change command (SET_PARAM) and an action command (COMMAND). Method declaration:

```
static int decodeCommand(uint8_t* data, int size, LensParam& paramId, LensCommand&
commandId, float& value);
```

Parameter	Description
data	Pointer to input command.
size	Size of command. Should be 11 bytes.
paramId	Lens parameter ID according to LensParam enum. After decoding SET_PARAM command the method will return parameter ID.
commandId	Lens command ID according to LensCommand enum. After decoding COMMAND the method will return command ID.
value	Lens parameter value (after decoding SET_PARAM command) or lens command argument (after decoding COMMAND).

Returns: 0 - in case decoding COMMAND, **1** - in case decoding SET_PARAM command or **-1** in case errors.

Data structures

LensCommand enum

Enum declaration:

```
enum class LensCommand
{
    /// Move zoom tele (in).
    ZOOM_TELE = 1,
    /// Move zoom wide (out).
    ZOOM_WIDE,
    /// Move zoom to position.
    ZOOM_TO_POS,
    /// Stop zoom moving including stop zoom to position command.
    ZOOM_STOP,
    /// Move focus far.
    FOCUS_FAR,
    /// Move focus near.
    FOCUS_NEAR,
    /// Move focus to position.
```

```
    FOCUS_TO_POS,
    /// Stop focus moving including stop focus to position command.
    FOCUS_STOP,
    /// Move iris open.
    IRIS_OPEN,
    /// Move iris close.
    IRIS_CLOSE,
    /// Move iris to position.
    IRIS_TO_POS,
    /// Stop iris moving including stop iris to position command.
    IRIS_STOP,
    /// Start autofocus.
    AF_START,
    /// Stop autofocus.
    AF_STOP,
    /// Restart lens controller.
    RESTART,
    /// Detect zoom and focus hardware ranges.
    DETECT_HW_RANGES
};
```

Table 2 - Lens commands description.

Command	Description
ZOOM_TELE	Move zoom tele (in). Command doesn't have arguments. User can set zoom movement speed via lens parameters.
ZOOM_WIDE	Move zoom wide (out). Command doesn't have arguments. User can set zoom movement speed via lens parameters.
ZOOM_TO_POS	Move zoom to position. Controller has zoom range from 0 (full wide) to 65535 (full tele) regardless of the hardware value of the zoom position. User can set zoom movement speed via lens parameters.
ZOOM_STOP	Stop zoom moving including stop zoom to position command.
FOCUS_FAR	Move focus far. Command doesn't have arguments. User can set focus movement speed via lens parameters.
FOCUS_NEAR	Move focus near. Command doesn't have arguments. User can set focus movement speed via lens parameters.
FOCUS_TO_POS	Move focus to position. Controller has focus range from 0 (full near) to 65535 (full far) regardless of the hardware value of the focus position. User can set focus movement speed via lens parameters.
FOCUS_STOP	Stop focus moving including stop focus to position command.
IRIS_OPEN	Move iris open. Command doesn't have arguments. User can set iris movement speed via lens parameters.
IRIS_CLOSE	Move iris close. Command doesn't have arguments. User can set iris movement speed via lens parameters.

Command	Description
IRIS_TO_POS	Move iris to position. Lens controller should have iris range from 0 (full close) to 65535 (full far) regardless of the hardware value of the iris position. If the minimum and maximum iris position limits are set by the user in the lens parameters, the range of the hardware iris position must be scaled to the 0-65535 user space range. Command argument: iris position 0-65535. User should be able to set iris movement speed via lens parameters.
IRIS_STOP	Stop iris moving including stop iris to position command. Command doesn't have arguments.
AF_START	Start autofocus. Command doesn't have arguments.
AF_STOP	Stop autofocus. Command doesn't have arguments.
RESTART	Restart lens controller. **Not supported** by **Flc** library.
DETECT_HW_RANGES	Detect hardware ranges. **Not supported** by **Flc** library.

LensParam enum

Enum declaration:

```
enum class LensParam
{
    /// Zoom position.
    ZOOM_POS = 1,
    /// Hardware zoom position.
    ZOOM_HW_POS,
    /// Focus position.
    FOCUS_POS,
    /// Hardware focus position.
    FOCUS_HW_POS,
    /// Iris position.
    IRIS_POS,
    /// Hardware iris position.
    IRIS_HW_POS,
    /// Focus mode.
    FOCUS_MODE,
    /// Filter mode.
    FILTER_MODE,
    /// Autofocus ROI top-left corner horizontal position in pixels.
    AF_ROI_X0,
    /// Autofocus ROI top-left corner vertical position in pixels.
    AF_ROI_Y0,
    /// Autofocus ROI bottom-right corner horizontal position in pixels.
    AF_ROI_X1,
    /// Autofocus ROI bottom-right corner vertical position in pixels.
    AF_ROI_Y1,
    /// Zoom speed.
    ZOOM_SPEED,
    /// Zoom hardware speed.
```

```
    ZOOM_HW_SPEED,
    /// Maximum zoom hardware speed.
    ZOOM_HW_MAX_SPEED,
    /// Focus speed.
    FOCUS_SPEED,
    /// Focus hardware speed.
    FOCUS_HW_SPEED,
    /// Maximum focus hardware speed.
    FOCUS_HW_MAX_SPEED,
    /// Iris speed.
    IRIS_SPEED,
    /// Iris hardware speed.
    IRIS_HW_SPEED,
    /// Maximum iris hardware speed.
    IRIS_HW_MAX_SPEED,
    /// Zoom hardware tele limit.
    ZOOM_HW_TELE_LIMIT,
    /// Zoom hardware wide limit.
    ZOOM_HW_WIDE_LIMIT,
    /// Focus hardware far limit.
    FOCUS_HW_FAR_LIMIT,
    /// Focus hardware near limit.
    FOCUS_HW_NEAR_LIMIT,
    /// Iris hardware open limit.
    IRIS_HW_OPEN_LIMIT,
    /// Iris hardware close limit.
    IRIS_HW_CLOSE_LIMIT,
    /// Focus factor if it was calculated.
    FOCUS_FACTOR,
    /// Lens connection status.
    IS_CONNECTED,
    /// Focus hardware speed in autofocus mode.
    FOCUS_HW_AF_SPEED,
    /// Threshold for changes of focus factor to start refocus.
    FOCUS_FACTOR_THRESHOLD,
    /// Timeout for automatic refocus in seconds.
    REFOCUS_TIMEOUT_SEC,
    /// Flag about active autofocus algorithm.
    AF_IS_ACTIVE,
    /// Iris mode.
    IRIS_MODE,
    /// ROI width (pixels).
    AUTO_AF_ROI_WIDTH,
    /// ROI height (pixels).
    AUTO_AF_ROI_HEIGHT,
    /// Video frame border size (along vertical and horizontal axes).
    AUTO_AF_ROI_BORDER,
    /// AF ROI mode (write/read). Value: 0 - Manual position, 1 - Auto position.
    AF_ROI_MODE,
    /// Lens extender mode.
    EXTENDER_MODE,
    /// Lens stabilization mode.
    STABILIZER_MODE,
    /// Autofocus range.
    AF_RANGE,
    /// Current horizontal Field of view, degree.
```

```
    X_FOV_DEG,
    /// Current vertical Field of view, degree.
    Y_FOV_DEG,
    /// Logging mode.
    LOG_MODE,
    /// Lens temperature, degree.
    TEMPERATURE,
    /// Lens controller initialization status.
    IS_OPEN,
    /// Lens type.
    TYPE,
    /// Lens custom parameter.
    CUSTOM_1,
    /// Lens custom parameter.
    CUSTOM_2,
    /// Lens custom parameter.
    CUSTOM_3
};
```

Table 4 - Lens params description.

Parameter	Access	Description
ZOOM_POS	read / write	Zoom position. Setting a parameter is equivalent to the command ZOOM_TO_POS. Controller has zoom range from 0 (full wide) to 65535 (full tele). User can set zoom movement speed via lens parameters.
ZOOM_HW_POS	read / write	Hardware zoom position. Parameter has same value as ZOOM_POS parameter.
FOCUS_POS	read / write	Focus position. Setting a parameter is equivalent to the command FOCUS_TO_POS. Lens controller has focus range from 0 (full near) to 65535 (full far). User can set focus movement speed via lens parameters.
FOCUS_HW_POS	read / write	Hardware focus position. Parameter has same value as FOCUS_POS parameter.
IRIS_POS	read / write	Iris position. Setting a parameter is equivalent to the command IRIS_TO_POS. Lens controller has iris range from 0 (full close) to 65535 (full open). If the minimum and maximum iris position limits are set by the user in the lens parameters, User can set iris movement speed via lens parameters.
IRIS_HW_POS	read / write	Hardware iris position. Parameter value depends on particular lens controller.
FOCUS_MODE	read / write	Focus mode. Value : **0** - Manual **1** - AF ON **2** - Quick AF ON

Parameter	Access	Description
FILTER_MODE	read / write	Filter mode. Value: **0** - no effect **1** - n-IR 850 nm **2** - n-IR 870 nm **3** - n-IR 950 nm **4** - no effect **5** - n-IR 850 nm **6** - n-IR 870 nm **7** - n-IR 950 nm **8** - no effect **9** - n-IR 850 nm **10** - n-IR 870 nm **11** - n-IR 950 nm **12** - no effect **13** - n-IR 850 n **14** - n-IR 870 nm **15** - n-IR 950 nm
AF_ROI_X0	read / write	**Not supported** by Flc library.
AF_ROI_Y0	read / write	**Not supported** by Flc library.
AF_ROI_X1	read / write	**Not supported** by Flc library.
AF_ROI_Y1	read / write	**Not supported** by Flc library.
ZOOM_SPEED	read / write	Zoom speed. Controller has zoom speed range from 0 to 100%.
ZOOM_HW_SPEED	read / write	**Not supported** by Flc library.
ZOOM_HW_MAX_SPEED	read / write	**Not supported** by Flc library.
FOCUS_SPEED	read / write	Focus speed. Controller has focus speed range from 0 to 100%.
FOCUS_HW_SPEED	read / write	**Not supported** by Flc library.
FOCUS_HW_MAX_SPEED	read / write	**Not supported** by Flc library.
IRIS_SPEED	read / write	**Not supported** by Flc library.

Parameter	Access	Description
IRIS_HW_SPEED	read / write	**Not supported** by Flc library.
IRIS_HW_MAX_SPEED	read / write	**Not supported** by Flc library.
ZOOM_HW_TELE_LIMIT	read / write	Zoom hardware tele limit. Value has range from 0 to 65535. Lens will stop zoom moving if hardware zoom position will be our of limits.
ZOOM_HW_WIDE_LIMIT	read / write	Zoom hardware wide limit. Value has range from 0 to 65535. Lens controller will stop zoom moving if hardware zoom position will be our of limits.
FOCUS_HW_FAR_LIMIT	read / write	Focus hardware far limit. Value has range from 0 to 65535. Lens controller will stop focus if hardware focus will be our of limits.
FOCUS_HW_NEAR_LIMIT	read / write	Focus hardware near limit. Value has range from 0 to 65535. Lens controller should stop focus if hardware focus will be our of limits.
IRIS_HW_OPEN_LIMIT	read / write	Iris hardware open limit. Value has range from 0 to 65535. Lens controller should control iris position. Lens controller will stop iris moving if hardware iris position will be our of limits.
IRIS_HW_CLOSE_LIMIT	read / write	Iris hardware close limit. Value has range from 0 to 65535. Lens controller should control iris position. Lens controller will stop iris moving if hardware iris position will be our of limits.
FOCUS_FACTOR	read only	**Not supported** by Flc library.
IS_CONNECTED	read only	Connection status. Connection status shows if the controller has data exchange with equipment. For example, if serial port open but equipment can be not active (no power). In this case connection status shows that controller doesn't have data exchange with equipment (method will return 0). It the controller has data exchange with equipment the method will return 1. If the controller not initialize the connection status always FALSE. Values: **0** - not connected. **1** - connected.
FOCUS_HW_AF_SPEED	read / write	**Not supported** by Flc library.
FOCUS_FACTOR_THRESHOLD	read / write	**Not supported** by Flc library.

Parameter	Access	Description
REFOCUS_TIMEOUT_SEC	read / write	**Not supported** by Flc library.
AF_IS_ACTIVE	read only	**Not supported** by Flc library.
IRIS_MODE	read / write	Iris mode. Values: **0** - auto **1** - manual
AUTO_AF_ROI_WIDTH	read / write	**Not supported** by Flc library.
AUTO_AF_ROI_HEIGHT	read / write	**Not supported** by Flc library.
AUTO_AF_ROI_BORDER	read / write	**Not supported** by Flc library.
AF_ROI_MODE	read / write	**Not supported** by Flc library.
EXTENDER_MODE	read / write	Extender mode. Values: **0** - x1.0 **1** - x2.0
STABILIZER_MODE	read / write	Extender mode. Values: **0** - off **1** - on.
AF_RANGE	read / write	Autofocus range. Values: **0** - full range search. **1** - 1/2 range search. **2** - 1/4 range search. **3** - 1/8 range search. **4** - 1/16 range search. **5** - 1/32 range search. **6** - 1/64 range search. **7-15** - full range search.
X_FOV_DEG	read only	**Not supported** by Flc library.
Y_FOV_DEG	read only	**Not supported** by Flc library.
LOG_MODE	read / write	Logging mode. Values: **0** - Disable, **1** - Only file, **2** - Only terminal (console), **3** - File and terminal.

Parameter	Access	Description
TEMPERATURE	read only	**Not supported** by Flc library.
IS_OPEN	read only	Controller initialization status. Open status shows if the controller initialized or not but doesn't show if controller has communication with equipment. For example, if serial port open but equipment can be not active (no power). In this case open status just shows that the controller has opened serial port. Values: 0 - not open (not initialized), 1 - open (initialized).
TYPE	read / write	**Not supported** by Flc library.
CUSTOM_1	read / write	**Not supported** by Flc library.
CUSTOM_2	read / write	**Not supported** by Flc library.
CUSTOM_3	read / write	**Not supported** by Flc library.

LensParams class description

LensParams class used for lens initialization or to get all actual controller params. Also **LensParams** provides structure to write/read params from JSON files (**JSON_READABLE** macro) and provides method to encode and decode params.

LensParams class declaration

LensParams interface class declared in **Lens.h** file. Class declaration:

```
class LensParams
{
public:
    /// Initialization string.
    std::string initString{"/dev/ttyUSB0;9600;7"};
    /// Zoom position.
    int zoomPos{0};
    /// Hardware zoom position.
    int zoomHwPos{0};
    /// Focus position.
    int focusPos{0};
    /// Hardware focus position.
    int focusHwPos{0};
    /// Iris position.
    int irisPos{0};
```

```
    /// Hardware iris position.
    int irisHwPos{0};
    /// Focus mode.
    int focusMode{0};
    /// Filter mode.
    int filterMode{0};
    /// Autofocus ROI top-left corner horizontal position in pixels.
    int afRoiX0{0};
    /// Autofocus ROI top-left corner vertical position in pixels.
    int afRoiY0{0};
    /// Autofocus ROI bottom-right corner horizontal position in pixels.
    int afRoiX1{0};
    /// Autofocus ROI bottom-right corner vertical position in pixels.
    int afRoiY1{0};
    /// Zoom speed.
    int zoomSpeed{50};
    /// Zoom hardware speed.
    int zoomHwSpeed{50};
    /// Maximum zoom hardware speed.
    int zoomHwMaxSpeed{50};
    /// Focus speed.
    int focusSpeed{50};
    /// Focus hardware speed.
    int focusHwSpeed{50};
    /// Maximum focus hardware speed.
    int focusHwMaxSpeed{50};
    /// Iris speed.
    int irisSpeed{50};
    /// Iris hardware speed.
    int irisHwSpeed{50};
    /// Maximum iris hardware speed.
    int irisHwMaxSpeed{50};
    /// Zoom hardware tele limit.
    int zoomHwTeleLimit{65535};
    /// Zoom hardware wide limit.
    int zoomHwWideLimit{0};
    /// Focus hardware far limit.
    int focusHwFarLimit{65535};
    /// Focus hardware near limit.
    int focusHwNearLimit{0};
    /// Iris hardware open limit.
    int irisHwOpenLimit{65535};
    /// Iris hardware close limit.
    int irisHwCloseLimit{0};
    /// Focus factor if it was calculated.
    float focusFactor{0.0f};
    /// Lens connection status.
    bool isConnected{false};
    /// Focus hardware speed in autofocus mode.
    int afHwSpeed{50};
    /// Timeout for automatic refocus in seconds.
    float focusFactorThreshold{0.0f};
    /// Timeout for automatic refocus in seconds.
    int refocusTimeoutSec{0};
    /// Flag about active autofocus algorithm.
    bool afIsActive{false};
```

```
    /// Iris mode.
    int irisMode{0};
    /// ROI width (pixels) for autofocus algorithm.
    int autoAfRoiWidth{150};
    /// ROI height (pixels) for autofocus algorithm.
    int autoAfRoiHeight{150};
    /// Video frame border size (along vertical and horizontal axes).
    int autoAfRoiBorder{100};
    /// AF ROI mode (write/read).
    int afRoiMode{0};
    /// Lens extender mode.
    int extenderMode{0};
    /// Lens stabilization mode.
    int stabiliserMode{0};
    /// Autofocus range.
    int afRange{0};
    /// Current horizontal Field of view, degree.
    float xFovDeg{1.0f};
    /// Current vertical Field of view, degree.
    float yFovDeg{1.0f};
    /// Logging mode.
    int logMode{0};
    /// Lens temperature, degree (read only).
    float temperature{0.0f};
    /// Lens controller initialization status.
    bool isOpen{false};
    /// Lens type. Value depends on implementation.
    int type{0};
    /// Lens custom parameter.
    float custom1{0.0f};
    /// Lens custom parameter.
    float custom2{0.0f};
    /// Lens custom parameter.
    float custom3{0.0f};
    /// List of points to calculate fiend of view.
    std::vector<FovPoint> fovPoints{std::vector<FovPoint>()};

    JSON_READABLE(LensParams, initString, focusMode, filterMode,
                  afRoiX0, afRoiY0, afRoiX1, afRoiY1, zoomHwMaxSpeed,
                  focusHwMaxSpeed, irisHwMaxSpeed, zoomHwTeleLimit,
                  zoomHwWideLimit, focusHwFarLimit, focusHwNearLimit,
                  irisHwOpenLimit, irisHwCloseLimit, afHwSpeed,
                  focusFactorThreshold, refocusTimeoutSec, irisMode,
                  autoAfRoiWidth, autoAfRoiHeight, autoAfRoiBorder,
                  afRoiMode, extenderMode, stabiliserMode, afRange,
                  logMode, type, custom1, custom2, custom3, fovPoints);

    /// operator =
    LensParams& operator= (const LensParams& src);

    /// Encode params.
    bool encode(uint8_t* data, int bufferSize, int& size,
                LensParamsMask* mask = nullptr);

    /// Decode params.
    bool decode(uint8_t* data, int dataSize);
```

```
};
```

Note: LensParams class fields description is equivalent to LensParam enum description except initString. initString form of initialization string if openLens(...) method. Also LensParams class include list of Field of View point (**fovPoints**) which not supported by **Flc** library.

Serialize lens params

LensParams class provides method **encode(...)** to serialize lens params. Serialization of lens params necessary in case when you need to send lens params via communication channels. Method doesn't encode **initString** string field and **fovPoints**. Method provides options to exclude particular parameters from serialization. To do this method inserts binary mask (7 bytes) where each bit represents particular parameter and **decode(...)** method recognizes it. Method declaration:

```
bool encode(uint8_t* data, int bufferSize, int& size, LensParamsMask* mask = nullptr);
```

Parameter	Value
data	Pointer to data buffer.
size	Size of encoded data.
bufferSize	Data buffer size. Buffer size must be >= 201 bytes.
mask	Parameters mask - pointer to **LensParamsMask** structure. **LensParamsMask** (declared in Lens.h file) determines flags for each field (parameter) declared in **LensParams class**. If the user wants to exclude any parameters from serialization, he can put a pointer to the mask. If the user wants to exclude a particular parameter from serialization, he should set the corresponding flag in the **LensParamsMask** structure.

LensParamsMask structure declaration:

```
typedef struct LensParamsMask
{
    bool zoomPos{true};
    bool zoomHwPos{true};
    bool focusPos{true};
    bool focusHwPos{true};
    bool irisPos{true};
    bool irisHwPos{true};
    bool focusMode{true};
    bool filterMode{true};
    bool afRoiX0{true};
    bool afRoiY0{true};
    bool afRoiX1{true};
    bool afRoiY1{true};
    bool zoomSpeed{true};
    bool zoomHwSpeed{true};
    bool zoomHwMaxSpeed{true};
    bool focusSpeed{true};
    bool focusHwSpeed{true};
    bool focusHwMaxSpeed{true};
```

```
    bool irisSpeed{true};
    bool irisHwSpeed{true};
    bool irisHwMaxSpeed{true};
    bool zoomHwTeleLimit{true};
    bool zoomHwWideLimit{true};
    bool focusHwFarLimit{true};
    bool focusHwNearLimit{true};
    bool irisHwOpenLimit{true};
    bool irisHwCloseLimit{true};
    bool focusFactor{true};
    bool isConnected{true};
    bool afHwSpeed{true};
    bool focusFactorThreshold{true};
    bool refocusTimeoutSec{true};
    bool afIsActive{true};
    bool irisMode{true};
    bool autoAfRoiWidth{true};
    bool autoAfRoiHeight{true};
    bool autoAfRoiBorder{true};
    bool afRoiMode{true};
    bool extenderMode{true};
    bool stabiliserMode{true};
    bool afRange{true};
    bool xFovDeg{true};
    bool yFovDeg{true};
    bool logMode{true};
    bool temperature{true};
    bool isOpen{false};
    bool type{true};
    bool custom1{true};
    bool custom2{true};
    bool custom3{true};
} LensParamsMask;
```

Example without parameters mask:

```
// Encode data.
LensParams in;
in.logMode = 3;
uint8_t data[1024];
int size = 0;
in.encode(data, 1024, size);
cout << "Encoded data size: " << size << " bytes" << endl;
```

Example with parameters mask:

```
// Prepare params.
LensParams in;
in.logMode = 3;

// Prepare mask.
LensParamsMask mask;
mask.logMode = false; // Exclude logMode. Others by default.

// Encode.
uint8_t data[1024];
int size = 0;
in.encode(data, 1024, size, &mask);
cout << "Encoded data size: " << size << " bytes" << endl;
```

Deserialize lens params

LensParams class provides method **decode(...)** to deserialize lens params. Deserialization of lens params necessary in case when you need to receive lens params via communication channels. Method automatically recognizes which parameters were serialized by **encode(...)** method. Method doesn't decode fields: **initString** and **fovPoints**. Method declaration:

```
bool decode(uint8_t* data, int dataSize);
```

Parameter	Value
data	Pointer to data buffer.
dataSize	Size of data.

Returns: TRUE if data decoded (deserialized) or FALSE if not.

Example:

```
// Encode data.
LensParams in;
uint8_t data[1024];
int size = 0;
in.encode(data, 1024, size);
cout << "Encoded data size: " << size << " bytes" << endl;

// Decode data.
LensParams out;
if (!out.decode(data, size))
    cout << "Can't decode data" << endl;
```

Read and write lens params to JSON file

Lens interface class library depends on **ConfigReader** library which provides method to read params from JSON file and to write params to JSON file. Example of writing and reading params to JSON file:

```
// Prepare random params.
LensParams in;
for (int i = 0; i < 5; ++i)
{
    FovPoint pt;
    pt.hwZoomPos = rand() % 255;
    pt.xFovDeg = rand() % 255;
    pt.yFovDeg = rand() % 255;
    in.fovPoints.push_back(pt);
}

// Write params to file.
cr::utils::ConfigReader inConfig;
inConfig.set(in, "lensParams");
inConfig.writeToFile("TestLensParams.json");

// Read params from file.
cr::utils::ConfigReader outConfig;
if(!outConfig.readFromFile("TestLensParams.json"))
{
    cout << "Can't open config file" << endl;
    return false;
}
```

TestLensParams.json will look like:

```
{
    "lensParams": {
        "afHwSpeed": 93,
        "afRange": 128,
        "afRoiMode": 239,
        "afRoiX0": 196,
        "afRoiX1": 252,
        "afRoiY0": 115,
        "afRoiY1": 101,
        "autoAfRoiBorder": 70,
        "autoAfRoiHeight": 125,
        "autoAfRoiWidth": 147,
        "custom1": 91.0,
        "custom2": 236.0,
        "custom3": 194.0,
        "extenderMode": 84,
        "filterMode": 49,
        "focusFactorThreshold": 98.0,
        "focusHwFarLimit": 228,
        "focusHwMaxSpeed": 183,
        "focusHwNearLimit": 47,
        "focusMode": 111,
        "fovPoints": [
```

```
            {
                "hwZoomPos": 55,
                "xFovDeg": 6.0,
                "yFovDeg": 51.0
            },
            {
                "hwZoomPos": 63,
                "xFovDeg": 249.0,
                "yFovDeg": 33.0
            },
            {
                "hwZoomPos": 4,
                "xFovDeg": 121.0,
                "yFovDeg": 144.0
            },
            {
                "hwZoomPos": 53,
                "xFovDeg": 214.0,
                "yFovDeg": 153.0
            },
            {
                "hwZoomPos": 143,
                "xFovDeg": 15.0,
                "yFovDeg": 218.0
            }
        ],
        "initString": "dfhglsjirhuhjfb",
        "irisHwCloseLimit": 221,
        "irisHwMaxSpeed": 79,
        "irisHwOpenLimit": 211,
        "irisMode": 206,
        "logMode": 216,
        "refocusTimeoutSec": 135,
        "stabiliserMode": 137,
        "type": 125,
        "zoomHwMaxSpeed": 157,
        "zoomHwTeleLimit": 68,
        "zoomHwWideLimit": 251
    }
}
```


Build and connect to your project

Typical commands to build **Flc** library:

```
cd Flc
mkdir build
cd build
cmake ..
make
```

If you want connect **Flc** library to your CMake project as source code you can make follow. For example, if your repository has structure:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
```

Create folder **3rdparty** and copy folder of **Flc** repository there. New structure of your repository:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    Flc
```

Create CMakeLists.txt file in **3rdparty** folder. CMakeLists.txt should contain:

```
cmake_minimum_required(VERSION 3.13)

###############################################################################
## 3RD-PARTY
## dependencies for the project
###############################################################################
project(3rdparty LANGUAGES CXX)

###############################################################################
## SETTINGS
## basic 3rd-party settings before use
###############################################################################
# To inherit the top-level architecture when the project is used as a submodule.
SET(PARENT ${PARENT}_YOUR_PROJECT_3RDPARTY)
# Disable self-overwriting of parameters inside included subdirectories.
SET(${PARENT}_SUBMODULE_CACHE_OVERWRITE OFF CACHE BOOL "" FORCE)

###############################################################################
## CONFIGURATION
## 3rd-party submodules configuration
###############################################################################
SET(${PARENT}_SUBMODULE_FLC                                 ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_FLC)
    SET(${PARENT}_FLC                                       ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_FLC_TEST                                  OFF CACHE BOOL "" FORCE)
endif()

###############################################################################
## INCLUDING SUBDIRECTORIES
## Adding subdirectories according to the 3rd-party configuration
###############################################################################
if (${PARENT}_SUBMODULE_FLC)
    add_subdirectory(Flc)
```

```
endif()
```

File **3rdparty/CMakeLists.txt** adds folder **Flc** to your project and excludes test application and example from compiling. Your repository new structure will be:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    CMakeLists.txt
    Flc
```

Next you need include folder 3rdparty in main **CMakeLists.txt** file of your repository. Add string at the end of your main **CMakeLists.txt**:

```
add_subdirectory(3rdparty)
```

Next you have to include Lens library in your **src/CMakeLists.txt** file:

```
target_link_libraries(${PROJECT_NAME} Flc)
```

Done!

Simple example

Simple example is application which initializes controller and provide few options to user to control camera.

```
#include <iostream>
#include "Flc.h"

int main(void)
{
    // Init camera controller.
    cr::fuji::Flc controller;
    if (!controller.openCamera("/dev/ttyUSB0;9600;50"))
        return -1;

    while (true)
    {
        // Main dialog.
        int option = -1;
        std::cout << "Options (1:Zoom tele, 2:Zoom wide, 3:Zoom stop) : ";
        std::cin >> option;

        // Get all lens params.
        cr::lens::LensParams lensParams;
        controller.getParams(lensParams);

        switch (option)
```

```
        {
        case 1: // Zoom tele.
            controller.executeCommand(cr::lens::LensCommand::ZOOM_TELE);
            break;
        case 2: // Zoom wide.
            controller.executeCommand(cr::lens::LensCommand::ZOOM_WIDE);
            break;
        case 3: // Zoom stop.
            controller.executeCommand(cr::lens::LensCommand::ZOOM_STOP);
            break;
        default:
            break;
        }
    }
    return 1;
}
```