Ylc C++ library

v4.0.0

Table of contents

- Overview
- Versions
- Library files
- Ylc class description
  - 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
- Data structures
  - LensCommand enum
  - LensParam enum
- LensParams class description
  - 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 **Ylc** C++ library is a software controller for Yamano lenses | VS Technology lenses. These lenses provide control interface over serial port. The **Ylc** 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) and SerialPort library (provides functions to work with serial ports, Apache 2.0 license). The **Ylc** 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. 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	11.09.2022	- First version.
2.0.0	30.03.2024	- Add Logger.
3.0.0	16.05.2023	- Lens interface updated. - Submodules updated.
4.0.0	13.02.2024	- Lens interface updated. - Submodules updated. - Documentation updated.

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.
README.md ---------------------- Documentation.
3rdparty ----------------------- Folder with 3rdparty libraries.
    CMakeLists.txt ------------- CMake file to include 3rdparty libraries.
    Lens ----------------------- Folder with Lens interface library source code.
    Logger --------------------- Folder with Logger library source code.
    SerialPort ----------------- Folder with SerialPort library source code.
    YlParser ------------------- Folder with YlParser library source code.
        CMakeLists.txt --------- CMake file if YlParser library.
        src -------------------- Folder with library source code.
            YlParser.h --------- Library header file.
```

```
            YlParser.cpp ------- C++ implementation file.
            YlParserVersion.h -- Header file with library version.
src ---------------------------- Folder with library source code.
    CMakeLists.txt ------------- CMake file of the library.
    Ylc.h ---------------------- Main library header file.
    Ylc.cpp -------------------- C++ implementation file.
    YlcVersion.h --------------- Header file with library version.
    YlcVersion.h.in ------------ CMake file to generate version header.
test --------------------------- Folder with test application files.
    CMakeLists.txt ------------- CMake file for test application.
    main.cpp ------------------- Source C++ file of test application.
example ------------------------ Folder with simple example files.
    CMakeLists.txt ------------- CMake file for simple example.
    main.cpp ------------------- Source C++ file of simple example.
```

Ylc class description

Class declaration

Ylc.h file contains **Ylc** class declaration.

```
class Ylc : public Lens
{
public:

    /// Class constructor.
    Ylc();

    /// Class destructor.
    ~Ylc();

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

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

    /// Init lens controller.
    bool initLens(LensParams& params) override;

    /// Close serial port and stop communication thread.
    void closeLens() override;

    /// Get controller initialization status.
    bool isLensOpen() override;

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

    /// Set the lens controller parameter.
    bool setParam(LensParam id, float value) override;
```

```
    /// Get the lens controller param.
    float getParam(LensParam id) override;

    /// Get the lens controller params.
    void getParams(LensParams& params) override;

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

    /// Add video frame for auto focus purposes.
    void addVideoFrame(cr::video::Frame& frame) override;

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

getVersion method

getVersion() static method returns string of current version of **Ylc** class. Method declaration:

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

Method can be used without **Ylc** class instance. Example:

```
cout << "Ylc version: " << Ylc::getVersion() << endl;
```

Console output:

```
Ylc version: 4.0.0
```

openLens method

openLens(...) opens serial port to communicate with Yamano lenses. Lens parameters will be initialized by default. After successful initialization the library will run communication threads (thread to communicate with equipment via serial port) if it not run before. If serial port already open this method will return TRUE. Method declaration:

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

Parameter	Value
initString	Initialization string contains full serial port name, baudrate, timeout separated by ";". Example: **"/dev/ttyUSB0;9600;50"**. Baudrate and timeout optional params. Variants: [serial port name] [serial port name];[baudrate] [serial port name];[baudrate];[timeout]

Returns: TRUE if the serial port open or FALSE if not.

initLens method

initLens(...) initializes lens 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 equipment via serial port) if it not run before. Method declaration:

```
bool initLens(LensParams& params) override;
```

Parameter	Value
params	LensParams parameters class. LensParams class includes initString wich used in **openLens(...)** method. See description of LensParams class.

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

isLensConnected method

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

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

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

setParam method

setParam(...) method sets new lens parameters value. **Ylc** provides thread-safe **setParam(...)** method call. This means that the **setParam(...)** method can be safely called from any thread. Method declaration:

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

Parameter	Description
id	Lens parameter ID according to LensParam enum.
value	Lens parameter value. Value depends on parameter ID (LensParam enum).

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

getParam method

getParam(...) method returns lens parameter value. **Ylc** provides thread-safe **getParam(...)** method call. This means that the **getParam(...)** method can be safely called from any thread. 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 doesn't exist (not supported).

getParams method

getParams(...) method returns lens parameters class. **Ylc** provides thread-safe **getParams(...)** method call. This means that the **getParams(...)** method can be safely called from any thread. Method declaration:

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

Parameter	Description
params	Reference to LensParams class object wich includes all lens parameters.

executeCommand method

executeCommand(...) method to execute lens action command. **Ylc** provides thread-safe **executeCommand(...)** method call. This means that the **executeCommand(...)** method can be safely called from any thread. Method declaration:

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

Parameter	Description
id	Lens action command ID according to LensCommand enum.
arg	Lens action command argument. Value depends on command ID (see description of LensCommand enum).

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

addVideoFrame method

addVideoFrame(...) method copies video frame data to lens controller to perform autofocus algorithm. Autofocus algorithm **not supported** by **Ylc**. The method doesn't do anything and only provides an **Lens** interface. Method declaration:

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

Parameter	Description
frame	Video frame object (see Frame class description).

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 classes. If command decoded the method will call setParam(...) or executeCommand(...) methods for lens interfaces. 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. To generate input command you may use encodeCommand(...) and encodeSetParamCommand(...) methods.
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.
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** class instance. 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.

Byte	Value	Description
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). Command doesn't have arguments. User should be able
    /// to set zoom movement speed via lens parameters.
    ZOOM_TELE = 1,
    /// Move zoom wide (out). Command doesn't have arguments. User should be
    /// able to set zoom movement speed via lens parameters.
    ZOOM_WIDE,
    /// Move zoom to position. Lens controller should have zoom range from
    /// 0 (full wide) to 65535 (full tele) regardless of the hardware value of
    /// the zoom position. If the minimum and maximum zoom position limits are
    /// set by the user in the lens parameters, the range of the hardware zoom
    /// position must be scaled to the user space 0-65535 range.
    /// Command argument: zoom position 0-65535. User should be able to set zoom
    /// movement speed via lens parameters.
    ZOOM_TO_POS,
    /// Stop zoom moving including stop zoom to position command.
    ZOOM_STOP,
    /// Move focus far. Command doesn't have arguments. User should be able to
    /// set focus movement speed via lens parameters.
    FOCUS_FAR,
    /// Move focus near. Command doesn't have arguments. User should be able to
    /// set focus movement speed via lens parameters.
    FOCUS_NEAR,
    /// Move focus to position. Lens controller should have focus range from 0
    /// (full near) to 65535 (full far) regardless of the hardware value of the
    /// focus position. If the minimum and maximum focus position limits are
    /// set by the user in the lens parameters, the range of the hardware focus
    /// position must be scaled to the 0-65535 user space range.
    /// Command argument: focus position 0-65535. User should be able to set
    /// focus movement speed via lens parameters.
    FOCUS_TO_POS,
    /// Stop focus moving including stop focus to position command.
    FOCUS_STOP,
    /// Move iris open. Command doesn't have arguments. User should be able to
    /// set iris movement speed via lens parameters.
    IRIS_OPEN,
    /// Move iris close. Command doesn't have arguments. User should be able
    /// to set iris movement speed via lens parameters.
    IRIS_CLOSE,
    /// 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_TO_POS,
    /// Stop iris moving including stop iris to position command.
    /// Command doesn't have arguments.
    IRIS_STOP,
    /// Start autofocus.  Command doesn't have arguments.
    AF_START,
    /// Stop autofocus.  Command doesn't have arguments.
    AF_STOP,
    /// Restart lens controller.
    RESTART,
    /// Detect zoom and focus hardware ranges. After execution this command the
    /// lens controller should automatically set at least parameters
    /// (LensParam enum): ZOOM_HW_TELE_LIMIT, ZOOM_HW_WIDE_LIMIT,
    /// FOCUS_HW_FAR_LIMIT and FOCUS_HW_NEAR_LIMIT.
    DETECT_HW_RANGES
};
```

Table 2 - Lens commands description. Some commands may be unsupported by lens controller.

Command	Description
ZOOM_TELE	Move zoom tele (in). Command **doesn't have arguments**.
ZOOM_WIDE	Move zoom wide (out). Command **doesn't have arguments**.
ZOOM_TO_POS	Move zoom to position. Command argument: zoom position **0 (full wide) - 65535 (full tele)**.
ZOOM_STOP	Stop zoom moving including stop zoom to position command. Command **doesn't have arguments**.
FOCUS_FAR	Move focus far. Command **doesn't have arguments**.
FOCUS_NEAR	Move focus near. Command **doesn't have arguments**.
FOCUS_TO_POS	Move focus to position. Command argument: focus position **0 (full near) - 65535 (full far)**.
FOCUS_STOP	Stop focus moving including stop focus to position command. Command **doesn't have arguments**.
IRIS_OPEN	Move iris open. Command **doesn't have arguments**.
IRIS_CLOSE	Move iris close. Command **doesn't have arguments**.
IRIS_TO_POS	Move iris to position. Command argument: iris position **0 (full close) - 65535 (full open)**.
IRIS_STOP	Stop iris moving including stop iris to position command.
AF_START	Start autofocus. Command **doesn't have arguments**.
AF_STOP	Stop autofocus. Command **doesn't have arguments**.
RESTART	**Not supported** by Ylc class.

Command	Description
DETECT_HW_RANGES	**Not supported** by Ylc class.

LensParam enum

Enum declaration:

```
enum class LensParam
{
    /// Zoom position. Setting a parameter is equivalent to the command
    /// ZOOM_TO_POS. Lens controller should have zoom range from 0 (full wide)
    /// to 65535 (full tele) regardless of the hardware value of the zoom
    /// position. If the minimum and maximum zoom position limits are set by
    /// the user in the lens parameters, the range of the hardware zoom position
    /// must be scaled to the user space 0-65535 range. Parameter value: zoom
    /// position 0-65535. User should be able to set zoom movement speed via
    /// lens parameters.
    ZOOM_POS = 1,
    /// Hardware zoom position. Parameter value depends on implementation and
    /// lens hardware.
    ZOOM_HW_POS,
    /// Focus position. Setting a parameter is equivalent to the command
    /// FOCUS_TO_POS. Lens controller should have focus range from 0 (full near)
    /// to 65535 (full far) regardless of the hardware value of the focus
    /// position. If the minimum and maximum focus position limits are set by
    /// the user in the lens parameters, the range of the hardware focus
    /// position must be scaled to the 0-65535 user space range. Parameter
    /// value: focus position 0-65535. User should be able to set focus movement
    /// speed via lens parameters.
    FOCUS_POS,
    /// Hardware focus position. Parameter value depends on implementation and
    /// lens hardware.
    FOCUS_HW_POS,
    /// Iris position. Setting a parameter is equivalent to the command
    /// IRIS_TO_POS. 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. Parameter value: iris
    /// position 0-65535. User should be able to set iris movement speed via
    /// lens parameters.
    IRIS_POS,
    /// Hardware iris position. Parameter value depends on particular lens
    /// controller.
    IRIS_HW_POS,
    /// Focus mode. Parameter value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual focus control,
    /// 1 - Auto focus control.
    FOCUS_MODE,
    /// Filter mode. Parameter value depends on implementation but it is
    /// recommended to keep default values: 0 - Filter on, 1 - Filter off.
    FILTER_MODE,
```

```
    /// Autofocus ROI top-left corner horizontal position in pixels.
    /// Autofocus ROI is rectangle.
    AF_ROI_X0,
    /// Autofocus ROI top-left corner vertical position in pixels.
    /// Autofocus ROI is rectangle.
    AF_ROI_Y0,
    /// Autofocus ROI bottom-right corner horizontal position in pixels.
    /// Autofocus ROI is rectangle.
    AF_ROI_X1,
    /// Autofocus ROI bottom-right corner vertical position in pixels.
    /// Autofocus ROI is rectangle.
    AF_ROI_Y1,
    /// Zoom speed. Lens controller should have zoom speed range from 0 to
    /// 100% of max hardware zoom speed (parameter ZOOM_HW_MAX_SPEED).
    /// If the user sets a new parameter value of the ZOOM_HW_SPEED the
    /// parameter ZOOM_SPEED must be updated automatically. Formula for
    /// calculating speed:
    /// ZOOM_SPEED = ( ZOOM_HW_SPEED / ZOOM_HW_MAX_SPEED) * 100.
    ZOOM_SPEED,
    /// Zoom hardware speed. Value depends on implementation and lens hardware.
    /// The value of the parameters must be <= ZOOM_HW_MAX_SPEED parameter.
    /// If the user sets a new parameter value of the ZOOM_SPEED parameter
    /// the parameter ZOOM_HW_SPEED must be updated automatically.
    /// Formula for calculating hardware speed:
    /// ZOOM_HW_SPEED = ( ZOOM_SPEED / 100 ) * ZOOM_HW_MAX_SPEED.
    ZOOM_HW_SPEED,
    /// Maximum zoom hardware speed. Value depends on implementation.
    /// If user sets new ZOOM_HW_MAX_SPEED value the parameters
    /// ZOOM_SPEED must be updated automatically. If new value of
    /// ZOOM_HW_MAX_SPEED parameter will be less than ZOOM_HW_SPEED the
    /// parameter ZOOM_HW_SPEED must be reduced automatically.
    ZOOM_HW_MAX_SPEED,
    /// Focus speed. Lens controller should have focus speed range from 0 to
    /// 100% of max hardware focus speed (parameter FOCUS_HW_MAX_SPEED).
    /// If the user sets a new parameter value of the FOCUS_HW_SPEED the
    /// parameter FOCUS_SPEED must be updated automatically. Formula for
    /// calculating speed: FOCUS_SPEED = ( FOCUS_HW_SPEED / FOCUS_HW_MAX_SPEED)
    /// * 100.
    FOCUS_SPEED,
    /// Focus hardware speed. Value depends on on implementation and lens
    /// hardware. The value of the parameters must be <= FOCUS_HW_MAX_SPEED
    /// parameter. If the user sets a new parameter value of the FOCUS_SPEED
    /// parameter the parameter FOCUS_HW_SPEED must be updated automatically.
    /// Formula for calculating hardware speed:
    /// FOCUS_HW_SPEED = ( FOCUS_SPEED / 100 ) * FOCUS_HW_MAX_SPEED.
    FOCUS_HW_SPEED,
    /// Maximum focus hardware speed. Value depends on implementation.
    /// If user sets new FOCUS_HW_MAX_SPEED value the parameters
    /// FOCUS_SPEED and FOCUS_HW_SPEED must be updated by lens controller
    /// automatically. If new value of FOCUS_HW_MAX_SPEED parameter will be
    /// less than FOCUS_HW_SPEED the parameter FOCUS_HW_SPEED must be reduced
    /// automatically.
    FOCUS_HW_MAX_SPEED,
    /// Iris speed. Lens controller should have iris speed range from 0 to 100%
    /// of max hardware iris speed (parameter IRIS_HW_MAX_SPEED). If the user
    /// sets a new parameter value of the IRIS_HW_SPEED the parameter IRIS_SPEED
```

```
    /// must be updated automatically. Formula for calculating speed:
    /// IRIS_SPEED = ( IRIS_HW_SPEED / IRIS_HW_MAX_SPEED) * 100.
    IRIS_SPEED,
    /// Iris hardware speed. Value depends on implementation and les hardware.
    /// The value of the parameters must be <= IRIS_HW_MAX_SPEED parameter.
    /// If the user sets a new parameter value of the IRIS_SPEED parameter
    /// the parameter IRIS_HW_SPEED must be updated automatically. Formula
    /// for calculating hardware speed:
    /// IRIS_HW_SPEED = ( IRIS_SPEED / 100 ) * IRIS_HW_MAX_SPEED.
    IRIS_HW_SPEED,
    /// Maximum iris hardware speed. Value depends on implementation. If user
    /// sets new IRIS_HW_MAX_SPEED value the parameters IRIS_SPEED and
    /// IRIS_HW_SPEED must be updated automatically. If new value of
    /// IRIS_HW_MAX_SPEED parameter will be less than IRIS_HW_SPEED the
    /// parameter IRIS_HW_SPEED must be reduced automatically.
    IRIS_HW_MAX_SPEED,
    /// Zoom hardware tele limit. Value depends on implementation and lens
    /// hardware. Lens controller should control zoom position. Lens controller
    /// should stop zoom moving if hardware zoom position will be our of limits.
    ZOOM_HW_TELE_LIMIT,
    /// Zoom hardware wide limit. Value depends on implementation and lens
    /// hardware. Lens controller should control zoom position. Lens controller
    /// should stop zoom moving if hardware zoom position will be our of limits.
    ZOOM_HW_WIDE_LIMIT,
    /// Focus hardware far limit. Value depends on on implementation and lens
    /// hardware. Lens controller should control focus position. Lens controller
    /// should stop focus moving if hardware focus position will be our of
    /// limits.
    FOCUS_HW_FAR_LIMIT,
    /// Focus hardware near limit. Value depends on implementation and lens
    /// hardware. Lens controller should control focus position. Lens controller
    /// should stop focus moving if hardware focus position will be our of
    /// limits.
    FOCUS_HW_NEAR_LIMIT,
    /// Iris hardware open limit. Value depends on implementation and lens
    /// hardware. Lens controller should control iris position. Lens controller
    /// should stop iris moving if hardware iris position will be our of limits.
    IRIS_HW_OPEN_LIMIT,
    /// Iris hardware close limit. Value depends on implementation and lens
    /// hardware. Lens controller should control iris position. Lens controller
    /// should stop iris moving if hardware iris position will be our of limits.
    IRIS_HW_CLOSE_LIMIT,
    /// Focus factor if it was calculated. If not calculated must be -1.
    /// Value depends on particular lens controller.
    FOCUS_FACTOR,
    /// Lens connection status. Connection status shows if the lens controller
    /// has data exchange with lens equipment. For example, if lens has serial
    /// port lens controller connects to serial port
    /// (opens serial port file in OS) but lens can be not active (no power).
    /// In this case connection status shows that lens controller doesn't have
    /// data exchange with lens equipment (methos will return 0). It lens
    /// controller has data exchange with lens equipment the method will
    /// return 1. If lens controller not initialize the connection status always
    /// FALSE. Value: 0 - not connected. 1 - connected.
    IS_CONNECTED,
    /// Focus hardware speed in autofocus mode. Value depends on implementation
```

```
    /// and lens hardware.
    FOCUS_HW_AF_SPEED,
    /// Threshold for changes of focus factor to start refocus. Value:
    /// 0% - no check, 100% - changing x2.
    FOCUS_FACTOR_THRESHOLD,
    /// Timeout for automatic refocus in seconds. Value:
    /// 0 - no automatic refocus, 100000 - maximum value.
    REFOCUS_TIMEOUT_SEC,
    /// Flag about active autofocus algorithm. Value: 0 - autofocus not working,
    /// 1 - working.
    AF_IS_ACTIVE,
    /// Iris mode. Value depends on implementation but it is recommended to keep
    /// default values: 0 - manual iris control, 1 - auto iris control.
    IRIS_MODE,
    /// ROI width (pixels) for autofocus algorithm when lens controller detects
    /// ROI position automatically. Value: from 8 to (video frame width -
    /// AUTO_AF_ROI_BORDER * 2).
    AUTO_AF_ROI_WIDTH,
    /// ROI height (pixels) for autofocus algorithm when lens controller
    /// detects ROI position automatically. Value: from 8
    /// (video frame width - AUTO_AF_ROI_BORDER * 2).
    AUTO_AF_ROI_HEIGHT,
    /// Video frame border size (along vertical and horizontal axes).
    /// Value: border size from 0 to video frame
    /// min(video frame width/height) / 2.
    AUTO_AF_ROI_BORDER,
    /// AF ROI mode (write/read). Value: 0 - Manual position, 1 - Auto position.
    AF_ROI_MODE,
    /// Lens extender mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - no extender, 1 - x2 extender.
    EXTENDER_MODE,
    /// Lens stabilization mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - no stabilization,
    /// 1 - stabilization.
    STABILIZER_MODE,
    /// Autofocus range. Value depends on implementation.
    AF_RANGE,
    /// Current horizontal Field of view, degree. Field of view calculated by
    /// lens controller according to initial params or by reading directly from
    /// lens hardware.
    X_FOV_DEG,
    /// Current vertical Field of view, degree. Field of view calculated by lens
    /// controller according to initial params or by reading directly from lens
    /// hardware.
    Y_FOV_DEG,
    /// Logging mode. Values: 0 - Disable, 1 - Only file, 2 - Only terminal,
    /// 3 - File and terminal.
    LOG_MODE,
    /// Lens temperature, degree.
    TEMPERATURE,
    /// Lens controller initialization status. Open status shows if the lens
    /// controller initialized or not but doesn't show if lens controller has
    /// communication with lens equipment. For example, if lens has serial port
    /// lens controller connects to serial port (opens serial port file in OS)
    /// but lens can be not active (no power). In this case open status just
    /// shows that lens controller has opened serial port. Values: 0 - not open
```

```
    /// not initialized), 1 - open (initialized).
    IS_OPEN,
    /// Lens type. Value depends on implementation. Type allows to lens
    /// initialize necessary parameters for particular lens hardware.
    TYPE,
    /// Lens custom parameter. Value depends on particular lens controller.
    /// Custom parameters used when particular lens equipment has specific
    /// unusual parameter.
    CUSTOM_1,
    /// Lens custom parameter. Value depends on particular lens controller.
    /// Custom parameters used when particular lens equipment has specific
    /// unusual parameter.
    CUSTOM_2,
    /// Lens custom parameter. Value depends on particular lens controller.
    /// Custom parameters used when particular lens equipment has specific
    /// unusual parameter.
    CUSTOM_3
};
```

Table 3 - Lens params description. Some params may be unsupported by lens controller.

Parameter	Access	Description
ZOOM_POS	read / write	Zoom position. Setting a parameter is equivalent to the command ZOOM_TO_POS. Param argument: zoom position **0 (full wide) - 65535 (full tele)**. It doesn't change the zoom position value immediately. The zoom position changes only when the lens moves.
ZOOM_HW_POS	read / write	Hardware zoom position. Setting a parameter move zoom to HW position. Param argument: zoom position **ZOOM_HW_WIDE_LIMIT - ZOOM_HW_TELE_LIMIT**. By default **ZOOM_HW_WIDE_LIMIT = 0**, **ZOOM_HW_TELE_LIMIT = 65535**. It doesn't change the zoom position value immediately. The zoom position changes only when the lens moves.
FOCUS_POS	read / write	Focus position. Setting a parameter is equivalent to the command FOCUS_TO_POS. Parameter value: focus position **0 (full near) - 65535 (full far)**. It doesn't change the focus position value immediately. The focus position changes only when the lens moves.
FOCUS_HW_POS	read / write	Focus position. Setting a parameter move focus to HW position. Parameter value: focus position **FOCUS_HW_NEAR_LIMIT - FOCUS_HW_FAR_LIMIT**. By default **FOCUS_HW_NEAR_LIMIT = 0, FOCUS_HW_FAR_LIMIT = 65535**. It doesn't change the focus position value immediately. The focus position changes only when the lens moves.

Parameter	Access	Description
IRIS_POS	read / write	Iris position. Setting a parameter is equivalent to the command IRIS_TO_POS. Param argument: iris position **0 (full close) - 65535 (full open)**. It doesn't change the iris position value immediately. The iris position changes only when the lens moves.
IRIS_HW_POS	read / write	Iris position. Setting a parameter move iris to HW position. Parameter value: iris position **IRIS_HW_CLOSE_LIMIT - IRIS_HW_OPEN_LIMIT**. By default **IRIS_HW_CLOSE_LIMIT = 0, IRIS_HW_OPEN_LIMIT= 65535**. It doesn't change the iris position value immediately. The iris position changes only when the lens moves.
FOCUS_MODE	read / write	Focus mode: **0** - Manual focus control.
FILTER_MODE	read / write	Filter mode: **0** - VC filter auxiliary (out) * 2 **1** - VC filter auxiliary (in) * 2
AF_ROI_X0	read / write	**Not supported** by Ylc class.
AF_ROI_Y0	read / write	**Not supported** by Ylc class.
AF_ROI_X1	read / write	**Not supported** by Ylc class.
AF_ROI_Y1	read / write	**Not supported** by Ylc class.
ZOOM_SPEED	read / write	**Not supported** by Ylc class.
ZOOM_HW_SPEED	read / write	**Not supported** by Ylc class.
ZOOM_HW_MAX_SPEED	read / write	**Not supported** by Ylc class.
FOCUS_SPEED	read / write	**Not supported** by Ylc class.
FOCUS_HW_SPEED	read / write	**Not supported** by Ylc class.
FOCUS_HW_MAX_SPEED	read / write	**Not supported** by Ylc class.
IRIS_SPEED	read / write	**Not supported** by Ylc class.

Parameter	Access	Description
IRIS_HW_SPEED	read / write	**Not supported** by Ylc class.
IRIS_HW_MAX_SPEED	read / write	**Not supported** by Ylc class.
ZOOM_HW_TELE_LIMIT	read / write	Zoom hardware tele limit. Default value **65535**.
ZOOM_HW_WIDE_LIMIT	read / write	Zoom hardware wide limit. Default value **0**.
FOCUS_HW_FAR_LIMIT	read / write	Focus hardware far limit. Default value **65535**.
FOCUS_HW_NEAR_LIMIT	read / write	Focus hardware near limit. Default value **0**.
IRIS_HW_OPEN_LIMIT	read / write	Iris hardware tele limit. Default value **65535**.
IRIS_HW_CLOSE_LIMIT	read / write	Iris hardware wide limit. Default value **0**.
FOCUS_FACTOR	read only	**Not supported** by Ylc class.
IS_CONNECTED	read only	Lens connection status. Connection status shows if the lens controller has data exchange with lens equipment. Values: **0** - no response from lens. **1** - connected.
FOCUS_HW_AF_SPEED	read / write	**Not supported** by Ylc class.
FOCUS_FACTOR_THRESHOLD	read / write	**Not supported** by Ylc class.
REFOCUS_TIMEOUT_SEC	read / write	**Not supported** by Ylc class.
AF_IS_ACTIVE	read only	**Not supported** by Ylc class.
IRIS_MODE	read / write	Iris mode: **0** - Auto iris off. **1** - Auto iris on.
AUTO_AF_ROI_WIDTH	read / write	**Not supported** by Ylc class.
AUTO_AF_ROI_HEIGHT	read / write	**Not supported** by Ylc class.

Parameter	Access	Description
AUTO_AF_ROI_BORDER	read / write	**Not supported** by Ylc class.
AF_ROI_MODE	read / write	**Not supported** by Ylc class.
EXTENDER_MODE	read / write	Extender mode: **0** - X2 extender lens in. **1** - X2 extender lens out.
STABILIZER_MODE	read / write	**Not supported** by Ylc class.
AF_RANGE	read / write	**Not supported** by Ylc class.
X_FOV_DEG	read / write	**Not supported** by Ylc class.
Y_FOV_DEG	read / write	**Not supported** by Ylc class.
LOG_MODE	read / write	Logging mode. Values: **0** - Disable. **1** - Only file. **2** - Only terminal. **3** - File and terminal.
TEMPERATURE	read only	**Not supported** by Ylc class.
IS_OPEN	read only	Lens controller initialization status. Open status shows if the lens controller initialized or not but doesn't show if lens controller has communication with lens equipment. Values: **0** - not open (not initialized). **1** - open (initialized).
TYPE	read / write	**Not supported** by Ylc class.
CUSTOM_1	read / write	**Not supported** by Ylc class.
CUSTOM_2	read / write	**Not supported** by Ylc class.
CUSTOM_3	read / write	**Not supported** by Ylc class.

LensParams class description

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

Class declaration

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

```
/// Field of view point class.
class FovPoint
{
public:
    /// Hardware zoom pos.
    int hwZoomPos{0};
    /// Horizontal field of view, degree.
    float xFovDeg{0.0f};
    /// Vertical field of view, degree.
    float yFovDeg{0.0f};

    JSON_READABLE(FovPoint, hwZoomPos, xFovDeg, yFovDeg);

    /// operator =
    FovPoint& operator= (const FovPoint& src);
};

/// Lens params structure.
class LensParams
{
public:
    /// Initialization string. Particular lens controller can have unique init
    /// string format. But it is recommended to use '**;**' symbol to divide
    /// parts of initialization string. Recommended initialization string format
    /// for controllers which uses serial port: "/dev/ttyUSB0;9600;100"
    /// ("/dev/ttyUSB0" - serial port name, "9600" - baudrate, "100" - serial
    /// port read timeout).
    std::string initString{"/dev/ttyUSB0;9600;20"};
    /// Zoom position. Setting a parameter is equivalent to the command
    /// ZOOM_TO_POS. Lens controller should have zoom range from 0 (full wide)
    /// to 65535 (full tele) regardless of the hardware value of the zoom
    /// position. If the minimum and maximum zoom position limits are set by
    /// the user in the lens parameters, the range of the hardware zoom position
    /// must be scaled to the user space 0-65535 range. Parameter value: zoom
    /// position 0-65535. User should be able to set zoom movement speed via
    /// lens parameters.
    int zoomPos{0};
    /// Hardware zoom position. Parameter value depends on implementation and
    /// lens hardware.
    int zoomHwPos{0};
    /// Focus position. Setting a parameter is equivalent to the command
```

```
    /// FOCUS_TO_POS. Lens controller should have focus range from 0 (full near)
    /// to 65535 (full far) regardless of the hardware value of the focus
    /// position. If the minimum and maximum focus position limits are set by
    /// the user in the lens parameters, the range of the hardware focus
    /// position must be scaled to the 0-65535 user space range. Parameter
    /// value: focus position 0-65535. User should be able to set focus movement
    /// speed via lens parameters.
    int focusPos{0};
    /// Hardware focus position. Parameter value depends on implementation and
    /// lens hardware.
    int focusHwPos{0};
    /// Iris position. Setting a parameter is equivalent to the command
    /// IRIS_TO_POS. 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. Parameter value: iris
    /// position 0-65535. User should be able to set iris movement speed via
    /// lens parameters.
    int irisPos{0};
    /// Hardware iris position. Parameter value depends on inplementation.
    int irisHwPos{0};
    /// Focus mode. Parameter value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual focus control,
    /// 1 - Auto focus control.
    int focusMode{0};
    /// Filter mode. Parameter value depends on implementation but it is
    /// recommended to keep default values: 0 - Filter on, 1 - Filter off.
    int filterMode{0};
    /// Autofocus ROI top-left corner horizontal position in pixels.
    /// Autofocus ROI is rectangle.
    int afRoiX0{0};
    /// Autofocus ROI top-left corner vertical position in pixels.
    /// Autofocus ROI is rectangle.
    int afRoiY0{0};
    /// Autofocus ROI bottom-right corner horizontal position in pixels.
    /// Autofocus ROI is rectangle.
    int afRoiX1{0};
    /// Autofocus ROI bottom-right corner vertical position in pixels.
    /// Autofocus ROI is rectangle.
    int afRoiY1{0};
    /// Zoom speed. Lens controller should have zoom speed range from 0 to
    /// 100% of max hardware zoom speed (parameter ZOOM_HW_MAX_SPEED). If the
    /// user sets a new parameter value of the ZOOM_HW_SPEED the parameter
    /// ZOOM_SPEED must be updated automatically. Formula for calculating speed:
    /// ZOOM_SPEED = ( ZOOM_HW_SPEED / ZOOM_HW_MAX_SPEED) * 100.
    int zoomSpeed{50};
    /// Zoom hardware speed. Value depends on implementation and lens hardware.
    /// The value of the parameters must be <= ZOOM_HW_MAX_SPEED parameter.
    /// If the user sets a new parameter value of the ZOOM_SPEED parameter
    /// the parameter ZOOM_HW_SPEED must be updated automatically. Formula for
    /// calculating hardware speed:
    /// ZOOM_HW_SPEED = ( ZOOM_SPEED / 100 ) * ZOOM_HW_MAX_SPEED.
    int zoomHwSpeed{50};
    /// Maximum zoom hardware speed. Value depends on implementation. If user
    /// sets new ZOOM_HW_MAX_SPEED value the parameters ZOOM_SPEED must be
```

```
    /// updated automatically. If new value of ZOOM_HW_MAX_SPEED parameter will
    /// be less than ZOOM_HW_SPEED the parameter ZOOM_HW_SPEED must be reduced
    /// automatically.
    int zoomHwMaxSpeed{50};
    /// Focus speed. Lens controller should have focus speed range from 0 to
    /// 100% of max hardware focus speed (parameter FOCUS_HW_MAX_SPEED). If the
    /// user sets a new parameter value of the FOCUS_HW_SPEED the parameter
    /// FOCUS_SPEED must be updated automatically. Formula for calculating
    /// speed: FOCUS_SPEED = ( FOCUS_HW_SPEED / FOCUS_HW_MAX_SPEED) * 100.
    int focusSpeed{50};
    /// Focus hardware speed. Value depends on on implementation and lens
    /// hardware. The value of the parameters must be <= FOCUS_HW_MAX_SPEED
    /// parameter. If the user sets a new parameter value of the FOCUS_SPEED
    /// parameter the parameter FOCUS_HW_SPEED must be updated automatically.
    /// Formula for calculating hardware speed:
    /// FOCUS_HW_SPEED = ( FOCUS_SPEED / 100 ) * FOCUS_HW_MAX_SPEED.
    int focusHwSpeed{50};
    /// Maximum focus hardware speed. Value depends on implementation. If user
    /// sets new FOCUS_HW_MAX_SPEED value the parameters FOCUS_SPEED and
    /// FOCUS_HW_SPEED must be updated by lens controller automatically.
    /// If new value of FOCUS_HW_MAX_SPEED parameter will be less than
    /// FOCUS_HW_SPEED the parameter FOCUS_HW_SPEED must be reduced
    /// automatically.
    int focusHwMaxSpeed{50};
    /// Iris speed. Lens controller should have iris speed range from 0 to 100%
    /// of max hardware iris speed (parameter IRIS_HW_MAX_SPEED). If the user
    /// sets a new parameter value of the IRIS_HW_SPEED the parameter IRIS_SPEED
    /// must be updated automatically. Formula for calculating speed:
    /// IRIS_SPEED = ( IRIS_HW_SPEED / IRIS_HW_MAX_SPEED) * 100.
    int irisSpeed{50};
    /// Iris hardware speed. Value depends on implementation and les hardware.
    /// The value of the parameters must be <= IRIS_HW_MAX_SPEED parameter.
    /// If the user sets a new parameter value of the IRIS_SPEED parameter the
    /// parameter IRIS_HW_SPEED must be updated automatically. Formula for
    /// calculating hardware speed:
    /// IRIS_HW_SPEED = ( IRIS_SPEED / 100 ) * IRIS_HW_MAX_SPEED.
    int irisHwSpeed{50};
    /// Maximum iris hardware speed. Value depends on implementation. If user
    /// sets new IRIS_HW_MAX_SPEED value the parameters IRIS_SPEED and
    /// IRIS_HW_SPEED must be updated automatically. If new value of
    /// IRIS_HW_MAX_SPEED parameter will be less than IRIS_HW_SPEED the
    /// parameter IRIS_HW_SPEED must be reduced automatically.
    int irisHwMaxSpeed{50};
    /// Zoom hardware tele limit. Value depends on implementation and lens
    /// hardware. Lens controller should control zoom position. Lens controller
    /// should stop zoom moving if hardware zoom position will be our of limits.
    int zoomHwTeleLimit{65535};
    /// Zoom hardware wide limit. Value depends on implementation and lens
    /// hardware. Lens controller should control zoom position. Lens controller
    /// should stop zoom moving if hardware zoom position will be our of limits.
    int zoomHwWideLimit{0};
    /// Focus hardware far limit. Value depends on on implementation and lens
    /// hardware. Lens controller should control focus position. Lens controller
    /// should stop focus moving if hardware focus position will be our of
    /// limits.
    int focusHwFarLimit{65535};
```

```
    /// Focus hardware near limit. Value depends on implementation and lens
    /// hardware. Lens controller should control focus position. Lens controller
    /// should stop focus moving if hardware focus position will be our of
    /// limits.
    int focusHwNearLimit{0};
    /// Iris hardware open limit. Value depends on implementation and lens
    /// hardware. Lens controller should control iris position. Lens controller
    /// should stop iris moving if hardware iris position will be our of limits.
    int irisHwOpenLimit{65535};
    /// Iris hardware close limit. Value depends on implementation and lens
    /// hardware. Lens controller should control iris position. Lens controller
    /// should stop iris moving if hardware iris position will be our of limits.
    int irisHwCloseLimit{0};
    /// Focus factor if it was calculated. If not calculated must be -1.
    /// Value depends on particular lens controller.
    float focusFactor{0.0f};
    /// Lens connection status. Connection status shows if the lens controller
    /// has data exchange with lens equipment. For example, if lens has serial
    /// port lens controller connects to serial port (opens serial port file
    /// in OS) but lens can be not active (no power). In this case connection
    /// status shows that lens controller doesn't have data exchange with lens
    /// equipment (methos will return 0). It lens controller has data exchange
    /// with lens equipment the method will return 1. If lens controller not
    /// initialize the connection status always FALSE. Value:
    /// false - not connected. true - connected.
    bool isConnected{false};
    /// Focus hardware speed in autofocus mode. Value depends on implementation
    /// and lens hardware.
    int afHwSpeed{50};
    /// Timeout for automatic refocus in seconds. Value: 0 - no automatic
    /// refocus, 100000 - maximum value.
    float focusFactorThreshold{0.0f};
    /// Timeout for automatic refocus in seconds. Value:
    /// 0 - no automatic refocus, 100000 - maximum value.
    int refocusTimeoutSec{0};
    /// Flag about active autofocus algorithm. Value: 0 - autofocus not working,
    /// 1 - working.
    bool afIsActive{false};
    /// Iris mode. Value depends on implementation but it is recommended to keep
    /// default values: 0 - manual iris control, 1 - auto iris control.
    int irisMode{0};
    /// ROI width (pixels) for autofocus algorithm when lens controller detects
    /// ROI position automatically. Value: from 8 to (video frame width -
    /// AUTO_AF_ROI_BORDER * 2).
    int autoAfRoiWidth{150};
    /// ROI height (pixels) for autofocus algorithm when lens controller detects
    /// ROI position automatically. Value: from 8 to (video frame width -
    /// AUTO_AF_ROI_BORDER * 2).
    int autoAfRoiHeight{150};
    /// Video frame border size (along vertical and horizontal axes).
    /// Value: border size from 0 to video
    /// frame min(video frame width/height) / 2.
    int autoAfRoiBorder{100};
    /// AF ROI mode (write/read). Value: 0 - Manual position, 1 - Auto position.
    int afRoiMode{0};
    /// Lens extender mode. Value depends on implementation but it is
```

```
    /// recommended to keep default values: 0 - no extender, 1 - x2 extender.
    int extenderMode{0};
    /// Lens stabilization mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - no stabilization,
    /// 1 - stabilization.
    int stabiliserMode{0};
    /// Autofocus range. Value depends on implementation.
    int afRange{0};
    /// Current horizontal Field of view, degree. Field of view calculated by
    /// lens controller according to initial params or by reading directly from
    /// lens hardware.
    float xFovDeg{1.0f};
    /// Current vertical Field of view, degree. Field of view calculated by lens
    /// controller according to initial params or by reading directly from lens
    /// hardware.
    float yFovDeg{1.0f};
    /// Logging mode. Values: 0 - Disable, 1 - Only file, 2 - Only terminal,
    /// 3 - File and terminal.
    int logMode{0};
    /// Lens temperature, degree (read only).
    float temperature{0.0f};
    /// Lens controller initialization status. Open status shows if the lens
    /// controller initialized or not but doesn't show if lens controller has
    /// communication with lens equipment. For example, if lens has serial port
    /// lens controller connects to serial port (opens serial port file in OS)
    /// but lens can be not active (no power). In this case open status just
    /// shows that lens controller has opened serial port.
    /// Values: false - not open (not initialized), true - open (initialized).
    bool isOpen{false};
    /// Lens type. Value depends on implementation. Type allows to lens
    /// initialize necessary parameters for particular lens hardware.
    int type{0};
    /// Lens custom parameter. Value depends on particular lens controller.
    /// Custom parameters used when particular lens equipment has specific
    /// unusual parameter.
    float custom1{0.0f};
    /// Lens custom parameter. Value depends on particular lens controller.
    /// Custom parameters used when particular lens equipment has specific
    /// unusual parameter.
    float custom2{0.0f};
    /// Lens custom parameter. Value depends on particular lens controller.
    /// Custom parameters used when particular lens equipment has specific
    /// unusual parameter.
    float custom3{0.0f};
    /// List of points to calculate fiend of view. Lens controller should
    /// calculate FOV table according to given list f points using
    /// approximation.
    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);
};
```

Table 4 - LensParams class fields description is equivalent to **LensParam** enum description.

Field	type	Description
initString	string	Initialization string contains full serial port name, baudrate, timeout separated by ";". Example: "/dev/ttyUSB0;9600;50". The same string used in openLens(...) method.
zoomPos	int	Zoom position **0 (full wide) - 65535 (full tele)**.
zoomHwPos	int	Hardware zoom position **0 (full wide) - 65535 (full tele)** by default.
focusPos	int	Focus position **0 (full near) - 65535 (full far)**.
focusHwPos	int	Hardware focus position **0 (full near) - 65535 (full far)** by default.
irisPos	int	Iris position **0 (full close) - 65535 (full open)**.
irisHwPos	int	Hardware iris position **0 (full close) - 65535 (full open)** by default.
focusMode	int	Focus mode: **0** - Manual focus control.
filterMode	int	Filter mode: **0** - VC filter auxiliary (out) * 2 **1** - VC filter auxiliary (in) * 2
afRoiX0	int	**Not supported** by Ylc class.
afRoiY0	int	**Not supported** by Ylc class.
afRoiX1	int	**Not supported** by Ylc class.
afRoiY1	int	**Not supported** by Ylc class.
zoomSpeed	int	**Not supported** by Ylc class.
zoomHwSpeed	int	**Not supported** by Ylc class.
zoomHwMaxSpeed	int	**Not supported** by Ylc class.

Field	type	Description
focusSpeed	int	**Not supported** by Ylc class.
focusHwSpeed	int	**Not supported** by Ylc class.
focusHwMaxSpeed	int	**Not supported** by Ylc class.
irisSpeed	int	**Not supported** by Ylc class.
irisHwSpeed	int	**Not supported** by Ylc class.
irisHwMaxSpeed	int	**Not supported** by Ylc class.
zoomHwTeleLimit	int	Zoom hardware tele limit. Default value 65535.
zoomHwWideLimit	int	Zoom hardware wide limit. Default value 0.
focusHwFarLimit	int	Focus hardware far limit. Default value 65535.
focusHwNearLimit	int	Focus hardware near limit. Default value 0.
irisHwOpenLimit	int	Iris hardware open limit. Default value 65535.
irisHwCloseLimit	int	Iris hardware close limit. Default value 0.
focusFactor	float	**Not supported** by Ylc class.
isConnected	bool	Lens connection status. Connection status shows if the lens controller has data exchange with lens equipment. Values: **0** - no response from lens. **1** - connected.
afHwSpeed	int	**Not supported** by Ylc class.
focusFactorThreshold	float	**Not supported** by Ylc class.
refocusTimeoutSec	int	**Not supported** by Ylc class.
afIsActive	bool	**Not supported** by Ylc class.
irisMode	int	Iris mode: **0** - Auto iris off. **1** - Auto iris on.
autoAfRoiWidth	int	**Not supported** by Ylc class.
autoAfRoiHeight	int	**Not supported** by Ylc class.
autoAfRoiBorder	int	**Not supported** by Ylc class.
afRoiMode	int	**Not supported** by Ylc class.
extenderMode	int	Extender mode: 0 - X2 extender lens in. 1 - X2 extender lens out.
stabiliserMode	int	**Not supported** by Ylc class.

Field	type	Description
afRange	int	**Not supported** by Ylc class.
xFovDeg	float	**Not supported** by Ylc class.
yFovDeg	float	**Not supported** by Ylc class.
logMode	int	Logging mode. Values: **0** - Disable. **1** - Only file. **2** - Only terminal. **3** - File and terminal.
temperature	float	**Not supported** by Ylc class.
isOpen	bool	Lens controller initialization status. Open status shows if the lens controller initialized or not but doesn't show if lens controller has communication with lens equipment. Values: 0 - not open (not initialized), 1 - open (initialized).
type	int	**Not supported** by Ylc class.
custom1	float	**Not supported** by Ylc class.
custom2	float	**Not supported** by Ylc class.
custom3	float	**Not supported** by Ylc class.
fovPoints	FovPoint	List of points to calculate fiend of view. Lens controller should calculate FOV table according to given list f points using approximation. Each point includes (**FovPoint** class): - **hwZoomPos** - hardware zoom position. - **xFovDeg** - horizontal FOV, degree for hwZoomPos. - **yFovDeg** - vertical FOV, degree for hwZoomPos.

None: *LensParams class fiellds listed in above reflect params set/get by methods setParam(...) and getParam(...).*

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.

Parameter	Value
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.

Parameter	Value
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 **Ylc** library:

```
cd Ylc
mkdir build
cd build
cmake ..
make
```

If you want to connect **Ylc** library to your CMake project as source code, you can do the following. For example, if your repository has structure:

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

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

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

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

```
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(${PARENT}_SUBMODULE_YLC                                 ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_YLC)
    SET(${PARENT}_YLC                                       ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_YLC_TEST                                  OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_YLC_EXAMPLE                               OFF CACHE BOOL "" FORCE)
endif()

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

File **3rdparty/CMakeLists.txt** adds folder **Ylc** to your project and excludes test applications and examples from compiling. The new structure of your repository will be:

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

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

```
add_subdirectory(3rdparty)
```

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

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

Done!

Simple example

A simple application shows how to use the **Ylc** library.

```
#include <chrono>
#include <iostream>
#include <string>
#include <thread>
#include "Ylc.h"

int main(void)
{
    // Init lens controller.
    cr::lens::Lens* lc = new cr::lens::Ylc();
    if (!lc->openLens("/dev/ttyUSB0"))
        return -1;

    // Get lens zoom position
    std:: cout<< "Zoom pos: " <<
    lc->getParam(cr::lens::LensParam::ZOOM_POS) << std::endl;

    // Go to zoom position 65535 (Full tele).
    lc->executeCommand(cr::lens::LensCommand::ZOOM_TO_POS, 65535);

    // Show zoom movement (changing position).
    for (int i = 0; i < 50; ++i)
    {
        std:: cout<< "Zoom pos: " <<
        lc->getParam(cr::lens::LensParam::ZOOM_POS) << std::endl;
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
    }
    return 1;
}
```