



# Lens interface C++ library

---

v4.0.3

---

## Table of contents

---

- [Overview](#)
- [Versions](#)
- [Lens interface 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](#)
  - [encodeSetParamCommand method](#)
  - [encodeCommand method](#)
  - [decodeCommand method](#)
- [Data structures](#)
  - [LensCommand enum](#)
  - [LensParam enum](#)
- [LensParams class description](#)
  - [Class declaration](#)
  - [Serialize lens params](#)
  - [Deserialize lens params](#)
  - [Read params from JSON file and write to JSON file](#)
- [Build and connect to your project](#)

# Overview

**Lens** C++ library provides standard interface as well defines data structures and rules for different lens controllers. **Lens** interface class doesn't do anything, just provides interface, data structures and service functions to encode/decode parameter. Different lens controller classes inherit interface from **Lens** C++ class. **Lens.h** file contains list of data structures (**LensCommand** enum, **LensParam** enum and **LensParams** class) and **Lens** class declaration. **LensCommand** enum contains IDs of commands supported by **Lens** class. **LensParam** enum contains IDs of params supported by **Lens** class. **LensParams** class contains fields for lens parameters values and provides methods to encode/decode and read/write lens parameters from JSON file. All lens controllers should include params and commands listed in **Lens.h** file. Lens interface class depends on **Frame** class (describes video frame and video frame data structures, necessary for autofocus functions) and **ConfigReader** library (provides methods to read/write JSON config files).

## Versions

**Table 1** - Library versions.

Version	Release date	What's new
1.0.0	21.10.2022	First version
2.0.0	17.03.2023	<ul style="list-style-type: none"><li>- Subrepository <b>LensControllerDataStructures</b> excluded.</li><li>- Data structures moved to <b>Lens.h</b> file.</li><li>- New parameters IDs added.</li><li>- Description updated.</li></ul>
3.0.0	23.04.2023	<ul style="list-style-type: none"><li>- Methods signatures changed.</li></ul>
3.1.0	08.05.2023	<ul style="list-style-type: none"><li>- Added new parameters.</li></ul>
3.2.0	10.05.2023	<ul style="list-style-type: none"><li>- Parameters list changed.</li></ul>
4.0.0	18.06.2023	<ul style="list-style-type: none"><li>- Added <b>LensParams</b> class which contains lens controller parameters.</li><li>- Added new methods to encode/decode lens control command.</li><li>- Added new lens parameters.</li><li>- Added tests.</li></ul>
4.0.1	28.06.2023	<ul style="list-style-type: none"><li>- ConfigReader submodule updated.</li><li>- Frame submodule updated.</li></ul>
4.0.2	29.06.2023	<ul style="list-style-type: none"><li>- Documentation updated.</li></ul>
4.0.3	01.07.2023	<ul style="list-style-type: none"><li>- Documentation updated.</li><li>- Lens class comments in source code updated.</li></ul>

# Lens interface class description

---

## Class declaration

---

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

```
namespace cr
{
    namespace lens
    {
        /**
         * @brief Lens controller interface class.
         */
        class Lens
        {
        public:
            /**
             * @brief Get lens class version.
             * @return Lens class version string in format "Major.Minor.Patch".
             */
            static std::string getVersion();
            /**
             * @brief Open lens controller. Can be used instead initLens(...) method.
             * @param initString Init string. Format depends on lens controller.
             * @return TRUE if the lens controller is init or FALSE.
             */
            virtual bool openLens(std::string initString) = 0;
            /**
             * @brief Init lens controller by structure. Can be used instead
             * openLens(...) method.
             * @param initString Init string. Format depends on lens controller.
             * @return TRUE if the lens controller is init or FALSE.
             */
            virtual bool initLens(LensParams& params) = 0;
            /**
             * @brief Close connection.
             */
            virtual void closeLens() = 0;
            /**
             * @brief Get lens open status.
             * @return TRUE if the lens is open or FALSE.
             */
            virtual bool isLensOpen() = 0;
            /**
             * @brief Get lens connection status. Lens can be open but no response from
             * lens hardware.
             * @return TRUE if the lens is open or FALSE.
             */
            virtual bool isLensConnected() = 0;
            /**
             * @brief Set the lens controller param.
```

```

    * @param id Param ID.
    * @param value Param value.
    * @return TRUE if the property set or FALSE.
    */
virtual bool setParam(LensParam id, float value) = 0;
/**
    * @brief Get the lens controller param.
    * @param id Param ID.
    * @return float Param value or -1 if the param not exists.
    */
virtual float getParam(LensParam id) = 0;
/**
    * @brief Get the lens controller paramw.
    * @param id Param ID.
    * @return Lens params structure.
    */
virtual LensParams getParams() = 0;
/**
    * @brief Execute command.
    * @param id Command ID.
    * @param arg Command argument.
    * @return TRUE if the command executed or FALSE.
    */
virtual bool executeCommand(LensCommand id, float arg = 0) = 0;
/**
    * @brief Add video frame for auto focus purposes. Some lens controllers
    * may not support this functions.
    * @param frame Video frame object.
    */
virtual void addVideoFrame(cr::video::Frame& frame) = 0;
/**
    * @brief Encode set param command.
    * @param data Pointer to data buffer. Must have size >= 11.
    * @param size Size of encoded data.
    * @param id Lens parameter id.
    * @param value Lens parameter value.
    */
static void encodeSetParamCommand(
    uint8_t* data, int& size, LensParam id, float value);
/**
    * @brief Encode command.
    * @param data Pointer to data buffer. Must have size >= 11.
    * @param size Size of encoded data.
    * @param id Lens command ID.
    * @param arg Lens command argument.
    */
static void encodeCommand(
    uint8_t* data, int& size, LensCommand id, float arg = 0.0f);
/**
    * @brief Decode command.
    * @param data Pointer to command data.
    * @param size Size of data.
    * @param paramId Output command ID.
    * @param commandId Output command ID.
    * @param value Param or command value.
    * @return 0 - command decoded, 1 - set param command decoded, -1 - error.

```

```

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

```

## getVersion method

**getVersion()** method returns string of current class version. Particular lens controller can have it's own **getVersion()** method. Method declaration:

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

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

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

Console output:

```
Lens class version: 4.0.2
```

## openLens method

**openLens(...)** method designed to initialize lens controller. This method can be used instead of **initLens(...)** method. Method declaration:

```
virtual bool openLens(std::string initString) = 0;
```

Parameter	Value
initString	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: <b>"/dev/ttyUSB0;9600;100"</b> ("/dev/ttyUSB0" - serial port name, "9600" - baudrate, "100" - serial port read timeout).

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

## initLens method

---

**initLens(...)** method designed to initialize lens controller by list of parameters. This method can be used instead of **openLens(...)** method (**LensParams** class includes **initString**) when you need initialize lens controller with not default parameters. Method declaration:

```
virtual bool initLens(LensParams& params) = 0;
```

Parameter	Value
params	Parameters ( <b>LensParams</b> class). LensParams class includes initString wich used in <b>openLens(...)</b> method. See description of <b>LensParams</b> class.

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

## closeLens method

---

**closeLens()** method designed to close connection to lens. Method declaration:

```
virtual void closeLens() = 0;
```

## isLensOpen method

---

**isLensOpen()** method designed to obtain lens initialization status. Open status shows if the lens controller initialized 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. Method declaration:

```
virtual bool isLensOpen() = 0;
```

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

## isLensConnected method

---

**isLensConnected()** method designed to obtain 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 (method will return FALSE). If lens controller has data exchange with lens equipment the method will return TRUE. If lens controller not initialize the connection status always FALSE. Method declaration:

```
virtual bool isLensConnected() = 0;
```

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

## setParam method

**setParam(...)** method designed to set new lens parameters value. Method declaration:

```
virtual bool setParam(LensParam id, float value) = 0;
```

Parameter	Description
id	Lens parameter ID (see description of <b>LensParam</b> enum).
value	Lens parameter value. Value depends on parameter ID (see description of <b>LensParam</b> enum).

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

## getParam method

**getParam(...)** method designed to obtain lens parameter value. Method declaration:

```
virtual float getParam(LensParam id) = 0;
```

Parameter	Description
id	Lens parameter ID (see description of <b>LensParam</b> enum).

**Returns:** parameter value or -1 if the parameter doesn't exist (not supported) in particular lens controller.

## getParams method

**getParams(...)** method designed to obtain lens parameters. Method declaration:

```
virtual LensParams getParams() = 0;
```

**Returns:** **LensParams** class which contains all current lens params.

## executeCommand method

**executeCommand(...)** method designed to execute lens command. Method declaration:

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

Parameter	Description
id	Lens command ID (see description of <b>LensCommand</b> enum).
arg	Lens command argument. Value depends on command ID (see description of <b>LensCommand</b> enum).

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

## addVideoFrame method

**addVideoFrame(...)** method designed to copy video frame data to lens controller to perform autofocus algorithm. Particular lens controller may not support autofocus algorithms. To perform autofocus lens controller calculates focus factor in autofocus ROI (focus factor can be obtained with methos **getParam(...)** and parameters **FOCUS\_FACTOR**). To calculate focus factor lens controller needs video frame. If particular lens controller supports autofocus algorithms the method **addVideoFrame(...)** should be called for each captured video frame. Method declaration:

```
virtual void addVideoFrame(cr::video::Frame& frame) = 0;
```

Parameter	Description
frame	Video frame object (see <b>Frame</b> class description).

**Returns:** TRUE is the video frame accepted or FALSE if not. **Note:** if particular lens controller doesn't support this function it should return **TRUE** (recommended).

## encodeSetParamCommand method

**encodeSetParamCommand(...)** static method 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 <b>LensParam</b> enum.
value	Parameter value.



**SET\_PARAM** command format:

Byte	Value	Description
0	0x01	SET_PARAM command header value.
1	0x04	Major version of Lens class.
2	0x00	Minor version of Lens class.
3	id	Parameter ID <b>int32_t</b> in Little-endian format.
4	id	Parameter ID <b>int32_t</b> in Little-endian format.
5	id	Parameter ID <b>int32_t</b> in Little-endian format.
6	id	Parameter ID <b>int32_t</b> in Little-endian format.
7	value	Parameter value <b>float</b> in Little-endian format.
8	value	Parameter value <b>float</b> in Little-endian format.
9	value	Parameter value <b>float</b> in Little-endian format.
10	value	Parameter value <b>float</b> 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

**encodeCommand(...)** static method 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 $\geq 11$ .
size	Size of encoded data. Will be 11 bytes.
id	Command ID according to <b>LensCommand</b> enum.
arg	Command argument value (value depends on command ID).

**COMMAND** format:

Byte	Value	Description
0	0x00	SET_PARAM command header value.
1	0x04	Major version of Lens class.
2	0x00	Minor version of Lens class.
3	id	Command ID <b>int32_t</b> in Little-endian format.
4	id	Command ID <b>int32_t</b> in Little-endian format.
5	id	Command ID <b>int32_t</b> in Little-endian format.
6	id	Command ID <b>int32_t</b> in Little-endian format.
7	arg	Command argument value <b>float</b> in Little-endian format.
8	arg	Command argument value <b>float</b> in Little-endian format.
9	arg	Command argument value <b>float</b> in Little-endian format.
10	arg	Command argument value <b>float</b> 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

**decodeCommand(...)** static method designed to decode command on lens controller side. To 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 <b>LensParam</b> enum. After decoding SET_PARAM command the method will return parameter ID.
commandId	Lens command ID according to <b>LensCommand</b> enum. After decoding COMMAND the method will return command ID.
value	Len 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

**Lens.h** file defines IDs for parameters (**LensParam** enum), IDs for commands (**LensCommand** enum).

## 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 particular lens controller.

Command	Description
ZOOM_TELE	Move zoom tele (in). Command doesn't have arguments. User should be able to set zoom movement speed via lens parameters.

Command	Description
ZOOM_WIDE	Move zoom wide (out). Command doesn't have arguments. User should be able to set zoom movement speed via lens parameters.
ZOOM_TO_POS	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_STOP	Stop zoom moving including stop zoom to position command.
FOCUS_FAR	Move focus far. Command doesn't have arguments. User should be able to set focus movement speed via lens parameters.
FOCUS_NEAR	Move focus near. Command doesn't have arguments. User should be able to set focus movement speed via lens parameters.
FOCUS_TO_POS	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_STOP	Stop focus moving including stop focus to position command.
IRIS_OPEN	Move iris open. Command doesn't have arguments. User should be able to set iris movement speed via lens parameters.
IRIS_CLOSE	Move iris close. Command doesn't have arguments. User should be able to set iris movement speed via lens parameters.
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.

Command	Description
DETECT_HW_RANGES	Detect zoom and focus hardware ranges. After execution this command the lens controller should automatically set at least parameters ( <b>LensParam</b> enum): ZOOM_HW_TELE_LIMIT, ZOOM_HW_WIDE_LIMIT, FOCUS_HW_FAR_LIMIT and FOCUS_HW_NEAR_LIMIT.

## 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  $\leq$  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  $\leq$  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  $\leq$  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 particular lens controller.

Parameter	Access	Description
ZOOM_POS	read / write	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_HW_POS	read / write	Hardware zoom position. Parameter value depends on implementation and lens hardware.

Parameter	Access	Description
FOCUS_POS	read / write	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_HW_POS	read / write	Hardware focus position. Parameter value depends on implementation and lens hardware.
IRIS_POS	read / write	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_HW_POS	read / write	Hardware iris position. Parameter value depends on particular lens controller.
FOCUS_MODE	read / write	Focus mode. Parameter value depends on implementation but it is recommended to keep default values: 0 - Manual focus control, 1 - Auto focus control.
FILTER_MODE	read / write	Filter mode. Parameter value depends on implementation but it is recommended to keep default values: 0 - Filter on, 1 - Filter off.
AF_ROI_X0	read / write	Autofocus ROI top-left corner horizontal position in pixels. Autofocus ROI is rectangle.
AF_ROI_Y0	read / write	Autofocus ROI top-left corner vertical position in pixels. Autofocus ROI is rectangle.
AF_ROI_X1	read / write	Autofocus ROI bottom-right corner horizontal position in pixels. Autofocus ROI is rectangle.
AF_ROI_Y1	read / write	Autofocus ROI bottom-right corner vertical position in pixels. Autofocus ROI is rectangle.

Parameter	Access	Description
ZOOM_SPEED	read / write	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_HW_SPEED	read / write	Zoom hardware speed. Value depends on implementation and lens hardware. The value of the parameters must be $\leq$ 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_MAX_SPEED	read / write	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.
FOCUS_SPEED	read / write	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: $FOCUS\_SPEED = (FOCUS\_HW\_SPEED / FOCUS\_HW\_MAX\_SPEED) * 100$ .

Parameter	Access	Description
FOCUS_HW_SPEED	read / write	Focus hardware speed. Value depends on on implementation and lens hardware. The value of the parameters must be $\leq$ 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: $\text{FOCUS\_HW\_SPEED} = ( \text{FOCUS\_SPEED} / 100 ) * \text{FOCUS\_HW\_MAX\_SPEED}$ .
FOCUS_HW_MAX_SPEED	read / write	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.
IRIS_SPEED	read / write	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: $\text{IRIS\_SPEED} = ( \text{IRIS\_HW\_SPEED} / \text{IRIS\_HW\_MAX\_SPEED} ) * 100$ .
IRIS_HW_SPEED	read / write	Iris hardware speed. Value depends on implementation and les hardware. The value of the parameters must be $\leq$ 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: $\text{IRIS\_HW\_SPEED} = ( \text{IRIS\_SPEED} / 100 ) * \text{IRIS\_HW\_MAX\_SPEED}$ .
IRIS_HW_MAX_SPEED	read / write	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.

Parameter	Access	Description
ZOOM_HW_TELE_LIMIT	read / write	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 out of limits.
ZOOM_HW_WIDE_LIMIT	read / write	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 out of limits.
FOCUS_HW_FAR_LIMIT	read / write	Focus hardware far 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 out of limits.
FOCUS_HW_NEAR_LIMIT	read / write	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 out of limits.
IRIS_HW_OPEN_LIMIT	read / write	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 out of limits.
IRIS_HW_CLOSE_LIMIT	read / write	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 out of limits.
FOCUS_FACTOR	read only	Focus factor if it was calculated. If not calculated must be -1. Value depends on particular lens controller.

Parameter	Access	Description
IS_CONNECTED	read only	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.
FOCUS_HW_AF_SPEED	read / write	Focus hardware speed in autofocus mode. Value depends on implementation and lens hardware.
FOCUS_FACTOR_THRESHOLD	read / write	Threshold for changes of focus factor to start refocus. Value: 0% - no check, 100% - changing x2.
REFOCUS_TIMEOUT_SEC	read / write	Timeout for automatic refocus in seconds. Value: 0 - no automatic refocus, 100000 - maximum value.
AF_IS_ACTIVE	read only	Flag about active autofocus algorithm. Value: 0 - autofocus not working, 1 - working.
IRIS_MODE	read / write	Iris mode. Value depends on implementation but it is recommended to keep default values: 0 - manual iris control, 1 - auto iris control.
AUTO_AF_ROI_WIDTH	read / write	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_HEIGHT	read / write	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).
AUTO_AF_ROI_BORDER	read / write	Video frame border size (along vertical and horizontal axes). Value: border size from 0 to video frame min(video frame width/height) / 2.
AF_ROI_MODE	read / write	AF ROI mode (write/read). Value: 0 - Manual position, 1 - Auto position.

Parameter	Access	Description
EXTENDER_MODE	read / write	Lens extender mode. Value depends on implementation but it is recommended to keep default values: 0 - no extender, 1 - x2 extender.
STABILIZER_MODE	read / write	Lens stabilization mode. Value depends on implementation but it is recommended to keep default values: 0 - no stabilization, 1 - stabilization.
AF_RANGE	read / write	Autofocus range. Value depends on implementation.
X_FOV_DEG	read only	Current horizontal 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	read only	Current vertical Field of view, degree. Field of view calculated by lens controller according to initial params or by reading directly from lens hardware.
LOG_MODE	read / write	Logging mode. Values: 0 - Disable, 1 - Only file, 2 - Only terminal, 3 - File and terminal.
TEMPERATURE	read only	Lens temperature, degree.
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. 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).
TYPE	read / write	Lens type. Value depends on implementation. Type allows to lens initialize necessary parameters for particular lens hardware.
CUSTOM_1	read / write	Lens custom parameter. Value depends on particular lens controller. Custom parameters used when particular lens equipment has specific unusual parameter.



Parameter	Access	Description
CUSTOM_2	read / write	Lens custom parameter. Value depends on particular lens controller. Custom parameters used when particular lens equipment has specific unusual parameter.
CUSTOM_3	read / write	Lens custom parameter. Value depends on particular lens controller. Custom parameters used when particular lens equipment has specific unusual parameter.

## LensParams class description

**LensParams** class used for lens controller initialization (**initLens(...)** method) or to get all actual params (**getParams()** method). Also **LensParams** provide structure to write/read params from JSON files (**JSON\_READABLE** macro) and provide 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);

    /**
     * @brief operator =
     * @param src Source object.
     * @return FovPoint object.
     */
    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 implementation.
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  $\leq$  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  $\leq$  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  $\leq$  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 out 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 out of limits.
int zoomHwWideLimit{0};
/// Focus hardware far 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 out 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 out 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 out 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 out 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 (method will return 0). If 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;
/// Lens custom parameter. Value depends on particular lens controller.
/// Custom parameters used when particular lens equipment has specific
/// unusual parameter.
float custom2;
/// Lens custom parameter. Value depends on particular lens controller.

```

```

    /// Custom parameters used when particular lens equipment has specific
    /// unusual parameter.
    float custom3;
    /// 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);

    /**
     * @brief operator =
     * @param src Source object.
     * @return LensParams obect.
     */
    LensParams& operator= (const LensParams& src);
    /**
     * @brief Encode params. The method doesn't encode initString and fovPoints.
     * @param data Pointer to data buffer.
     * @param size Size of data.
     */
    void encode(uint8_t* data, int& size);
    /**
     * @brief Decode params. The method doesn't decode initString and fovPoints.
     * @param data Pointer to data.
     * @return TRUE is params decoded or FALSE if not.
     */
    bool decode(uint8_t* data);
};

```

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

Field	type	Description
initString	string	<p>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:</p> <p><b>"/dev/ttyUSB0;9600;100"</b> ("/dev/ttyUSB0" - serial port name, "9600" - baudrate, "100" - serial port read timeout).</p>

Field	type	Description
zoomPos	int	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.
zoomHwPos	int	Hardware zoom position. Parameter value depends on implementation and lens hardware.
focusPos	int	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.
focusHwPos	int	Hardware focus position. Parameter value depends on implementation and lens hardware.
irisPos	int	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.
irisHwPos	int	Hardware iris position. Parameter value depends on implementation.
focusMode	int	Focus mode. Parameter value depends on implementation but it is recommended to keep default values: 0 - Manual focus control, 1 - Auto focus control.

Field	type	Description
filterMode	int	Filter mode. Parameter value depends on implementation but it is recommended to keep default values: 0 - Filter on, 1 - Filter off.
afRoiX0	int	Autofocus ROI top-left corner horizontal position in pixels. Autofocus ROI is rectangle.
afRoiY0	int	Autofocus ROI top-left corner vertical position in pixels. Autofocus ROI is rectangle.
afRoiX1	int	Autofocus ROI bottom-right corner horizontal position in pixels. Autofocus ROI is rectangle.
afRoiY1	int	Autofocus ROI bottom-right corner vertical position in pixels. Autofocus ROI is rectangle.
zoomSpeed	int	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: $\text{ZOOM\_SPEED} = (\text{ZOOM\_HW\_SPEED} / \text{ZOOM\_HW\_MAX\_SPEED}) * 100.$
zoomHwSpeed	int	Zoom hardware speed. Value depends on implementation and lens hardware. The value of the parameters must be $\leq$ 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: $\text{ZOOM\_HW\_SPEED} = (\text{ZOOM\_SPEED} / 100) * \text{ZOOM\_HW\_MAX\_SPEED}.$
zoomHwMaxSpeed	int	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.
focusSpeed	int	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: $\text{FOCUS\_SPEED} = (\text{FOCUS\_HW\_SPEED} / \text{FOCUS\_HW\_MAX\_SPEED}) * 100.$



Field	type	Description
focusHwSpeed	int	Focus hardware speed. Value depends on on implementation and lens hardware. The value of the parameters must be $\leq$ 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: $\text{FOCUS\_HW\_SPEED} = ( \text{FOCUS\_SPEED} / 100 ) * \text{FOCUS\_HW\_MAX\_SPEED}.$
focusHwMaxSpeed	int	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.
irisSpeed	int	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: $\text{IRIS\_SPEED} = ( \text{IRIS\_HW\_SPEED} / \text{IRIS\_HW\_MAX\_SPEED} ) * 100.$
irisHwSpeed	int	Iris hardware speed. Value depends on implementation and les hardware. The value of the parameters must be $\leq$ 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: $\text{IRIS\_HW\_SPEED} = ( \text{IRIS\_SPEED} / 100 ) * \text{IRIS\_HW\_MAX\_SPEED}.$
irisHwMaxSpeed	int	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.

Field	type	Description
zoomHwTeleLimit	int	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 out of limits.
zoomHwWideLimit	int	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 out of limits.
focusHwFarLimit	int	Focus hardware far 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 out of limits.
focusHwNearLimit	int	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 out of limits.
irisHwOpenLimit	int	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 out of limits.
irisHwCloseLimit	int	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 out of limits.
focusFactor	float	Focus factor if it was calculated. If not calculated must be -1. Value depends on particular lens controller.

Field	type	Description
isConnected	bool	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.
afHwSpeed	int	Focus hardware speed in autofocus mode. Value depends on implementation and lens hardware.
focusFactorThreshold	float	Timeout for automatic refocus in seconds. Value: 0 - no automatic refocus, 100000 - maximum value.
refocusTimeoutSec	int	Timeout for automatic refocus in seconds. Value: 0 - no automatic refocus, 100000 - maximum value.
aflsActive	bool	Flag about active autofocus algorithm. Value: 0 - autofocus not working, 1 - working.
irisMode	int	Iris mode. Value depends on implementation but it is recommended to keep default values: 0 - manual iris control, 1 - auto iris control.
autoAfRoiWidth	int	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).
autoAfRoiHeight	int	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).
autoAfRoiBorder	int	Video frame border size (along vertical and horizontal axes). Value: border size from 0 to video frame min(video frame width/height) / 2.
afRoiMode	int	AF ROI mode (write/read). Value: 0 - Manual position, 1 - Auto position.
extenderMode	int	Lens extender mode. Value depends on implementation but it is recommended to keep default values: 0 - no extender, 1 - x2 extender.

Field	type	Description
stabiliserMode	int	Lens stabilization mode. Value depends on implementation but it is recommended to keep default values: 0 - no stabilization, 1 - stabilization.
afRange	int	Autofocus range. Value depends on implementation.
xFovDeg	float	Current horizontal Field of view, degree. Field of view calculated by lens controller according to initial params or by reading directly from lens hardware.
yFovDeg	float	Current vertical Field of view, degree. Field of view calculated by lens controller according to initial params or by reading directly from lens hardware.
logMode	int	Logging mode. Values: 0 - Disable, 1 - Only file, 2 - Only terminal, 3 - File and terminal.
temperature	float	Lens temperature, degree.
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. 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).
type	int	Lens type. Value depends on implementation. Type allows to lens initialize necessary parameters for particular lens hardware.
custom1	float	Lens custom parameter. Value depends on particular lens controller. Custom parameters used when particular lens equipment has specific unusual parameter.
custom2	float	Lens custom parameter. Value depends on particular lens controller. Custom parameters used when particular lens equipment has specific unusual parameter.
custom3	float	Lens custom parameter. Value depends on particular lens controller. Custom parameters used when particular lens equipment has specific unusual parameter.

Field	type	Description
fovPoints	FovPoint	<p>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 (<b>FovPoint</b> class):</p> <ul style="list-style-type: none"> <li>- <b>hwZoomPos</b> - hardware zoom position.</li> <li>- <b>xFovDeg</b> - horizontal FOV, degree for hwZoomPos.</li> <li>- <b>yFovDeg</b> - vertical FOV, degree for hwZoomPos.</li> </ul>

**None:** *LensParams* class fiellds listed in Table 4 **must** reflect params set/get by methods *setParam(...)* and *getParam(...)*.

## Serialize lens params

**LensParams** class provides method **encode(...)** to serialize lens params (fields of *LensParams* class, see Table 4). 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 declaration:

```
void encode(uint8_t* data, int& size);
```

Parameter	Value
data	Pointer to data buffer.
size	Size of encoded data.

Example:

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

## Deserialize lens params

**LensParams** class provides method **decode(...)** to deserialize lens params (fields of *LensParams* class, see Table 4). Deserialization of lens params necessary in case when you need to receive lens params via communication channels. Method doesn't decode fields: **initString** and **fovPoints**. Method declaration:

```
bool decode(uint8_t* data);
```

Parameter	Value
data	Pointer to data buffer.

**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, size);
cout << "Encoded data size: " << size << " bytes" << endl;

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

## Read params from JSON file and write 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": "dfhglsjiruhjfb",
    "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 **Lens** library:

```
git clone https://github.com/ConstantRobotics-Ltd/Lens.git  
cd Lens  
git submodule update --init --recursive  
mkdir build  
cd build  
cmake ..  
make
```

If you want connect **Lens** 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
```

You can add repository **Lens** as submodule by commands:

```
cd <your repository folder>  
git submodule add https://github.com/ConstantRobotics-Ltd/Lens.git 3rdparty/Lens  
git submodule update --init --recursive
```

In you repository folder will be created folder **3rdparty/Lens** which contains files of **Lens** repository with subrepositories **Frame** and **ConfigReader**. New structure of your repository:

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

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(${PARENT}_SUBMODULE_LENS ON CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_LENS)
    SET(${PARENT}_LENS ON CACHE BOOL "" FORCE)
    SET(${PARENT}_LENS_TEST OFF CACHE BOOL "" FORCE)
endif()

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

File **3rdparty/CMakeLists.txt** adds folder **Lens** to your project and excludes test application (Lens class test applications) from compiling. Your repository new structure will be:

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

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} Lens)
```

Done!