Lens interface C++ library

v4.0.0

Table of contents

- Read params from JSON file and write to JSON file

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 form **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 lens parameters and read/write lens parameters from JSON file. All lens controller 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) 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	- Subrepository **LensControllerDataStrctures** excluded. - Data structures moved to Lens.h file. - New parameters IDs added. - Description updated.
3.0.0	23.04.2023	- Methods signatures changed.
3.1.0	08.05.2023	- Added new parameters.
3.2.0	10.05.2023	- Parameters list changed.
4.0.0	18.06.2023	- Added **LensParams** class which contains lens controller parameters. - Added new methods to encode/decode lens control command. - Added new lens parameters. - Added tests.

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 of 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 version of **Lens** class. 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 controller version: " << Lens::getVersion() << std::endl;
```

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. It is recommended to use ';' symbol to divide part of initialization string. Example of lens controller initialization which uses serial port: "/dev/ttyUSB0;9600;100" ("/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 values. Method declaration:

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

Parameter	Value
params	Parameters (**LensParams** 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() 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 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. 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 (methos will return FALSE). It 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 is 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 according to **LensParam** enum (see description of **LensParam** enum).
value	Lens parameter value. Value depends on parameter ID (see description of **LensParam** enum).

Returns: TRUE is 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 according to LensParam enum (see description of LensParam enum).

Returns: parameter value or -1 of the parameters doesn't exist 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 according to **LensCommand** enum (see description of **LensCommand** enum).
arg	Lens command argument. Value depends on command ID (see description of **LensCommand** enum).

Returns: TRUE is the command was executed 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 **Frame** 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 lens parameter value for remote lens controller. Sometimes there is a need to control a lens remotely. To do this, the developer has to develop his own protocol and according to it encode the command and deliver it over the communication channel to lens controller. To simplify this, the **Lens** interface 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	0x04	Major version of Lens class.
2	0x00	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.

Byte	Value	Description
10	value	Parameter value **float** in Little-endian format.

encodeSetParamCommand(...) is static and used without **Lens** class instance. This method used on client side (remote control system). Command encoding 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 command to check any lens parameter value for lens remote control. Sometimes there is a need to control a lens remotely. To do this, the developer has to develop his own protocol and according to it encode the command and deliver it over the communication channel to lens controller. To simplify this, the **Lens** interface 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. 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 **LensParam** enum.
arg	Command argument value.

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 **int32_t** in Little-endian format.
4	id	Command ID **int32_t** in Little-endian format.

Byte	Value	Description
5	id	Command ID **int32_t** in Little-endian format.
6	id	Command ID **int32_t** in Little-endian format.
7	arg	Command argument value **float** in Little-endian format.
8	arg	Command argument value **float** in Little-endian format.
9	arg	Command argument value **float** in Little-endian format.
10	arg	Command argument value **float** in Little-endian format.

encodeCommand(...) is static and used without **Lens** class instance. This method used on client side (remote control system). Command 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. Sometimes there is a need to control a lens remotely. To do this, the developer has to develop his own protocol and according to it decode the command on lens controller size. To simplify this, the **Lens** interface class contains static method to decode input command. 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	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) and **LensParams** class.

LensCommand enum

Enum declaration:

```
enum class LensCommand
{
    /// Zoom tele. No arguments.
    ZOOM_TELE = 1,
    /// Zoom wide. No arguments.
    ZOOM_WIDE,
    /// Zoom to position. Argument:
    /// zoom position 0(full wide) - 65535(full tele).
    ZOOM_TO_POS,
    /// Stop zoom. No arguments.
    ZOOM_STOP,
    /// Focus far. No arguments.
    FOCUS_FAR,
    /// Focus near. No arguments.
    FOCUS_NEAR,
    /// Focus to position. Argument:
    /// focus position 0(full near) - 65535(full far).
    FOCUS_TO_POS,
    /// Focus stop. No arguments.
    FOCUS_STOP,
    /// Iris open. No arguments.
    IRIS_OPEN,
    /// Iris close. No arguments.
    IRIS_CLOSE,
    /// Iris to position. Argument:
    /// iris position 0(full close) - 65535(full open).
    IRIS_TO_POS,
    /// Iris stop. No arguments.
    IRIS_STOP,
    /// Autofocus start. No arguments.
    AF_START,
    /// Autofocus stop. No arguments.
    AF_STOP,
    /// Restart lens controller. No arguments.
    RESTART,
    /// Do zoom and focus hardware range detection.
    DETECT_HW_RANGES
};
```

Table 2 - Lens commands description. Some commands maybe 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.
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.

Command	Description
RESTART	Restart lens controller.
DETECT_HW_RANGES	Detect zoom and focus hardware ranges. After execution this command the lens controller should automatically set parameters (LensParam enum): ZOOM_HW_TELE_LIMIT, ZOOM_HW_WIDE_LIMIT, FOCUS_HW_FAR_LIMIT and FOCUS_HW_NEAR_LIMIT.

LensParam enum

Enum declaration:

```
/// Lens params.
enum class LensParam
{
    /// Zoom position (write/read). Value:
    /// zoom position 0(full wide) - 65535(full tele).
    ZOOM_POS = 1,
    /// Hardware zoom position (write/read). Argument:
    /// zoom position. Value depends on particular lens controller.
    ZOOM_HW_POS,
    /// Focus position (write/read). Value:
    /// focus position 0(full near) - 65535(full far).
    FOCUS_POS,
    /// Hardware focus position (write/read). Value:
    /// focus position. Value depends on particular lens controller.
    FOCUS_HW_POS,
    /// Iris position (write/read). Value:
    /// iris position 0(full close) - 65535(full open).
    IRIS_POS,
    /// Hardware iris position (write/read). Value:
    /// iris position. Value depends on particular lens controller.
    IRIS_HW_POS,
    /// Focus mode (write/read). Value:
    /// Value depends on particular lens controller.
    FOCUS_MODE,
    /// Filter mode (write/read).  Value:
    /// Value depends on particular lens controller.
    FILTER_MODE,
    /// Auto focus ROI top-left coordinate (write/read). Value:
    /// ROI top-left horizontal coordinate in pixels.
    AF_ROI_X0,
    /// Auto focus ROI top-left coordinate (write/read). Value:
    /// ROI top-left vertical coordinate in pixels.
    AF_ROI_Y0,
    /// Auto focus ROI bottom-right coordinate (write/read). Value:
    /// ROI bottom-right horizontal coordinate in pixels.
    AF_ROI_X1,
    /// Auto focus ROI bottom-right coordinate (write/read). Value:
    /// ROI bottom-right vertical coordinate in pixels.
    AF_ROI_Y1,
    /// Zoom speed (write/read). Value:
```

```
    /// zoom speed 0 to 100 %.
    ZOOM_SPEED,
    /// Hardware zoom speed (write/read). Value:
    /// zoom speed. Value depends on particular lens controller.
    ZOOM_HW_SPEED,
    /// Max hardware zoom speed (write/read). Value:
    /// zoom speed. Value depends on particular lens controller.
    ZOOM_HW_MAX_SPEED,
    /// Focus speed (write/read). Value:
    /// focus speed 0 to 100 %.
    FOCUS_SPEED,
    /// Hardware focus speed (write/read). Value:
    /// focus speed. Value depends on particular lens controller.
    FOCUS_HW_SPEED,
    /// Max hardware focus speed (write/read). Argument:
    /// focus speed. Value depends on particular lens controller.
    FOCUS_HW_MAX_SPEED,
    /// Iris speed (write/read). Value:
    /// iris speed 0 to 100 %.
    IRIS_SPEED,
    /// Hardware iris speed (write/read). Value:
    /// iris speed. Value depends on particular lens controller.
    IRIS_HW_SPEED,
    /// Max hardware iris speed (write/read). Value:
    /// iris speed. Value depends on particular lens controller.
    IRIS_HW_MAX_SPEED,
    /// Hardware zoom tele limit (write/read). Value:
    /// hardware zoom position. Value depends on particular lens controller.
    ZOOM_HW_TELE_LIMIT,
    /// Hardware zoom wide limit (write/read). Value:
    /// hardware zoom position. Value depends on particular lens controller.
    ZOOM_HW_WIDE_LIMIT,
    /// Hardware focus far limit (write/read). Value:
    /// hardware focus position. Value depends on particular lens controller.
    FOCUS_HW_FAR_LIMIT,
    /// Hardware focus near limit (write/read). Value:
    /// hardware focus position. Value depends on particular lens controller.
    FOCUS_HW_NEAR_LIMIT,
    /// Hardware iris open limit (write/read). Value:
    /// hardware iris position. Value depends on particular lens controller.
    IRIS_HW_OPEN_LIMIT,
    /// Hardware iris close limit (write/read). Value:
    /// hardware iris position. Value depends on particular lens controller.
    IRIS_HW_CLOSE_LIMIT,
    /// Focus factor value (read only). Value:
    /// Value depends on particular lens controller.
    FOCUS_FACTOR,
    /// Connection status (read only). Value:
    /// 0 - not connected, 1 - connected.
    IS_CONNECTED,
    /// Hardware focus speed in AF mode (write/read). Value:
    /// hardware focus speed. Value depends on particular lens controller.
    FOCUS_HW_AF_SPEED,
    /// Threshold for focus factor to start refocus (write/read). Value:
    /// threshold %: 0 - no check, 100 - changing x2.
    FOCUS_FACTOR_THRESHOLD,
```

```
    /// Refocus timeout, sec (write/read). Value:
    /// 0 - no refocus, to 100000 sec.
    REFOCUS_TIMEOUT_SEC,
    /// AF process mode (read only). Value: 0 - not active, 1 - active.
    AF_IS_ACTIVE,
    /// Iris mode. (write/read). Value:
    /// Value depends on particular lens controller.
    IRIS_MODE,
    /// Auto ROI width (write/read). Value: 0 to video frame size, pxl.
    AUTO_AF_ROI_WIDTH,
    /// Auto ROI height (write/read). Value: 0 to video frame size, pxl.
    AUTO_AF_ROI_HEIGHT,
    /// Auto ROI frame border in pixels (write/read). Value:
    /// border size from 0 to video frame min(width/height) / 2.
    AUTO_AF_ROI_BORDER,
    /// AF ROI mode (write/read). Value:
    /// 0 - Manual position, 1 - Auto position.
    AF_ROI_MODE,
    /// Optical extender mode (write/read). Value:
    /// Value depends on particular lens controller. Default: 0 -Off, 1 -On.
    EXTENDER_MODE,
    /// Stabilizer mode (write/read). Value:
    /// 0 - Off, 1 - On.
    STABILIZER_MODE,
    /// AF range (write/read). Value:
    /// Value depends on particular lens controller.
    AF_RANGE,
    /// Horizontal Field of view, degree (read only).
    X_FOV_DEG,
    /// Vertical Field of view, degree (read only).
    Y_FOV_DEG,
    /// Logging mode.
    /// Default values:
    /// 0 - Disable.
    /// 1 - Only file.
    /// 2 - Only terminal.
    /// 3 - File and terminal.
    LOG_MODE,
    /// Lens temperature, degree.
    TEMPERATURE,
    /// Open status: 1 - lens control port open, 0 - not open.
    IS_OPEN,
    /// Lens type. Value depends on implementation.
    TYPE,
    /// Lens custom param 1. Value depends on implementation.
    CUSTOM_1,
    /// Lens custom param 2. Value depends on implementation.
    CUSTOM_2,
    /// Lens custom param 3. Value depends on implementation.
    CUSTOM_3
};
```

Table 3 - Lens params description. Some params maybe 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 (zoom full wide) and maximum (zoom full tele) hardware 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 particular lens controller.
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 (focus full near) to 65535 (focus full far) regardless of the hardware value of the focus position. If the minimum (focus full near) and maximum (focus full far) hardware focus position limits are set by the user in the lens parameters, the range of the hardware focus position must be scaled to the user space 0-65535 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 particular lens controller.
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 open) regardless of the hardware value of the iris position. If the minimum (iris full close) and maximum (iris full open) iris position limits are set by the user in the lens parameters, the range of the hardware iris position must be scaled to the user space 0-65535 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.

Parameter	Access	Description
FOCUS_MODE	read / write	Focus mode. Parameter value depends on particular lens controller. Default values: 0 - Manual focus control, 1 - Auto focus control.
FILTER_MODE	read / write	Filter mode. Parameter value depends on particular lens controller. 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.
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 parameter the parameter ZOOM_SPEED must be updated automatically by lens controller. 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 particular lens controller. 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 by lens controller. Formula for calculating hardware speed: ZOOM_HW_SPEED = (ZOOM_SPEED / 100) * ZOOM_HW_MAX_SPEED.

Parameter	Access	Description
ZOOM_HW_MAX_SPEED	read / write	Maximum zoom hardware speed. Value depends on particular lens controller. If user sets new ZOOM_HW_MAX_SPEED value the parameters ZOOM_SPEED and ZOOM_HW_SPEED must be updated by lens controller 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 parameter the parameter FOCUS_SPEED must be updated automatically by lens controller. Formula for calculating speed: FOCUS_SPEED = (FOCUS_HW_SPEED / FOCUS_HW_MAX_SPEED) * 100.
FOCUS_HW_SPEED	read / write	Focus hardware speed. Value depends on particular lens controller. 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 by lens controller. Formula for calculating hardware speed: FOCUS_HW_SPEED = (FOCUS_SPEED / 100) * FOCUS_HW_MAX_SPEED.
FOCUS_HW_MAX_SPEED	read / write	Maximum focus hardware speed. Value depends on particular lens controller. 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.

Parameter	Access	Description
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 parameter the parameter IRIS_SPEED must be updated automatically by lens controller. Formula for calculating speed: IRIS_SPEED = (IRIS_HW_SPEED / IRIS_HW_MAX_SPEED) * 100.
IRIS_HW_SPEED	read / write	Iris hardware speed. Value depends on particular lens controller. 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 by lens controller. Formula for calculating hardware speed: IRIS_HW_SPEED = (IRIS_SPEED / 100) * IRIS_HW_MAX_SPEED.
IRIS_HW_MAX_SPEED	read / write	Maximum iris hardware speed. Value depends on particular lens controller. If user sets new IRIS_HW_MAX_SPEED value the parameters IRIS_SPEED and IRIS_HW_SPEED must be updated by lens controller 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.
ZOOM_HW_TELE_LIMIT	read / write	Zoom hardware tele limit. Value depends on particular lens controller. 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	read / write	Zoom hardware wide limit. Value depends on particular lens controller. Lens controller should control zoom position. Lens controller should stop zoom moving if hardware zoom position will be our of limits.
FOCUS_HW_FAR_LIMIT	read / write	Focus hardware far limit. Value depends on particular lens controller. Lens controller should control focus position. Lens controller should stop focus moving if hardware focus position will be our of limits.

Parameter	Access	Description
FOCUS_HW_NEAR_LIMIT	read / write	Focus hardware near limit. Value depends on particular lens controller. Lens controller should control focus position. Lens controller should stop focus moving if hardware focus position will be our of limits.
IRIS_HW_OPEN_LIMIT	read / write	Iris hardware open limit. Value depends on particular lens controller. 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	read / write	Iris hardware close limit. Value depends on particular lens controller. Lens controller should control iris position. Lens controller should stop iris moving if hardware iris position will be our of limits.
FOCUS_FACTOR	read only	Focus factor if it was calculated. If not calculated must be -1. Value depends on particular lens controller.
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 particular lens controller.
FOCUS_FACTOR_THRESHOLD	read / write	Threshold for changes of focus factor to start refocus. Value: threshold %: 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.

Parameter	Access	Description
IRIS_MODE	read / write	Iris mode. Value depends on particular lens controller. 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.
EXTENDER_MODE	read / write	Lens extender mode. Value depends on particular lens controller. Default values: 0 - no extender, 1 - x2 extender.
STABILIZER_MODE	read / write	Lens stabilization mode. Value depends on particular lens controller. Default values: 0 - no stabilization, 1 - stabilization.
AF_RANGE	read / write	Autofocus range. Value depends on particular lens controller.
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 equipment.
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 equipment.
LOG_MODE	read / write	Logging mode. Default values: 0 - Disable. 1 - Only file. 2 - Only terminal. 3 - File and terminal.
TEMPERATURE	read only	Lens temperature, degree.

Parameter	Access	Description
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 particular lens controller. Type allows to lens initialize necessary parameters for particular lens mode from one supplier.
CUSTOM_1	read / write	Lens custom parameter. Value depends on particular lens controller. Custom parameters used when particular lens equipment has specific unusual parameter.
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:

```
namespace cr
{
namespace lens
{

/// 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 obect.
     */
    FovPoint& operator= (const FovPoint& src);
};

/// Lens params structure.
class LensParams
{
public:
    /// Initialization string. Formats depends on implementation.
    std::string initString{"/dev/ttyUSB0;9600;20"};
    /// Zoom position (write/read). Value:
    /// zoom position 0(full wide) - 65535(full tele).
    int zoomPos{0};
    /// Hardware zoom position (write/read). Argument:
    /// zoom position. Value depends on particular lens controller.
    int zoomHwPos{0};
    /// Focus position (write/read). Value:
    /// focus position 0(full near) - 65535(full far).
    int focusPos{0};
    /// Hardware focus position (write/read). Value:
    /// focus position. Value depends on particular lens controller.
    int focusHwPos{0};
    /// Iris position (write/read). Value:
    /// iris position 0(full close) - 65535(full open).
    int irisPos{0};
    /// Hardware iris position (write/read). Value:
    /// iris position. Value depends on particular lens controller.
    int irisHwPos{0};
    /// Focus mode (write/read). Value:
    /// Value depends on particular lens controller.
    int focusMode{0};
    /// Filter mode (write/read).  Value:
    /// Value depends on particular lens controller.
    int filterMode{0};
    /// Auto focus ROI top-left coordinate (write/read). Value:
    /// ROI top-left horizontal coordinate in pixels.
    int afRoiX0{0};
    /// Auto focus ROI top-left coordinate (write/read). Value:
    /// ROI top-left vertical coordinate in pixels.
    int afRoiY0{0};
```

```
    /// Auto focus ROI bottom-right coordinate (write/read). Value:
    /// ROI bottom-right horizontal coordinate in pixels.
    int afRoiX1{0};
    /// Auto focus ROI bottom-right coordinate (write/read). Value:
    /// ROI bottom-right vertical coordinate in pixels.
    int afRoiY1{0};
    /// Zoom speed (write/read). Value:
    /// zoom speed 0 to 100 %.
    int zoomSpeed{50};
    /// Hardware zoom speed (write/read). Value:
    /// zoom speed. Value depends on particular lens controller.
    int zoomHwSpeed{50};
    /// Max hardware zoom speed (write/read). Value:
    /// zoom speed. Value depends on particular lens controller.
    int zoomHwMaxSpeed{50};
    /// Focus speed (write/read). Value:
    /// focus speed 0 to 100 %.
    int focusSpeed{50};
    /// Hardware focus speed (write/read). Value:
    /// focus speed. Value depends on particular lens controller.
    int focusHwSpeed{50};
    /// Max hardware focus speed (write/read). Argument:
    /// focus speed. Value depends on particular lens controller.
    int focusHwMaxSpeed{50};
    /// Iris speed (write/read). Value:
    /// iris speed 0 to 100 %.
    int irisSpeed{50};
    /// Hardware iris speed (write/read). Value:
    /// iris speed. Value depends on particular lens controller.
    int irisHwSpeed{50};
    /// Max hardware iris speed (write/read). Value:
    /// iris speed. Value depends on particular lens controller.
    int irisHwMaxSpeed{50};
    /// Hardware zoom tele limit (write/read). Value:
    /// hardware zoom position. Value depends on particular lens controller.
    int zoomHwTeleLimit{65535};
    /// Hardware zoom wide limit (write/read). Value:
    /// hardware zoom position. Value depends on particular lens controller.
    int zoomHwWideLimit{0};
    /// Hardware focus far limit (write/read). Value:
    /// hardware focus position. Value depends on particular lens controller.
    int focusHwFarLimit{65535};
    /// Hardware focus near limit (write/read). Value:
    /// hardware focus position. Value depends on particular lens controller.
    int focusHwNearLimit{0};
    /// Hardware iris open limit (write/read). Value:
    /// hardware iris position. Value depends on particular lens controller.
    int irisHwOpenLimit{65535};
    /// Hardware iris close limit (write/read). Value:
    /// hardware iris position. Value depends on particular lens controller.
    int irisHwCloseLimit{0};
    /// Focus factor value (read only). Value:
    /// Value depends on particular lens controller.
    float focusFactor{0.0f};
    /// Connection status (read only). Value:
    /// 0 - not connected, 1 - connected.
```

```
    bool isConnected{false};
    /// Hardware focus speed in AF mode (write/read). Value:
    /// hardware focus speed. Value depends on particular lens controller.
    int afHwSpeed{50};
    /// Threshold for focus factor to start refocus (write/read). Value:
    /// threshold %: 0 - no check, 100 - changing x2.
    float focusFactorThreshold{0.0f};
    /// Refocus timeout, sec (write/read). Value:
    /// 0 - no refocus, to 100000 sec.
    int refocusTimeoutSec{0};
    /// AF process mode (read only). Value: 0 - not active, 1 - active.
    bool afIsActive{false};
    /// Iris mode. (write/read). Value:
    /// Value depends on particular lens controller.
    int irisMode{0};
    /// Auto ROI width (write/read). Value: 0 to video frame size, pxl.
    int autoAfRoiWidth{150};
    /// Auto ROI height (write/read). Value: 0 to video frame size, pxl.
    int autoAfRoiHeight{150};
    /// Auto ROI frame border in pixels (write/read). Value:
    /// border size from 0 to video frame min(width/height) / 2.
    int autoAfRoiBorder{100};
    /// AF ROI mode (write/read). Value:
    /// 0 - Manual position, 1 - Auto position.
    int afRoiMode{0};
    /// Optical extender mode (write/read). Value:
    /// Value depends on particular lens controller. Default: 0 -Off, 1 -On.
    int extenderMode{0};
    /// Stabilizer mode (write/read). Value:
    /// 0 - Off, 1 - On.
    int stabiliserMode{0};
    /// AF range (write/read). Value:
    /// Value depends on particular lens controller.
    int afRange{0};
    /// Horizontal fiels of view, degree (read only).
    float xFovDeg{1.0f};
    /// Vertical fiels of view, degree (read only).
    float yFovDeg{1.0f};
    /// Logging mode.
    /// Default 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};
    /// Open status: 1 - lens control port open, 0 - not open.
    bool isOpen{false};
    /// Lens type service value (write/read). Value depends on implementation.
    int type{0};
    /// Lens custom param 1. Value depends on implementation.
    float custom1;
    /// Lens custom param 2. Value depends on implementation.
    float custom2;
    /// Lens custom param 3. Value depends on implementation.
```

```
    float custom3;
    /// List of field of view points.
    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.

Field	type	Description
initString	string	Initialization string. Particular lens controller can have unique init string format. It is recommended to use ';' symbol to divide part of initialization string. Example of lens controller initialization which uses serial port: "/dev/ttyUSB0;9600;100" ("/dev/ttyUSB0" - serial port name, "9600" - baudrate, "100" - serial port read timeout).
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 (zoom full wide) and maximum (zoom full tele) hardware 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 particular lens controller.
focusPos	int	Focus position. Setting a parameter is equivalent to the command FOCUS_TO_POS. Lens controller should have focus range from 0 (focus full near) to 65535 (focus full far) regardless of the hardware value of the focus position. If the minimum (focus full near) and maximum (focus full far) hardware focus position limits are set by the user in the lens parameters, the range of the hardware focus position must be scaled to the user space 0-65535 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 particular lens controller.

Field	type	Description
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 open) regardless of the hardware value of the iris position. If the minimum (iris full close) and maximum (iris full open) iris position limits are set by the user in the lens parameters, the range of the hardware iris position must be scaled to the user space 0-65535 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 particular lens controller.
focusMode	int	Focus mode. Parameter value depends on particular lens controller. Default values: 0 - Manual focus control, 1 - Auto focus control.
filterMode	int	Filter mode. Parameter value depends on particular lens controller. 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 zoomHwMaxSpeed).
zoomHwSpeed	int	Zoom hardware speed. Value depends on particular lens controller.
zoomHwMaxSpeed	int	Maximum zoom hardware speed. Value depends on particular lens controller.
focusSpeed	int	Focus speed. Lens controller should have focus speed range from 0 to 100% of max hardware focus speed (parameter focusHwMaxSpeed).
focusHwSpeed	int	Focus hardware speed. Value depends on particular lens controller.
focusHwMaxSpeed	int	Maximum focus hardware speed. Value depends on particular lens controller.

Field	type	Description
irisSpeed	int	Iris speed. Lens controller should have iris speed range from 0 to 100% of max hardware iris speed (parameter irisHwMaxSpeed).
irisHwSpeed	int	Iris hardware speed. Value depends on particular lens controller.
irisHwMaxSpeed	int	Maximum iris hardware speed. Value depends on particular lens controller.
zoomHwTeleLimit	int	Zoom hardware tele limit. Value depends on particular lens controller. Lens controller should control zoom position. Lens controller should stop zoom moving if hardware zoom position will be our of limits.
zoomHwWideLimit	int	Zoom hardware wide limit. Value depends on particular lens controller. Lens controller should control zoom position. Lens controller should stop zoom moving if hardware zoom position will be our of limits.
focusHwFarLimit	int	Focus hardware far limit. Value depends on particular lens controller. Lens controller should control focus position. Lens controller should stop focus moving if hardware focus position will be our of limits.
focusHwNearLimit	int	Focus hardware near limit. Value depends on particular lens controller. Lens controller should control focus position. Lens controller should stop focus moving if hardware focus position will be our of limits.
irisHwOpenLimit	int	Iris hardware open limit. Value depends on particular lens controller. Lens controller should control iris position. Lens controller should stop iris moving if hardware iris position will be our of limits.
irisHwCloseLimit	int	Iris hardware close limit. Value depends on particular lens controller. Lens controller should control iris position. Lens controller should stop iris moving if hardware iris position will be our of limits.
focusFactor	float	Focus factor if it was calculated. If not calculated must be -1. Value depends on particular lens controller.
isConnected	bool	Lens connection status. Connection status shows if the lens controller has data exchange with lens equipment.
afHwSpeed	int	Focus hardware speed in autofocus mode. Value depends on particular lens controller.

Field	type	Description
focusFactorThreshold	float	Threshold for changes of focus factor to start refocus. Value: threshold %: 0 - no check, 100 - changing x2.
refocusTimeoutSec	int	Timeout for automatic refocus in seconds. Value: 0 - no automatic refocus, 100000 - maximum value.
afIsActive	bool	Flag about active autofocus algorithm. Value: FALSE - autofocus not working, TRUE - working.
irisMode	int	Iris mode. Value depends on particular lens controller. 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 particular lens controller. Default values: 0 - no extender, 1 - x2 extender.
stabiliserMode	int	Lens stabilization mode. Value depends on particular lens controller. Default values: 0 - no stabilization, 1 - stabilization.
afRange	int	Autofocus range. Value depends on particular lens controller.
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 equipment.
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 equipment.

Field	type	Description
logMode	int	Logging mode. Default 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.
type	int	Lens type. Value depends on particular lens controller. Type allows to lens initialize necessary parameters for particular lens mode from one supplier.
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.
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 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, int size);
```

Parameter	Value
data	Pointer to encode data buffer.
size	Size of encoded 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, size);
cout << "Encoded data size: " << size << " bytes" << endl;

// Decode data.
LensParams out;
if (!out.decode(data, size))
    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;
}
```