VentusCamera C++ library

v1.0.0

Table of contents

Overview

The **VentusCamera** C++ library is a software controller for **WIND firmware** compatible cameras, which mostly includes **Ventus** cameras produced by SIERRA-OLYMPIA TECHNOLOGIES INC. company. The **VentusCamera** library inherits Lens and Camera interfaces, so the library has set of two different interfaces to control camera and lens. It includes source code of libraries: Lens interface library (provides interface and data structures to control lenses, Apache 2.0 license), Camera interface library (provides interface and data structures to control cameras, Apache 2.0 license), Logger library (provides function to print log information in console and files, Apache 2.0 license) and SerialPort library (provides functions to work with serial ports, Apache 2.0 license). The **VentusCamera** library provides simple interface to be integrated in any C++ projects. The library repository (folder) is provided by source code and doesn't have third-party dependencies to be specially installed in OS. It is developed with C++17 standard and compatible with Linux and Windows.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	10.03.2024	First version of the library.

Library files

The **VentusCamera** C++ library is provided as source code only. Users are given a set of files in the form of a CMake project (repository). The repository structure is outlined below:

```
CMakeLists.txt ----------------- Main CMake file of the library.
3rdparty ----------------------- Folder with third-party libraries.
    CMakeLists.txt ------------- CMake file which includes third-party libraries.
    Camera --------------------- Folder with the Camera library.
    FipProtocolParser ---------- Folder with the WIND protocol parser source code.
    Lens ----------------------- Folder with the Lens library.
    Logger --------------------- Folder with the Logger library.
    SerialPort ----------------- Folder with the SerialPort library.
src ---------------------------- Folder with source code of the library.
    CMakeLists.txt ------------- CMake file of the library.
    VentusCamera.cpp ----------- Source code file of the library.
    VentusCamera.h ------------- Header file which includes VentusCamera class
declaration.
    VentusCameraVersion.h ------ Header file which includes version of the library.
    VentusCameraVersion.h.in --- CMake service file to generate version file.
test --------------------------- Folder with test application.
    CMakeLists.txt ------------- CMake file for test application.
    main.cpp ------------------- Source code file of test application.
example ------------------------ Folder with example application.
    CMakeLists.txt ------------- CMake file for example application.
    main.cpp ------------------- Source code file of example application.
```

VentusCamera class description

VentusCamera class declaration

VentusCamera interface class declared in **VentusCamera.h** file. The **VentusCamera** class includes Lens and Camera interfaces. Class declaration:

```
class VentusCamera : public cr::camera::Camera, public cr::lens::Lens
{
public:

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

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

    // ########################################
    // CAMERA CONTROL INTERFACE
```

```
    // ######################################

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

    /// Init camera by parameters structure.
    bool initCamera(cr::camera::CameraParams& params) override;

    /// Method closes serial port and stops all threads.
    void closeCamera() override;

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

    /// Get camera connection status.
    bool isCameraConnected() override;

    /// Set the camera parameter.
    bool setParam(cr::camera::CameraParam id, float value) override;

    /// Get the camera parameter.
    float getParam(cr::camera::CameraParam id) override;

    /// Get the camera params.
    void getParams(cr::camera::CameraParams& params) override;

    /// Execute camera command.
    bool executeCommand(cr::camera::CameraCommand id) override;

    // ######################################
    // LENS CONTROL INTERFACE
    // ######################################

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

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

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

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

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

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

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

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

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

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

getVersion method

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

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

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

```
std::cout << "VentusCamera class version: " << cr::camera::VentusCamera::getVersion();
```

Console output:

```
VentusCamera class version: 1.0.0
```

decodeAndExecuteCommand method

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

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

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

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

openLens method

The **openCamera(...)** opens serial port and runs threads to communicate with **WIND Firmware** based devices. To initialize serial port user can call openCamera(...) or openLens(...) (no matter which one). If serial port is already open the method will return TRUE. Camera and lens parameters will be initialized with default values. Method declaration:

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

Parameter	Value
initString	Initialization string. initString can have 3 different form of initialization string for the device: - [serial port name];[baudrate];[camera address];[wait data timeout] - [serial port name];[baudrate];[camera address] - [serial port name];[baudrate] When camera address is not given it is set to 1 as default and same for wait data timeout - 100ms. Recommended initialization string format for controllers, which uses serial port: **"/dev/ttyUSB0;9600;1;100"**

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

openCamera method

The **openCamera(...)** opens serial port and runs threads to communicate with **WIND Firmware** based devices. To initialize serial port user can call openCamera(...) or openLens(...) (no matter which one). If serial port is already open the method will return TRUE. Camera and lens parameters will be initialized with default values. Method declaration:

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

Parameter	Value
initString	Initialization string. initString can have 3 different form of initialization string for the device: - [serial port name];[baudrate];[camera address];[wait data timeout] - [serial port name];[baudrate];[camera address] - [serial port name];[baudrate] hen camera address is not given it is set to 1 as default and same for wait data timeout - 100ms. Recommended initialization string format for controllers, which uses serial port: **"/dev/ttyUSB0;9600;1;100"**

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

initLens method

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

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

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

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

initCamera method

The **initCamera(...)** initializes controller and sets camera params (Camera interface). The method will set given camera params and after it will call openCamera(...) method. **Lens** parameters will be initialized by default (if not initialized by default). After successful initialization the library will run communication threads (thread to communicate with equipment via serial port) if it was not ran before. Method declaration:

```
bool initCamera(CameraParams& params) override;
```

Parameter	Value
params	CameraParams class object. CameraParams class includes initString, which is used in openCamera(...) method.

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

closeLens method

The **closeLens()** closes serial port and stops all communication threads. The result of the method is equal to closeCamera() (user can use any of them). Method declaration:

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

closeCamera method

The **closeCamera()** closes serial port and stops all communication threads. The result of the method is equal to closeLens() method (user can use any of them). Method declaration:

```
void closeCamera() override;
```

isLensOpen method

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

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

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

isCameraOpen method

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

```
bool isCameraOpen() override;
```

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

isLensConnected method

The **isLensConnected()** shows if the controller receives responses from equipment (camera). For example, if serial port open but equipment not active (no power). In this case methods isLensOpen() and isCameraOpen() will return TRUE but **isLensConnected()** method will return FALSE. This method is equal to isCameraConnected() method (user can use any of them). Method declaration:

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

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

isCameraConnected method

The **isCameraConnected()** shows if the controller receives responses from equipment (camera). For example, if serial port open but equipment not active (no power). In this case methods isLensOpen() and isCameraOpen() methods will return TRUE but **isCameraConnected()** method will return FALSE. This method is equal to isLensConnected() method(user can use any of them). Method declaration:

```
bool isCameraConnected() override;
```

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

setParam method

The **setParam(...)** is overloaded method. The method intended to set Camera or Lens parameter value. Method declaration:

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

Parameter	Description
id	Lens parameter ID according to LensParam, camera parameter ID according to CameraParam.
value	Camera or Lens parameter value. Valid values depend on parameter ID.

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

getParam method

The **getParam(...)** is overloaded method. The method intended to obtain Camera or Lens parameter value. Method declaration:

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

Parameter	Description
id	Lens parameter ID according to LensParam or camera parameter ID according to CameraParam.

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

getParams method

The **getParams(...)** is overloaded method. The method intended to obtain Camera or Lens parameters structure. Method declaration:

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

Parameter	Description
params	Reference to LensParams class object or CameraParams class object.

executeCommand method

The **executeCommand(...)** is overloaded method. The method intended to execute Camera or Lens action command. Method declaration:

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

Parameter	Description
id	Camera command ID according to CameraCommand or lens command ID according to LensCommand.
arg	Only for lens interface. Lens command argument. Valid values depend on command ID.

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

addVideoFrame method

addVideoFrame(...) copies video frame data to lens controller to perform autofocus algorithm. It is not supported by VentusCamera. Method declaration:

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

Parameter	Description
frame	Frame class object.

encodeSetParamCommand method of Lens class

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

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

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

Parameter	Description
value	Parameter value.

SET_PARAM command format:

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

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

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

encodeCommand method of Lens class

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

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

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

COMMAND format:

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

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

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

decodeCommand method of Lens class

decodeCommand(...) static method of Lens interface class designed to decode command on lens controller side. o control a lens remotely, the developer has to develop his own protocol and according to it decode the command on lens controller side. To simplify this, the **Lens** interface class contains static method to decode input command (commands should be encoded by methods **encodeSetParamsCommand(...)** or

encodeCommand(...)). The **Lens** class provides two types of commands: a parameter change command (SET_PARAM) and an action command (COMMAND). Method declaration:

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

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

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

encodeSetParamCommand method of Camera class

encodeSetParamCommand(...) static method of Camera interface class encodes command to change any remote camera parameter value. To control a camera remotely, the developer has to design his own protocol and according to it encode the command and deliver it over the communication channel. To simplify this, the **Camera** class contains static methods for encoding the control command. The **Camera** 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, CameraParam 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 CameraParam enum.
value	Parameter value.

SET_PARAM command format:

Byte	Value	Description
0	0x01	SET_PARAM command header value.
1	Major	Major version of Camera class.

Byte	Value	Description
2	Minor	Minor version of Camera class.
3	id	Parameter ID **int32_t** in Little-endian format.
4	id	Parameter ID **int32_t** in Little-endian format.
5	id	Parameter ID **int32_t** in Little-endian format.
6	id	Parameter ID **int32_t** in Little-endian format.
7	value	Parameter value **float** in Little-endian format.
8	value	Parameter value **float** in Little-endian format.
9	value	Parameter value **float** in Little-endian format.
10	value	Parameter value **float** in Little-endian format.

encodeSetParamCommand(...) is static and used without **Camera** class instance. This method used on client side (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.
Camera::encodeSetParamCommand(data, size, CameraParam::ROI_X0, outValue);
```

encodeCommand method of Camera class

encodeCommand(...) static method of Camera interface class encodes command for camera remote control. To control a camera remotely, the developer has to design his own protocol and according to it encode the command and deliver it over the communication channel. To simplify this, the **Camera** class contains static methods for encoding the control command. The **Camera** 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, CameraCommand id);
```

Parameter	Description
data	Pointer to data buffer for encoded command. Must have size >= 7.
size	Size of encoded data. Will be 7 bytes.
id	Command ID according to CameraCommand enum.

COMMAND format:

Byte	Value	Description
0	0x00	COMMAND header value.
1	Major	Major version of Camera class.
2	Minor	Minor version of Camera class.
3	id	Command ID **int32_t** in Little-endian format.
4	id	Command ID **int32_t** in Little-endian format.
5	id	Command ID **int32_t** in Little-endian format.
6	id	Command ID **int32_t** in Little-endian format.

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

```
// Buffer for encoded data.
uint8_t data[7];
// Size of encoded data.
int size = 0;
// Encode command.
Camera::encodeCommand(data, size, CameraCommand::NUC);
```

decodeCommand method of Camera class

decodeCommand(...) static method of Camera interface class decodes command on camera controller side. Method declaration:

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

Parameter	Description
data	Pointer to input command.
size	Size of command. Must be 11 bytes for SET_PARAM and 7 bytes for COMMAND.
paramId	Camera parameter ID according to CameraParam enum. After decoding SET_PARAM command the method will return parameter ID.
commandId	Camera command ID according to CameraCommand enum. After decoding COMMAND the method will return command ID.
value	Camera parameter value (after decoding SET_PARAM command).

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

Data structures

LensCommand enum

Enum declaration:

```
enum class LensCommand
{
    /// Move zoom tele (in).
    ZOOM_TELE = 1,
    /// Move zoom wide (out).
    ZOOM_WIDE,
    /// Move zoom to position.
    ZOOM_TO_POS,
    /// Stop zoom moving including stop zoom to position command.
    ZOOM_STOP,
    /// Move focus far.
    FOCUS_FAR,
    /// Move focus near.
    FOCUS_NEAR,
    /// Move focus to position.
    FOCUS_TO_POS,
    /// Stop focus moving including stop focus to position command.
    FOCUS_STOP,
    /// Move iris open.
    IRIS_OPEN,
    /// Move iris close.
    IRIS_CLOSE,
    /// Move iris to position.
    IRIS_TO_POS,
    /// Stop iris moving including stop iris to position command.
    IRIS_STOP,
    /// Start autofocus.
    AF_START,
    /// Stop autofocus.
    AF_STOP,
    /// Restart lens controller.
    RESTART,
    /// Detect zoom and focus hardware ranges.
    DETECT_HW_RANGES
};
```

Table 2 - Lens commands description.

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

Command	Description
ZOOM_WIDE	Move zoom wide (out). Command doesn't have arguments. User can set zoom movement speed via lens parameters.
ZOOM_TO_POS	Move zoom to position. Controller has zoom range from 0 (full wide) to 65535 (full tele) regardless of the hardware value of the zoom position. User can set zoom movement speed via lens parameters.
ZOOM_STOP	Stop zoom moving including stop zoom to position command.
FOCUS_FAR	Move focus far. Command doesn't have arguments. User can set focus movement speed via lens parameters.
FOCUS_NEAR	Move focus near. Command doesn't have arguments. User can set focus movement speed via lens parameters.
FOCUS_TO_POS	Move focus to position. Controller has focus range from 0 (full near) to 65535 (full far) regardless of the hardware value of the focus position. User can set focus movement speed via lens parameters.
FOCUS_STOP	Stop focus moving including stop focus to position command.
IRIS_OPEN	**Not supported** by **VentusCamera** library.
IRIS_CLOSE	**Not supported** by **VentusCamera** library.
IRIS_TO_POS	**Not supported** by **VentusCamera** library.
IRIS_STOP	**Not supported** by **VentusCamera** library.
AF_START	Start autofocus. Command doesn't have arguments.
AF_STOP	Stop autofocus, switch to manual focus. Command doesn't have arguments.
RESTART	**Not supported** by **VentusCamera** library.
DETECT_HW_RANGES	**Not supported** by **VentusCamera** library.

CameraCommand enum

Enum declaration:

```
enum class CameraCommand
{
    /// Restart camera controller.
    RESTART = 1,
    /// Do NUC.
    NUC,
    /// Apply settings.
    APPLY_PARAMS,
    /// Save params.
    SAVE_PARAMS,
    /// Menu on.
    MENU_ON,
```

```
    /// Menu off.
    MENU_OFF,
    /// Menu set.
    MENU_SET,
    /// Menu up.
    MENU_UP,
    /// Menu down.
    MENU_DOWN,
    /// Menu left.
    MENU_LEFT,
    /// Menu right.
    MENU_RIGHT,
    /// Freeze, Argument: time msec.
    FREEZE,
    /// Disable freeze.
    DEFREEZE
};
```

Table 3 - Camera commands description.

Command	Description
RESTART	**Not supported** by **VentusCamera** library.
NUC	Execute NUC - perform Flat Field Correction. It is possible to set NUC mode via **setParam(...)**. Use **0** - to perform Flat Field Correction with shutter closed and **1** to perform with shutter open.
APPLY_PARAMS	**Not supported** by **VentusCamera** library.
SAVE_PARAMS	**Not supported** by **VentusCamera** library.
MENU_ON	**Not supported** by **VentusCamera** library.
MENU_OFF	**Not supported** by **VentusCamera** library.
MENU_SET	**Not supported** by **VentusCamera** library.
MENU_UP	**Not supported** by **VentusCamera** library.
MENU_DOWN	**Not supported** by **VentusCamera** library.
MENU_LEFT	**Not supported** by **VentusCamera** library.
MENU_RIGHT	**Not supported** by **VentusCamera** library.
FREEZE	**Not supported** by **VentusCamera** library.
DEFREEZE	**Not supported** by **VentusCamera** library.

LensParam enum

Enum declaration:

```
enum class LensParam
{
    /// Zoom position.
    ZOOM_POS = 1,
    /// Hardware zoom position.
    ZOOM_HW_POS,
    /// Focus position.
    FOCUS_POS,
    /// Hardware focus position.
    FOCUS_HW_POS,
    /// Iris position.
    IRIS_POS,
    /// Hardware iris position.
    IRIS_HW_POS,
    /// Focus mode.
    FOCUS_MODE,
    /// Filter mode.
    FILTER_MODE,
    /// Autofocus ROI top-left corner horizontal position in pixels.
    AF_ROI_X0,
    /// Autofocus ROI top-left corner vertical position in pixels.
    AF_ROI_Y0,
    /// Autofocus ROI bottom-right corner horizontal position in pixels.
    AF_ROI_X1,
    /// Autofocus ROI bottom-right corner vertical position in pixels.
    AF_ROI_Y1,
    /// Zoom speed.
    ZOOM_SPEED,
    /// Zoom hardware speed.
    ZOOM_HW_SPEED,
    /// Maximum zoom hardware speed.
    ZOOM_HW_MAX_SPEED,
    /// Focus speed.
    FOCUS_SPEED,
    /// Focus hardware speed.
    FOCUS_HW_SPEED,
    /// Maximum focus hardware speed.
    FOCUS_HW_MAX_SPEED,
    /// Iris speed.
    IRIS_SPEED,
    /// Iris hardware speed.
    IRIS_HW_SPEED,
    /// Maximum iris hardware speed.
    IRIS_HW_MAX_SPEED,
    /// Zoom hardware tele limit.
    ZOOM_HW_TELE_LIMIT,
    /// Zoom hardware wide limit.
    ZOOM_HW_WIDE_LIMIT,
    /// Focus hardware far limit.
    FOCUS_HW_FAR_LIMIT,
    /// Focus hardware near limit.
```

```
    FOCUS_HW_NEAR_LIMIT,
    /// Iris hardware open limit.
    IRIS_HW_OPEN_LIMIT,
    /// Iris hardware close limit.
    IRIS_HW_CLOSE_LIMIT,
    /// Focus factor if it was calculated.
    FOCUS_FACTOR,
    /// Lens connection status.
    IS_CONNECTED,
    /// Focus hardware speed in autofocus mode.
    FOCUS_HW_AF_SPEED,
    /// Threshold for changes of focus factor to start refocus.
    FOCUS_FACTOR_THRESHOLD,
    /// Timeout for automatic refocus in seconds.
    REFOCUS_TIMEOUT_SEC,
    /// Flag about active autofocus algorithm.
    AF_IS_ACTIVE,
    /// Iris mode.
    IRIS_MODE,
    /// ROI width (pixels).
    AUTO_AF_ROI_WIDTH,
    /// ROI height (pixels).
    AUTO_AF_ROI_HEIGHT,
    /// Video frame border size (along vertical and horizontal axes).
    AUTO_AF_ROI_BORDER,
    /// AF ROI mode (write/read). Value: 0 - Manual position, 1 - Auto position.
    AF_ROI_MODE,
    /// Lens extender mode.
    EXTENDER_MODE,
    /// Lens stabilization mode.
    STABILIZER_MODE,
    /// Autofocus range.
    AF_RANGE,
    /// Current horizontal Field of view, degree.
    X_FOV_DEG,
    /// Current vertical Field of view, degree.
    Y_FOV_DEG,
    /// Logging mode.
    LOG_MODE,
    /// Lens temperature, degree.
    TEMPERATURE,
    /// Lens controller initialization status.
    IS_OPEN,
    /// Lens type.
    TYPE,
    /// Lens custom parameter.
    CUSTOM_1,
    /// Lens custom parameter.
    CUSTOM_2,
    /// Lens custom parameter.
    CUSTOM_3
};
```

Table 5 - Lens params description.

Parameter	Access	Description
ZOOM_POS	read / write	Zoom position. Setting a parameter is equivalent to the command ZOOM_TO_POS. Controller has zoom range from 0 (full wide) to 65535 (full tele). User can set zoom movement speed via lens parameters.
ZOOM_HW_POS	read / write	Hardware zoom position. Parameter has same value as ZOOM_POS parameter.
FOCUS_POS	read / write	Focus position. Setting a parameter is equivalent to the command FOCUS_TO_POS. Controller has focus range from 0 (full near) to 65535 (full far). User can set focus movement speed via lens parameters.
FOCUS_HW_POS	read / write	Hardware focus position. Parameter has same value as FOCUS_POS parameter.
IRIS_POS	read / write	**Not supported** by **VentusCamera** library.
IRIS_HW_POS	read / write	**Not supported** by **VentusCamera** library.
FOCUS_MODE	read / write	Focus mode. Values : **0** - Abort Autofocus **1** - Run Autofocus non blocking mode. **2** - Run Autofocus blocking mode. **3** - Moves the focus motor to the 'infinity focus' position.
FILTER_MODE	read / write	**Not supported** by **VentusCamera** library.
AF_ROI_X0	read / write	**Not supported** by **VentusCamera** library.
AF_ROI_Y0	read / write	**Not supported** by **VentusCamera** library.
AF_ROI_X1	read / write	**Not supported** by **VentusCamera** library.
AF_ROI_Y1	read / write	**Not supported** by **VentusCamera** library.
ZOOM_SPEED	read / write	Zoom speed. Controller has zoom speed range from 0 to 100%.
ZOOM_HW_SPEED	read / write	Zoom speed. Controller has zoom speed range from 0 to 100%.
ZOOM_HW_MAX_SPEED	read / write	**Not supported** by **VentusCamera** library.

Parameter	Access	Description
FOCUS_SPEED	read / write	Focus speed. Controller has focus speed range from 0 to 100%.
FOCUS_HW_SPEED	read / write	Focus speed. Controller has focus speed range from 0 to 100%.
FOCUS_HW_MAX_SPEED	read / write	**Not supported** by **VentusCamera** library.
IRIS_SPEED	read / write	**Not supported** by **VentusCamera** library.
IRIS_HW_SPEED	read / write	**Not supported** by **VentusCamera** library.
IRIS_HW_MAX_SPEED	read / write	**Not supported** by **VentusCamera** library.
ZOOM_HW_TELE_LIMIT	read / write	**Not supported** by **VentusCamera** library.
ZOOM_HW_WIDE_LIMIT	read / write	**Not supported** by **VentusCamera** library.
FOCUS_HW_FAR_LIMIT	read / write	**Not supported** by **VentusCamera** library.
FOCUS_HW_NEAR_LIMIT	read / write	**Not supported** by **VentusCamera** library.
IRIS_HW_OPEN_LIMIT	read / write	**Not supported** by **VentusCamera** library.
IRIS_HW_CLOSE_LIMIT	read / write	**Not supported** by **VentusCamera** library.
FOCUS_FACTOR	read only	**Not supported** by **VentusCamera** library.
IS_CONNECTED	read only	Connection status. Connection status shows if the controller has data exchange with equipment. For example, if serial port open but equipment can be not active (no power). In this case connection status shows that controller doesn't have data exchange with equipment (methos will return 0). It the controller has data exchange with equipment the method will return 1. If the controller not initialize the connection status always FALSE. Values: **0** - not connected. **1** - connected.
FOCUS_HW_AF_SPEED	read / write	**Not supported** by **VentusCamera** library.

Parameter	Access	Description
FOCUS_FACTOR_THRESHOLD	read / write	**Not supported** by **VentusCamera** library.
REFOCUS_TIMEOUT_SEC	read / write	**Not supported** by **VentusCamera** library.
AF_IS_ACTIVE	read only	**Not supported** by **VentusCamera** library.
IRIS_MODE	read / write	**Not supported** by **VentusCamera** library.
AUTO_AF_ROI_WIDTH	read / write	**Not supported** by **VentusCamera** library.
AUTO_AF_ROI_HEIGHT	read / write	**Not supported** by **VentusCamera** library.
AUTO_AF_ROI_BORDER	read / write	**Not supported** by **VentusCamera** library.
AF_ROI_MODE	read / write	**Not supported** by **VentusCamera** library.
EXTENDER_MODE	read / write	**Not supported** by **VentusCamera** library.
STABILIZER_MODE	read / write	**Not supported** by **VentusCamera** library.
AF_RANGE	read / write	**Not supported** by **VentusCamera** library.
X_FOV_DEG	read only	**Not supported** by **VentusCamera** library.
Y_FOV_DEG	read only	**Not supported** by **VentusCamera** library.
LOG_MODE	read / write	Logging mode. Values: **0** - Disable, **1** - Only file, **2** - Only terminal (console), **3** - File and terminal.
TEMPERATURE	read only	**Not supported** by **VentusCamera** library.

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

CameraParam enum

Enum declaration:

```
enum class CameraParam
{
    /// Video frame width. Value from 0 to 16384.
    WIDTH = 1,
    /// Video frame height Value from 0 to 16384.
    HEIGHT,
    /// Display menu mode.
    DISPLAY_MODE,
    /// Video output type.
    VIDEO_OUTPUT,
    /// Logging mode.
    LOG_MODE,
    /// Exposure mode.
    EXPOSURE_MODE,
    /// Exposure time of the camera sensor.
    EXPOSURE_TIME,
    /// White balance mode.
    WHITE_BALANCE_MODE,
    /// White balance area.
    WHITE_BALANCE_AREA,
    /// White dynamic range mode.
    WIDE_DYNAMIC_RANGE_MODE,
    /// Image stabilization mode.
    STABILIZATION_MODE,
    /// ISO sensitivity.
    ISO_SENSITIVITY,
```

```
    /// Scene mode.
    SCENE_MODE,
    /// FPS.
    FPS,
    /// Brightness mode.
    BRIGHTNESS_MODE,
    /// Brightness. Value 0 - 100%.
    BRIGHTNESS,
    /// Contrast. Value 1 - 100%.
    CONTRAST,
    /// Gain mode.
    GAIN_MODE,
    /// Gain. Value 1 - 100%.
    GAIN,
    /// Sharpening mode.
    SHARPENING_MODE,
    /// Sharpening. Value 1 - 100%.
    SHARPENING,
    /// Palette.
    PALETTE,
    /// Analog gain control mode.
    AGC_MODE,
    /// Shutter mode.
    SHUTTER_MODE,
    /// Shutter position. 0 (full close) - 65535 (full open).
    SHUTTER_POSITION,
    /// Shutter speed. Value: 0 - 100%.
    SHUTTER_SPEED,
    /// Digital zoom mode.
    DIGITAL_ZOOM_MODE,
    /// Digital zoom. Value 1.0 (x1) - 20.0 (x20).
    DIGITAL_ZOOM,
    /// Exposure compensation mode.
    EXPOSURE_COMPENSATION_MODE,
    /// Exposure compensation position.
    EXPOSURE_COMPENSATION_POSITION,
    /// Defog mode.
    DEFOG_MODE,
    /// Dehaze mode.
    DEHAZE_MODE,
    /// Noise reduction mode.
    NOISE_REDUCTION_MODE,
    /// Black and white filter mode.
    BLACK_WHITE_FILTER_MODE,
    /// Filter mode.
    FILTER_MODE,
    /// NUC mode for thermal cameras.
    NUC_MODE,
    /// Auto NUC interval for thermal cameras.
    AUTO_NUC_INTERVAL_MSEC,
    /// Image flip mode.
    IMAGE_FLIP,
    /// DDE mode.
    DDE_MODE,
    /// DDE level.
    DDE_LEVEL,
```

```
    /// ROI top-left horizontal position, pixels.
    ROI_X0,
    /// ROI top-left vertical position, pixels.
    ROI_Y0,
    /// ROI bottom-right horizontal position, pixels.
    ROI_X1,
    /// ROI bottom-right vertical position, pixels.
    ROI_Y1,
    /// Camera temperature, degree.
    TEMPERATURE,
    /// ALC gate.
    ALC_GATE,
    /// Sensor sensitivity.
    SENSETIVITY,
    /// Changing mode (day / night).
    CHANGING_MODE,
    /// Changing level (day / night).
    CHANGING_LEVEL,
    /// Chroma level. Values: 0 - 100%.
    CHROMA_LEVEL,
    /// Details, enhancement. Values: 0 - 100%.
    DETAIL,
    /// Camera settings profile.
    PROFILE,
    /// Connection status (read only). Shows if we have respond from camera.
    /// Value: 0 - not connected, 2 - connected.
    IS_CONNECTED,
    /// Open status (read only):
    /// 1 - camera control port open, 0 - not open.
    IS_OPEN,
    /// Camera type.
    TYPE,
    /// Camera custom param.
    CUSTOM_1,
    /// Camera custom param.
    CUSTOM_2,
    /// Camera custom param.
    CUSTOM_3
};
```

Table 6 - Camera params description.

Parameter	Access	Description
WIDTH	read / write	**Not supported** by **VentusCamera** library.
HEIGHT	read / write	**Not supported** by **VentusCamera** library.
DISPLAY_MODE	read / write	**Not supported** by **VentusCamera** library.
VIDEO_OUTPUT	read / write	**Not supported** by **VentusCamera** library.

Parameter	Access	Description
LOG_MODE	read / write	Logging mode. Values: **0** - Disable, **1** - Only file, **2** - Only terminal (console), **3** - File and terminal.
EXPOSURE_MODE	read / write	**Not supported** by **VentusCamera** library.
EXPOSURE_TIME	read / write	**Not supported** by **VentusCamera** library.
WHITE_BALANCE_MODE	read / write	**Not supported** by **VentusCamera** library.
WHITE_BALANCE_AREA	read / write	**Not supported** by **VentusCamera** library.
WHITE_DINAMIC_RANGE_MODE	read / write	**Not supported** by **VentusCamera** library.
STABILIZATION_MODE	read / write	**Not supported** by **VentusCamera** library.
ISO_SENSITIVITY	read / write	**Not supported** by **VentusCamera** library.
SCENE_MODE	read / write	**Not supported** by **VentusCamera** library.
FPS	read / write	**Not supported** by **VentusCamera** library.
BRIGHTNESS_MODE	read / write	**Not supported** by **VentusCamera** library.
BRIGHTNESS	read / write	**Not supported** by **VentusCamera** library.
CONTRAST	read / write	**Not supported** by **VentusCamera** library.
GAIN_MODE	read / write	**Not supported** by **VentusCamera** library.
GAIN	read / write	**Not supported** by **VentusCamera** library.
SHARPENING_MODE	read / write	**Not supported** by **VentusCamera** library.
SHARPENING	read / write	**Not supported** by **VentusCamera** library.

Parameter	Access	Description
PALETTE	read / write	**Not supported** by **VentusCamera** library.
AGC_MODE	read / write	**Not supported** by **VentusCamera** library.
SHUTTER_MODE	read / write	**Not supported** by **VentusCamera** library.
SHUTTER_POSITION	read / write	Shutter position value. Supported values: **0** - close shutter **1** - open shutter.
SHUTTER_SPEED	read / write	**Not supported** by **VentusCamera** library.
DIGITAL_ZOOM_MODE	read / write	**Not supported** by **VentusCamera** library.
DIGITAL_ZOOM	read only	**Not supported** by **VentusCamera** library.
EXPOSURE_COMPENSATION_MODE	read only	**Not supported** by **VentusCamera** library.
EXPOSURE_COMPENSATION_POSITION	read / write	**Not supported** by **VentusCamera** library.
DEFOG_MODE	read / write	**Not supported** by **VentusCamera** library.
DEHAZE_MODE	read / write	**Not supported** by **VentusCamera** library.
NOISE_REDUCTION_MODE	read / write	**Not supported** by **VentusCamera** library.
BLACK_WHITE_FILTER_MODE	read only	**Not supported** by **VentusCamera** library.
FILTER_MODE	read / write	**Not supported** by **VentusCamera** library.
NUC_MODE	read / write	Set NUC mode - sets whether the Flat Field Correction (FFC) will be performed automatically with chosen period or only manually on command: **0** - manually **1** - automatically.
AUTO_NUC_INTERVAL	read / write	Set the period for automatic Flat Field Correction (FFC). Supported values: **10 - 1080** - values in seconds.

Parameter	Access	Description
IMAGE_FLIP	read / write	**Not supported** by **VentusCamera** library.
DDE_MODE	read / write	**Not supported** by **VentusCamera** library.
DDE_LEVEL	read / write	**Not supported** by **VentusCamera** library.
ROI_X0	read / write	**Not supported** by **VentusCamera** library.
ROI_Y0	read / write	**Not supported** by **VentusCamera** library.
ROI_X1	read / write	**Not supported** by **VentusCamera** library.
ROI_Y1	read / write	**Not supported** by **VentusCamera** library.
TEMPERATURE	read only	**Not supported** by **VentusCamera** library.
ALC_GATE	read / write	**Not supported** by **VentusCamera** library.
SENSITIVITY	read / write	**Not supported** by **VentusCamera** library.
CHANGING_MODE	read / write	**Not supported** by **VentusCamera** library.
CHANGING_LEVEL	read / write	**Not supported** by **VentusCamera** library.
CHROMA_LEVEL	read / write	**Not supported** by **VentusCamera** library.
DETAIL	read / write	**Not supported** by **VentusCamera** library.
PROFILE	read / write	**Not supported** by **VentusCamera** library.

Parameter	Access	Description
IS_CONNECTED	read only	Connection status. Connection status shows if the controller has data exchange with equipment. For example, if serial port open but equipment can be not active (no power). In this case connection status shows that controller doesn't have data exchange with equipment (methos will return 0). It the controller has data exchange with equipment the method will return 1. If the controller not initialize the connection status always FALSE. Value: 0 - not connected. 1 - connected.
IS_OPEN	read only	Controller initialization status. Open status shows if the controller initialized or not but doesn't show if controller has communication with equipment. For example, if serial port open but equipment can be not active (no power). In this case open status just shows that the controller has opened serial port. Values: 0 - not open (not initialized), 1 - open (initialized).
TYPE	read / write	**Not supported** by **VentusCamera** library.
CUSTOM_1	read / write	**Not supported** by **VentusCamera** library.
CUSTOM_2	read / write	**Not supported** by **VentusCamera** library.
CUSTOM_3	read / write	**Not supported** by **VentusCamera** library.

LensParams class description

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

LensParams class declaration

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

```
class LensParams
{
public:
    /// Initialization string.
```

```
    std::string initString{"/dev/ttyUSB0;9600;7"};
    /// Zoom position.
    int zoomPos{0};
    /// Hardware zoom position.
    int zoomHwPos{0};
    /// Focus position.
    int focusPos{0};
    /// Hardware focus position.
    int focusHwPos{0};
    /// Iris position.
    int irisPos{0};
    /// Hardware iris position.
    int irisHwPos{0};
    /// Focus mode.
    int focusMode{0};
    /// Filter mode.
    int filterMode{0};
    /// Autofocus ROI top-left corner horizontal position in pixels.
    int afRoiX0{0};
    /// Autofocus ROI top-left corner vertical position in pixels.
    int afRoiY0{0};
    /// Autofocus ROI bottom-right corner horizontal position in pixels.
    int afRoiX1{0};
    /// Autofocus ROI bottom-right corner vertical position in pixels.
    int afRoiY1{0};
    /// Zoom speed.
    int zoomSpeed{50};
    /// Zoom hardware speed.
    int zoomHwSpeed{50};
    /// Maximum zoom hardware speed.
    int zoomHwMaxSpeed{50};
    /// Focus speed.
    int focusSpeed{50};
    /// Focus hardware speed.
    int focusHwSpeed{50};
    /// Maximum focus hardware speed.
    int focusHwMaxSpeed{50};
    /// Iris speed.
    int irisSpeed{50};
    /// Iris hardware speed.
    int irisHwSpeed{50};
    /// Maximum iris hardware speed.
    int irisHwMaxSpeed{50};
    /// Zoom hardware tele limit.
    int zoomHwTeleLimit{65535};
    /// Zoom hardware wide limit.
    int zoomHwWideLimit{0};
    /// Focus hardware far limit.
    int focusHwFarLimit{65535};
    /// Focus hardware near limit.
    int focusHwNearLimit{0};
    /// Iris hardware open limit.
    int irisHwOpenLimit{65535};
    /// Iris hardware close limit.
    int irisHwCloseLimit{0};
    /// Focus factor if it was calculated.
```

```
    float focusFactor{0.0f};
    /// Lens connection status.
    bool isConnected{false};
    /// Focus hardware speed in autofocus mode.
    int afHwSpeed{50};
    /// Timeout for automatic refocus in seconds.
    float focusFactorThreshold{0.0f};
    /// Timeout for automatic refocus in seconds.
    int refocusTimeoutSec{0};
    /// Flag about active autofocus algorithm.
    bool afIsActive{false};
    /// Iris mode.
    int irisMode{0};
    /// ROI width (pixels) for autofocus algorithm.
    int autoAfRoiWidth{150};
    /// ROI height (pixels) for autofocus algorithm.
    int autoAfRoiHeight{150};
    /// Video frame border size (along vertical and horizontal axes).
    int autoAfRoiBorder{100};
    /// AF ROI mode (write/read).
    int afRoiMode{0};
    /// Lens extender mode.
    int extenderMode{0};
    /// Lens stabilization mode.
    int stabiliserMode{0};
    /// Autofocus range.
    int afRange{0};
    /// Current horizontal Field of view, degree.
    float xFovDeg{1.0f};
    /// Current vertical Field of view, degree.
    float yFovDeg{1.0f};
    /// Logging mode.
    int logMode{0};
    /// Lens temperature, degree (read only).
    float temperature{0.0f};
    /// Lens controller initialization status.
    bool isOpen{false};
    /// Lens type. Value depends on implementation.
    int type{0};
    /// Lens custom parameter.
    float custom1{0.0f};
    /// Lens custom parameter.
    float custom2{0.0f};
    /// Lens custom parameter.
    float custom3{0.0f};
    /// List of points to calculate fiend of view.
    std::vector<FovPoint> fovPoints{std::vector<FovPoint>()};

    JSON_READABLE(LensParams, initString, focusMode, filterMode,
                  afRoiX0, afRoiY0, afRoiX1, afRoiY1, zoomHwMaxSpeed,
                  focusHwMaxSpeed, irisHwMaxSpeed, zoomHwTeleLimit,
                  zoomHwWideLimit, focusHwFarLimit, focusHwNearLimit,
                  irisHwOpenLimit, irisHwCloseLimit, afHwSpeed,
                  focusFactorThreshold, refocusTimeoutSec, irisMode,
                  autoAfRoiWidth, autoAfRoiHeight, autoAfRoiBorder,
                  afRoiMode, extenderMode, stabiliserMode, afRange,
```

```
                    logMode, type, custom1, custom2, custom3, fovPoints);

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

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

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

LensParams class fields description is equivalent to LensParam enum description except initString. initString can have 3 different form of initialization string for the device:
- [serial port name];[baudrate];[camera address];[wait data timeout]
- [serial port name];[baudrate];[camera address]
- [serial port name];[baudrate]

When camera address is not given it is set to 1 as default and same for wait data timeout - 100ms.

Serialize lens params

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

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

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

LensParamsMask structure declaration:

```
typedef struct LensParamsMask
{
    bool zoomPos{true};
    bool zoomHwPos{true};
    bool focusPos{true};
```

```
    bool focusHwPos{true};
    bool irisPos{true};
    bool irisHwPos{true};
    bool focusMode{true};
    bool filterMode{true};
    bool afRoiX0{true};
    bool afRoiY0{true};
    bool afRoiX1{true};
    bool afRoiY1{true};
    bool zoomSpeed{true};
    bool zoomHwSpeed{true};
    bool zoomHwMaxSpeed{true};
    bool focusSpeed{true};
    bool focusHwSpeed{true};
    bool focusHwMaxSpeed{true};
    bool irisSpeed{true};
    bool irisHwSpeed{true};
    bool irisHwMaxSpeed{true};
    bool zoomHwTeleLimit{true};
    bool zoomHwWideLimit{true};
    bool focusHwFarLimit{true};
    bool focusHwNearLimit{true};
    bool irisHwOpenLimit{true};
    bool irisHwCloseLimit{true};
    bool focusFactor{true};
    bool isConnected{true};
    bool afHwSpeed{true};
    bool focusFactorThreshold{true};
    bool refocusTimeoutSec{true};
    bool afIsActive{true};
    bool irisMode{true};
    bool autoAfRoiWidth{true};
    bool autoAfRoiHeight{true};
    bool autoAfRoiBorder{true};
    bool afRoiMode{true};
    bool extenderMode{true};
    bool stabiliserMode{true};
    bool afRange{true};
    bool xFovDeg{true};
    bool yFovDeg{true};
    bool logMode{true};
    bool temperature{true};
    bool isOpen{false};
    bool type{true};
    bool custom1{true};
    bool custom2{true};
    bool custom3{true};
} LensParamsMask;
```

Example without parameters mask:

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

Example with parameters mask:

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

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

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

Deserialize lens params

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

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

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

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

Example:

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

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

Read and write lens params to JSON file

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

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

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

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

TestLensParams.json will look like:

```
{
    "lensParams": {
        "afHwSpeed": 93,
        "afRange": 128,
        "afRoiMode": 239,
        "afRoiX0": 196,
        "afRoiX1": 252,
        "afRoiY0": 115,
        "afRoiY1": 101,
```

```
        "autoAfRoiBorder": 70,
        "autoAfRoiHeight": 125,
        "autoAfRoiWidth": 147,
        "custom1": 91.0,
        "custom2": 236.0,
        "custom3": 194.0,
        "extenderMode": 84,
        "filterMode": 49,
        "focusFactorThreshold": 98.0,
        "focusHwFarLimit": 228,
        "focusHwMaxSpeed": 183,
        "focusHwNearLimit": 47,
        "focusMode": 111,
        "fovPoints": [
            {
                "hwZoomPos": 55,
                "xFovDeg": 6.0,
                "yFovDeg": 51.0
            },
            {
                "hwZoomPos": 63,
                "xFovDeg": 249.0,
                "yFovDeg": 33.0
            },
            {
                "hwZoomPos": 4,
                "xFovDeg": 121.0,
                "yFovDeg": 144.0
            },
            {
                "hwZoomPos": 53,
                "xFovDeg": 214.0,
                "yFovDeg": 153.0
            },
            {
                "hwZoomPos": 143,
                "xFovDeg": 15.0,
                "yFovDeg": 218.0
            }
        ],
        "initString": "dfhglsjirhuhjfb",
        "irisHwCloseLimit": 221,
        "irisHwMaxSpeed": 79,
        "irisHwOpenLimit": 211,
        "irisMode": 206,
        "logMode": 216,
        "refocusTimeoutSec": 135,
        "stabiliserMode": 137,
        "type": 125,
        "zoomHwMaxSpeed": 157,
        "zoomHwTeleLimit": 68,
        "zoomHwWideLimit": 251
    }
}
```

CameraParams class description

CameraParams class used for camera controller initialization or to get all actual params. Also **CameraParams** provide structure to write/read params from JSON files (**JSON_READABLE** macro) and provide methos to encode and decode params.

CameraParams Class declaration

CameraParams interface class declared in **Camera.h** file. Class declaration:

```
class CameraParams
{
public:
    /// Initialization string.
    std::string initString{"/dev/ttyUSB0;9600;7"};
    /// Video frame width. Value from 0 to 16384.
    int width{0};
    /// Video frame height Value from 0 to 16384.
    int height{0};
    /// Display menu mode.
    int displayMode{0};
    /// Video output type.
    int videoOutput{0};
    /// Logging mode.
    int logMode{0};
    /// Exposure mode.
    int exposureMode{1};
    /// Exposure time of the camera sensor.
    int exposureTime{0};
    /// White balance mode.
    int whiteBalanceMode{1};
    /// White balance area.
    int whiteBalanceArea{0};
    /// White dynamic range mode.
    int wideDynamicRangeMode{0};
    /// Image stabilization mode.
    int stabilisationMode{0};
    /// ISO sensitivity.
    int isoSensetivity{0};
    /// Scene mode.
    int sceneMode{0};
    /// FPS.
    float fps{0.0f};
    /// Brightness mode.
    int brightnessMode{1};
    /// Brightness. Value 0 - 100%.
    int brightness{0};
    /// Contrast. Value 1 - 100%.
    int contrast{0};
    /// Gain mode.
    int gainMode{1};
```

```
    /// Gain. Value 1 - 100%.
    int gain{0};
    /// Sharpening mode.
    int sharpeningMode{0};
    /// Sharpening. Value 1 - 100%.
    int sharpening{0};
    /// Palette.
    int palette{0};
    /// Analog gain control mode.
    int agcMode{1};
    /// Shutter mode.
    int shutterMode{1};
    /// Shutter position. 0 (full close) - 65535 (full open).
    int shutterPos{0};
    /// Shutter speed. Value: 0 - 100%.
    int shutterSpeed{0};
    /// Digital zoom mode.
    int digitalZoomMode{0};
    /// Digital zoom. Value 1.0 (x1) - 20.0 (x20).
    float digitalZoom{1.0f};
    /// Exposure compensation mode.
    int exposureCompensationMode{0};
    /// Exposure compensation position.
    int exposureCompensationPosition{0};
    /// Defog mode.
    int defogMode{0};
    /// Dehaze mode.
    int dehazeMode{0};
    /// Noise reduction mode.
    int noiseReductionMode{0};
    /// Black and white filter mode.
    int blackAndWhiteFilterMode{0};
    /// Filter mode.
    int filterMode{0};
    /// NUC mode for thermal cameras.
    int nucMode{0};
    /// Auto NUC interval for thermal cameras.
    int autoNucIntervalMsec{0};
    /// Image flip mode.
    int imageFlip{0};
    /// DDE mode.
    int ddeMode{0};
    /// DDE level.
    float ddeLevel{0};
    /// ROI top-left horizontal position, pixels.
    int roiX0{0};
    /// ROI top-left vertical position, pixels.
    int roiY0{0};
    /// ROI bottom-right horizontal position, pixels.
    int roiX1{0};
    /// ROI bottom-right vertical position, pixels.
    int roiY1{0};
    /// Camera temperature, degree.
    float temperature{0.0f};
    /// ALC gate.
    int alcGate{0};
```

```
    /// Sensor sensitivity.
    float sensitivity{0};
    /// Changing mode (day / night).
    int changingMode{0};
    /// Changing level (day / night).
    float changingLevel{0.0f};
    /// Chroma level. Values: 0 - 100%.
    int chromaLevel{0};
    /// Details, enhancement. Values: 0 - 100%.
    int detail{0};
    /// Camera settings profile.
    int profile{0};
    /// Connection status (read only).
    bool isConnected{false};
    /// Open status (read only).
    bool isOpen{false};
    /// Camera type.
    int type{0};
    /// Camera custom param.
    float custom1{0.0f};
    /// Camera custom param.
    float custom2{0.0f};
    /// Camera custom param.
    float custom3{0.0f};

    JSON_READABLE(CameraParams, initString, width, height, displayMode,
                  videoOutput, logMode, exposureMode, exposureTime,
                  whiteBalanceMode, whiteBalanceArea, wideDynamicRangeMode,
                  stabilisationMode, isoSensetivity, sceneMode, fps,
                  brightnessMode, brightness, contrast, gainMode, gain,
                  sharpeningMode, sharpening, palette, agcMode, shutterMode,
                  shutterPos, shutterSpeed, digitalZoomMode, digitalZoom,
                  exposureCompensationMode, exposureCompensationPosition,
                  defogMode, dehazeMode, noiseReductionMode,
                  blackAndWhiteFilterMode, filterMode, nucMode,
                  autoNucIntervalMsec, imageFlip, ddeMode, ddeLevel,
                  roiX0, roiY0, roiX1, roiY1, alcGate, sensitivity,
                  changingMode, changingLevel, chromaLevel, detail,
                  profile, type, custom1, custom2, custom3)

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

    /// Encode params. The method doesn't encode initString.
    bool encode(uint8_t* data, int bufferSize, int& size,
                CameraParamsMask* mask = nullptr);

    /// Decode params. The method doesn't decode initString.
    bool decode(uint8_t* data, int dataSize);
};
```

CameraParams class fields description is equivalent to CameraParam enum description except initString.
Initialization string. initString can have 3 different form of initialization string for the device:
- [serial port name];[baudrate];[camera address];[wait data timeout]
- [serial port name];[baudrate];[camera address]

- [serial port name];[baudrate]
When camera address is not given it is set to 1 as default and same for wait data timeout - 100ms.

Serialize camera params

CameraParams class provides method **encode(...)** to serialize camera params. Serialization of camera params necessary in case when you have to send camera params via communication channels. Method doesn't encode **initString** field. Method provides options to exclude particular parameters from serialization. To do this method inserts binary mask (8 bytes) where each bit represents particular parameter and **decode(...)** method recognizes it. Method declaration:

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

Parameter	Value
data	Pointer to data buffer. Buffer size must be >= 237 bytes.
bufferSize	Data buffer size. Buffer size must be >= 237 bytes.
size	Size of encoded data.
mask	Parameters mask - pointer to **CameraParamsMask** structure. **CameraParamsMask** (declared in Camera.h file) determines flags for each field (parameter) declared in CameraParams class. If the user wants to exclude any parameters from serialization, he can put a pointer to the mask. If the user wants to exclude a particular parameter from serialization, he should set the corresponding flag in the **CameraParamsMask** structure.

Returns: TRUE if params encoded (serialized) or FALSE if not.

CameraParamsMask structure declaration:

```
typedef struct CameraParamsMask
{
    bool width{true};
    bool height{true};
    bool displayMode{true};
    bool videoOutput{true};
    bool logMode{true};
    bool exposureMode{true};
    bool exposureTime{true};
    bool whiteBalanceMode{true};
    bool whiteBalanceArea{true};
    bool wideDynamicRangeMode{true};
    bool stabilisationMode{true};
    bool isoSensetivity{true};
    bool sceneMode{true};
    bool fps{true};
    bool brightnessMode{true};
    bool brightness{true};
    bool contrast{true};
    bool gainMode{true};
```

```
    bool gain{true};
    bool sharpeningMode{true};
    bool sharpening{true};
    bool palette{true};
    bool agcMode{true};
    bool shutterMode{true};
    bool shutterPos{true};
    bool shutterSpeed{true};
    bool digitalZoomMode{true};
    bool digitalZoom{true};
    bool exposureCompensationMode{true};
    bool exposureCompensationPosition{true};
    bool defogMode{true};
    bool dehazeMode{true};
    bool noiseReductionMode{true};
    bool blackAndWhiteFilterMode{true};
    bool filterMode{true};
    bool nucMode{true};
    bool autoNucIntervalMsec{true};
    bool imageFlip{true};
    bool ddeMode{true};
    bool ddeLevel{true};
    bool roiX0{true};
    bool roiY0{true};
    bool roiX1{true};
    bool roiY1{true};
    bool temperature{true};
    bool alcGate{true};
    bool sensitivity{true};
    bool changingMode{true};
    bool changingLevel{true};
    bool chromaLevel{true};
    bool detail{true};
    bool profile{true};
    bool isConnected{true};
    bool isOpen{true};
    bool type{true};
    bool custom1{true};
    bool custom2{true};
    bool custom3{true};
} CameraParamsMask;
```

Example without parameters mask:

```
// Encode data.
CameraParams in;
in.profile = 10;
uint8_t data[1024];
int size = 0;
in.encode(data, 1024, size);
cout << "Encoded data size: " << size << " bytes" << endl;
```

Example with parameters mask:

```
// Prepare params.
CameraParams in;
in.profile = 3;

// Prepare mask.
CameraParamsMask mask;
mask.profile = false; // Exclude profile. Others by default.

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

Deserialize camera params

CameraParams class provides method **decode(...)** to deserialize camera params (fields of CameraParams class, see Table 4). Deserialization of camera params necessary in case when you need to receive params via communication channels. Method automatically recognizes which parameters were serialized by **encode(...)** method. Method doesn't decode **initString** field. Method declaration:

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

Parameter	Value
data	Pointer to data buffer with serialized camera params.
dataSize	Size of command data.

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

Example:

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

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

Read and write camera params to JSON file

Camera 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:

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

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

TestCameraParams.json will look like:

```
{
    "cameraParams": {
        "agcMode": 252,
        "alcGate": 125,
        "autoNucIntervalMsec": 47,
        "blackAndWhiteFilterMode": 68,
        "brightness": 67,
        "brightnessMode": 206,
        "changingLevel": 84.0,
        "changingMode": 239,
        "chromeLevel": 137,
        "contrast": 65,
        "custom1": 216.0,
        "custom2": 32.0,
        "custom3": 125.0,
        "ddeLevel": 25,
        "ddeMode": 221,
        "defogMode": 155,
        "dehazeMode": 239,
        "detail": 128,
        "digitalZoom": 47.0,
        "digitalZoomMode": 157,
        "displayMode": 2,
        "exposureCompensationMode": 213,
        "exposureCompensationPosition": 183,
        "exposureMode": 192,
        "exposureTime": 16,
        "filterMode": 251,
        "fps": 19.0,
        "gain": 111,
        "gainMode": 130,
        "height": 219,
        "imageFlip": 211,
        "initString": "dfhglsjirhuhjfb",
```

```
        "isoSensetivity": 32,
        "logMode": 252,
        "noiseReductionMode": 79,
        "nucMode": 228,
        "palette": 115,
        "profile": 108,
        "roiX0": 93,
        "roiX1": 135,
        "roiY0": 98,
        "roiY1": 206,
        "sceneMode": 195,
        "sensitivity": 70.0,
        "sharpening": 196,
        "sharpeningMode": 49,
        "shutterMode": 101,
        "shutterPos": 157,
        "shutterSpeed": 117,
        "stabilisationMode": 170,
        "type": 55,
        "videoOutput": 18,
        "whiteBalanceArea": 236,
        "whiteBalanceMode": 30,
        "wideDynamicRangeMode": 21,
        "width": 150
    }
}
```


Build and connect to your project

Typical commands to build **VentusCamera** library:

```
cd VentusCamera
git submodule update --init --recursive
mkdir build
cd build
cmake ..
make
```

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

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

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

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

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(${PARENT}_SUBMODULE_VENTUS_CAMERA                    ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_VENTUS_CAMERA)
    SET(${PARENT}_VENTUS_CAMERA                          ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_VENTUS_CAMERA_TEST                     OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_VENTUS_CAMERA_EXAMPLE                  OFF CACHE BOOL "" FORCE)
endif()

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

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

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

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} VentusCamera)
```

Done!

Simple example

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

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

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

    while (true)
    {
        // Main dialog.
        int option = -1;
        std::cout << "Options (1:Zoom tele, 2:Zoom wide, 3:Zoom stop), " <<
        "4:Brightness+1, 5:Brightness-1 : ";
        std::cin >> option;

        // Get all camera params.
        cr::camera::CameraParams cameraParams;
        controller.getParams(cameraParams);

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

        switch (option)
```

```
        {
        case 1: // Zoom tele.
            controller.executeCommand(cr::lens::LensCommand::ZOOM_TELE);
            break;
        case 2: // Zoom wide.
            controller.executeCommand(cr::lens::LensCommand::ZOOM_WIDE);
            break;
        case 3: // Zoom stop.
            controller.executeCommand(cr::lens::LensCommand::ZOOM_STOP);
            break;
        case 4:
            controller.setParam(cr::camera::CameraParam::BRIGHTNESS,
            cameraParams.brightness + 1);
            break;
        case 5:
            controller.setParam(cr::camera::CameraParam::BRIGHTNESS,
            cameraParams.brightness - 1);
            break;
        default:
            break;
        }
    }
    return 1;
}
```