HitachiCamera C++ library

v2.0.1

Table of contents

Overview

The **HitachiCamera** C++ library is a software controller for Hitachi HD-SDI Camera Series. The **HitachiCamera** library inherits Camera interface. It includes source code of libraries: Camera interface library (provides interface and data structures to control cameras, Apache 2.0 license), Logger logging library (provides function to print log information in console and files, Apache 2.0 license), SerialPort library (provides functions to work with serial ports, Apache 2.0 license). The **HitachiCamera** library provides simple interface to be integrated in any C++ projects. The library repository (folder) provided by source code and doesn't have third-party dependencies to be specially installed in OS. It developed with C++17 standard and compatible with Linux and Windows.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	30.05.2023	First version
2.0.0	20.10.2023	- New Camera interface
2.0.1	21.02.2024	- Camera interface updated. - Examples added. - Documentation updated.

Library files

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

```
CMakeLists.txt ------------------ Main CMake file of the library.
3rdparty ------------------------ Folder with third-party libraries.
    CMakeLists.txt -------------- CMake file to include third-party libraries.
    Camera ---------------------- Folder of the Camera library source code.
    Logger ---------------------- Folder of the Logger library source code.
    SerialPort ------------------ Folder of the SerialPort library source code.
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.
src ----------------------------- Folder with source code of the library.
    CMakeLists.txt -------------- CMake file of the library.
    HitachiCamera.cpp ----------- Source code file of the library.
```

```
    HitachiCamera.h ------------- Header file which includes HitachiCamera class
declaration.
    HitachiCameraVersion.h ------ Header file which includes version of the library.
    HitachiCameraVersion.h.in --- CMake service file to generate version file.
```

HitachiCamera class description

HitachiCamera Class declaration

The **HitachiCamera** interface class declared in **HitachiCamera.h** file. The HitachiCamera class inherits Camera interfaces. Class declaration:

```
class HitachiCamera : public cr::camera::Camera
{
public:

    /// Class constructor.
    HitachiCamera();

    /// Class destructor.
    ~HitachiCamera();

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

    /// Init camera controller.
    bool openCamera(std::string initString) override;

    /// Init camera controller by parameters class.
    bool initCamera(CameraParams& params) override;

    /// Close camera serial port.
    void closeCamera() override;

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

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

    /// Set the camera controller param.
    bool setParam(CameraParam id, float value) override;

    /// Get the camera controller param.
    float getParam(CameraParam id) override;

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

    /// Execute camera controller action command.
```

```
    bool executeCommand(CameraCommand id) override;

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

getVersion method

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

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

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

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

Console output:

```
HitachiCamera class version: 2.0.1
```

openCamera method

The **openCamera(...)** opens serial port to communicate with Hitachi camera. Method declaration:

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

Parameter	Value
initString	Initialization string. Example of camera controller initialization which user serial port: "/dev/ttyUSB0;9600" ("/dev/ttyUSB0" - serial port name, "9600" - baudrate).

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

initCamera method

The **initCamera(...)** initializes controller and sets camera params (Camera interface). The method will call openCamera(...) method and after will set given camera params. Method declaration:

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

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

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

closeCamera method

The **closeCamera()** method closes serial port. Method declaration:

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

isCameraOpen method

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

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

Returns: TRUE is the controller initialized (serial port open) 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 method isCameraOpen(...) will return TRUE but **isCameraConnected()** method will return FALSE. The HitachiCamera library checks connection status every time when send command to camera. Method declaration:

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

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

setParam method

The **setParam(...)** method sets new camera controller parameter value. Method declaration:

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

Parameter	Description
id	Camera controller parameter ID according to CameraParam enum.
value	Parameter value. Value depends on parameter ID.

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

getParam method

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

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

Parameter	Description
id	Camera controller parameter ID according according to CameraParam enum.

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

getParams method

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

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

Parameter	Description
params	Reference to CameraParams class object.

executeCommand method

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

```
bool executeCommand(CameraCommand id) override;
```

Parameter	Description
id	Camera controller command ID according to CameraCommand enum.

Returns: TRUE is the command was executed or FALSE if not.

decodeAndExecuteCommand method

The **decodeAndExecuteCommand(...)** method decodes and executes command on controller side. Method will decode commands which encoded by encodeCommand(...) and encodeSetParamCommand(...) methods of Camera interface class. If command decoded the method will call setParam(...) or executeCommand(...) methods for camera 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 data.
size	Size of command. Must be 11 bytes for SET_PARAM and 7 bytes for COMMAND.

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

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.
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.

Byte	Value	Description
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.

Byte	Value	Description
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

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 2 - Camera commands description.

Command	Description
RESTART	**Not supported** by HitachiCamera library.
NUC	**Not supported** by HitachiCamera library.
APPLY_PARAMS	**Not supported** by HitachiCamera library.
SAVE_PARAMS	**Not supported** by HitachiCamera library.
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	**Not supported** by HitachiCamera library.
DEFREEZE	**Not supported** by HitachiCamera 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 3 - Camera params description.

Parameter	Access	Description
WIDTH	read / write	**Not supported** by HitachiCamera library.
HEIGHT	read / write	**Not supported** by HitachiCamera library.
DISPLAY_MODE	read / write	Display mode. Values: **0** - Menu Off **1** - Menu On
VIDEO_OUTPUT	read / write	**Not supported** by HitachiCamera library.
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 HitachiCamera library.
EXPOSURE_TIME	read / write	**Not supported** by HitachiCamera library.
WHITE_BALANCE_MODE	read / write	**Not supported** by HitachiCamera library.
WHITE_BALANCE_AREA	read / write	**Not supported** by HitachiCamera library.
WHITE_DINAMIC_RANGE_MODE	read / write	**Not supported** by HitachiCamera library.

Parameter	Access	Description
STABILIZATION_MODE	read / write	**Not supported** by HitachiCamera library.
ISO_SENSITIVITY	read / write	**Not supported** by HitachiCamera library.
SCENE_MODE	read / write	**Not supported** by HitachiCamera library.
FPS	read / write	**Not supported** by HitachiCamera library.
BRIGHTNESS_MODE	read / write	**Not supported** by HitachiCamera library.
BRIGHTNESS	read / write	**Not supported** by HitachiCamera library.
CONTRAST	read / write	**Not supported** by HitachiCamera library.
GAIN_MODE	read / write	Gain mode. Values: **0** - Off **1** - AUTOx2 **2** - AUTOx4 **3** - AUTOx6 **4** - AUTOx8 **5** - AUTOx10 **6** - AUTOx12 **7** - AUTOx16 **8** - AUTOx32
GAIN	read / write	**Not supported** by HitachiCamera library.
SHARPENING_MODE	read / write	**Not supported** by HitachiCamera library.
SHARPENING	read / write	**Not supported** by HitachiCamera library.
PALETTE	read / write	**Not supported** by HitachiCamera library.

Parameter	Access	Description
AGC_MODE	read / write	AGC mode. Values: **0** - Off **1** - AUTOx2 **2** - AUTOx4 **3** - AUTOx6 **4** - AUTOx8 **5** - AUTOx10 **6** - AUTOx12 **7** - AUTOx16 **8** - AUTOx32
SHUTTER_MODE	read / write	**Not supported** by HitachiCamera library.
SHUTTER_POSITION	read / write	**Not supported** by HitachiCamera library.
SHUTTER_SPEED	read / write	**Not supported** by HitachiCamera library.
DIGITAL_ZOOM_MODE	read / write	**Not supported** by HitachiCamera library.
DIGITAL_ZOOM	read only	**Not supported** by HitachiCamera library.
EXPOSURE_COMPENSATION_MODE	read only	**Not supported** by HitachiCamera library.
EXPOSURE_COMPENSATION_POSITION	read / write	**Not supported** by HitachiCamera library.
DEFOG_MODE	read / write	Defog mode. Values: **0** - Off **1** - Low **2** - Medium **3** - High
DEHAZE_MODE	read / write	**Not supported** by HitachiCamera library.
NOISE_REDUCTION_MODE	read / write	**Not supported** by HitachiCamera library.
BLACK_WHITE_FILTER_MODE	read only	Black white filter mode. Values: **0** - Off **1** - On **2** - AutoHigh **3** - AutoMed **4** - AutoLow

Parameter	Access	Description
FILTER_MODE	read / write	**Not supported** by HitachiCamera library.
NUC_MODE	read / write	**Not supported** by HitachiCamera library.
AUTO_NUC_INTERVAL	read / write	**Not supported** by HitachiCamera library.
IMAGE_FLIP	read / write	**Not supported** by HitachiCamera library.
DDE_MODE	read / write	**Not supported** by HitachiCamera library.
DDE_LEVEL	read / write	**Not supported** by HitachiCamera library.
ROI_X0	read / write	**Not supported** by HitachiCamera library.
ROI_Y0	read / write	**Not supported** by HitachiCamera library.
ROI_X1	read / write	**Not supported** by HitachiCamera library.
ROI_Y1	read / write	**Not supported** by HitachiCamera library.
TEMPERATURE	read only	**Not supported** by HitachiCamera library.
ALC_GATE	read / write	ALC gate mode. Values: **0** - Off **1** - On **2** - Zone1 **3** - Zone2 **4** - Zone3 **5** - Zone4 **6** - Zone5 **7** - Zone6 **8** - Zone7 **9** - Zone8 **10** - Zone9
SENSITIVITY	read / write	**Not supported** by HitachiCamera library.
CHANGING_MODE	read / write	**Not supported** by HitachiCamera library.

Parameter	Access	Description
CHANGING_LEVEL	read / write	**Not supported** by HitachiCamera library.
CHROMA_LEVEL	read / write	**Not supported** by HitachiCamera library.
DETAIL	read / write	**Not supported** by HitachiCamera library.
PROFILE	read / write	Profile. Values: **1 - 4**
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 HitachiCamera library.
CUSTOM_1	read / write	**Not supported** by HitachiCamera library.
CUSTOM_2	read / write	**Not supported** by HitachiCamera library.
CUSTOM_3	read / write	**Not supported** by HitachiCamera library.

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"};
    /// 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);
};
```

Table 4 - CameraParams class fields description is equivalent to **CameraParam** enum description.

Field	type	Description
initString	string	Initialization string contains full serial port name, baudrate separated by "**;**". Example: **"/dev/ttyUSB0;9600"**.
width	int	**Not supported** by HitachiCamera class.
height	int	**Not supported** by HitachiCamera class.
displayMode	int	Display mode. Values: **0** - Menu Off **1** - Menu On
videoOutput	int	**Not supported** by HitachiCamera class.
logMode	int	Logging mode. Values: **0** - Disable **1** - Only file **2** - Only terminal (console) **3** - File and terminal
exposureMode	int	**Not supported** by HitachiCamera class.
exposureTime	int	**Not supported** by HitachiCamera class.
whiteBalanceMode	int	**Not supported** by HitachiCamera class.
whiteBalanceArea	int	**Not supported** by HitachiCamera class.
wideDynamicRangeMode	int	**Not supported** by HitachiCamera class.
stabilisationMode	int	**Not supported** by HitachiCamera class.
isoSensetivity	int	**Not supported** by HitachiCamera class.
sceneMode	int	**Not supported** by HitachiCamera class.
fps	float	**Not supported** by HitachiCamera class.
brightnessMode	int	**Not supported** by HitachiCamera class.
brightness	int	**Not supported** by HitachiCamera class.
contrast	int	**Not supported** by HitachiCamera class.
gainMode	int	Gain mode. Values: **0** - Off **1** - AUTOx2 **2** - AUTOx4 **3** - AUTOx6 **4** - AUTOx8 **5** - AUTOx10 **6** - AUTOx12 **7** - AUTOx16 **8** - AUTOx32

Field	type	Description
gain	int	**Not supported** by HitachiCamera class.
sharpeningMode	int	**Not supported** by HitachiCamera class.
sharpening	int	**Not supported** by HitachiCamera class.
palette	int	**Not supported** by HitachiCamera class.
agcMode	int	AGC mode. Values: **0** - Off **1** - AUTOx2 **2** - AUTOx4 **3** - AUTOx6 **4** - AUTOx8 **5** - AUTOx10 **6** - AUTOx12 **7** - AUTOx16 **8** - AUTOx32
shutterMode	int	**Not supported** by HitachiCamera class.
shutterPos	int	**Not supported** by HitachiCamera class.
shutterSpeed	int	**Not supported** by HitachiCamera class.
digitalZoomMode	int	**Not supported** by HitachiCamera class.
digitalZoom	float	**Not supported** by HitachiCamera class.
exposureCompensationMode	int	**Not supported** by HitachiCamera class.
exposureCompensationPosition	int	**Not supported** by HitachiCamera class.
defogMode	int	Defog mode. Values: **0** - Off **1** - Low **2** - Medium **3** - High
dehazeMode	int	**Not supported** by HitachiCamera class.
noiseReductionMode	int	**Not supported** by HitachiCamera class.
blackAndWhiteFilterMode	int	Black white filter mode. Values: **0** - Off **1** - On **2** - AutoHigh **3** - AutoMed **4** - AutoLow
filterMode	int	**Not supported** by HitachiCamera class.
nucMode	int	**Not supported** by HitachiCamera class.

Field	type	Description
autoNucIntervalMsec	int	**Not supported** by HitachiCamera class.
imageFlip	int	**Not supported** by HitachiCamera class.
ddeMode	int	**Not supported** by HitachiCamera class.
ddeLevel	float	**Not supported** by HitachiCamera class.
roiX0	int	**Not supported** by HitachiCamera class.
roiY0	int	**Not supported** by HitachiCamera class.
roiX1	int	**Not supported** by HitachiCamera class.
roiY1	int	**Not supported** by HitachiCamera class.
temperature	float	**Not supported** by HitachiCamera class.
alcGate	int	ALC gate mode. Values: **0** - Off **1** - On **2** - Zone1 **3** - Zone2 **4** - Zone3 **5** - Zone4 **6** - Zone5 **7** - Zone6 **8** - Zone7 **9** - Zone8 **10** - Zone9
sensitivity	float	**Not supported** by HitachiCamera class.
changingMode	int	**Not supported** by HitachiCamera class.
changingLevel	float	**Not supported** by HitachiCamera class.
chromaLevel	int	**Not supported** by HitachiCamera class.
detail	int	**Not supported** by HitachiCamera class.
profile	int	Profile. Values: **1 - 4**.
isConnected	bool	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.

Field	type	Description
isOpen	bool	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	int	**Not supported** by HitachiCamera class.
custom1	float	**Not supported** by HitachiCamera class.
custom2	float	**Not supported** by HitachiCamera class.
custom3	float	**Not supported** by HitachiCamera class.

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

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 **HitachiCamera** library:

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

If you want connect **HitachiCamera** 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 **HitachiCamera** repository there. New structure of your repository:

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

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(${PARENT}_SUBMODULE_HITACHI_CAMERA                  ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_HITACHI_CAMERA)
    SET(${PARENT}_HITACHI_CAMERA                        ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_HITACHI_CAMERA_TEST                   OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_HITACHI_CAMERA_EXAMPLE                OFF CACHE BOOL "" FORCE)
endif()

###############################################################################
## INCLUDING SUBDIRECTORIES
## Adding subdirectories according to the 3rd-party configuration
###############################################################################
```

```
if (${PARENT}_SUBMODULE_HITACHI_CAMERA)
    add_subdirectory(HitachiCamera)
endif()
```

File **3rdparty/CMakeLists.txt** adds folder **HitachiCamera** 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
    HitachiCamera
```

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} HitachiCamera)
```

Done!

Simple example

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

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

int main(void)
{
    // Init camera controller.
    cr::camera::HitachiCamera controller;
    if (!controller.openCamera("/dev/ttyUSB0;9600"))
        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);

        switch (option)
        {
        case 1:
            controller.setParam(cr::camera::CameraParam::BRIGHTNESS,
            cameraParams.brightness + 1);
            break;
        case 2:
            controller.setParam(cr::camera::CameraParam::BRIGHTNESS,
            cameraParams.brightness - 1);
            break;
        default:
            break;
        }
    }
    return 1;
}
```