Camera interface C++ library

v2.5.0

Table of contents

Overview

Camera C++ library provides standard interface as well defines data structures and rules for different camera controllers. **Camera** interface class doesn't do anything, just provides interface and provides methods to encode/decode commands and encode/decode params. Different camera controller classes inherit interface form **Camera** C++ class. **Camera.h** file contains list of data structures (**CameraCommand enum**, **CameraParam enum** and **CameraParams class**) and **Camera** class declaration. **CameraCommand enum** contains IDs of commands supported by **Camera** class. **CameraParam enum** contains IDs of params supported by **Camera** class. All camera controllers should include params and commands listed in **Camera.h** file. Camera interface class depends on **ConfigReader** library (provides methods to read/write JSON config files).

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	05.05.2023	First version
1.1.0	08.05.2023	- Added new parameter.
1.2.0	10.05.2023	- Parameters list changed.
2.0.0	30.06.2023	- Added new parameters. - Added new methods to encode/decode commands. - Added new class CameraParams to store camera parameters. - Added license. - Repository made public.
2.1.0	12.07.2023	- Added CameraParamsMask structure. - Names of params updated. - Updated encode(...) and decode(...) methods of CameraParams.
2.2.0	20.09.2023	- Updated encode(...) and decode(...) methods of CameraParams. - Added decodeAndExecuteCommand(...) method. - Added example of camera controller implementation.
2.2.1	22.09.2023	- Fixed mistakes in documentation.
2.3.0	26.09.2023	- Changed getParams method return type.
2.4.0	13.12.2023	- Virtual destructor added.
2.5.0	08.01.2024	- Name of parameters updated.

Library files

The **Camera** library is a CMake project. Library files:

```
CMakeLists.txt ------------------- Main CMake file of the library.
3rdparty ------------------------- Folder with third-party libraries.
    CMakeLists.txt --------------- CMake file which includes third-party libraries.
    ConfigReader ----------------- Source code of the ConfigReader library.
example -------------------------- Folder with simple example of VCodecImsdk usage.
    CMakeLists.txt --------------- CMake file for example custom camera class.
    CustomCamera.cpp ------------- Source code file of the CustomCamera class.
    CustomCamera.h --------------- Header with CustomCamera class declaration.
    CustomCameraVersion.h -------- Header file which includes CustomCamera class version.
    CustomCameraVersion.h.in ----- CMake service file to generate version file.
test ----------------------------- Folder with codec test application.
    CMakeLists.txt --------------- CMake file for codec test application.
    main.cpp --------------------- Source code file of Camera class test application.
src ------------------------------ Folder with source code of the library.
    CMakeLists.txt --------------- CMake file of the library.
    Camera.cpp ------------------- Source code file of the library.
    Camera.h --------------------- Header file which includes Camera class declaration.
    CameraVersion.h -------------- Header file which includes version of the library.
    CameraVersion.h.in ----------- CMake service file to generate version file.
```

Camera interface class description

Class declaration

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

```
class Camera
{
public:

    /// Class destructor.
    virtual ~Camera();

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

    /// Open camera controller.
    virtual bool openCamera(std::string initString) = 0;

    /// Init camera controller by structure.
    virtual bool initCamera(CameraParams& params) = 0;

    /// Close camera connection.
    virtual void closeCamera() = 0;
```

```
    /// Get camera open status.
    virtual bool isCameraOpen() = 0;

    /// Get camera open status.
    virtual bool isCameraConnected() = 0;

    /// Set the camera controller param.
    virtual bool setParam(CameraParam id, float value) = 0;

    /// Get the camera controller param.
    virtual float getParam(CameraParam id) = 0;

    /// Get the camera controller params structure.
    virtual void getParams(CameraParams& params) = 0;

    /// Execute camera controller command.
    virtual bool executeCommand(CameraCommand id) = 0;

    /// Encode set param command.
    static void encodeSetParamCommand(
            uint8_t* data, int& size, CameraParam id, float value);

    /// Encode command.
    static void encodeCommand(
            uint8_t* data, int& size, CameraCommand id);

    /// Decode command.
    static int decodeCommand(uint8_t* data,
                             int size,
                             CameraParam& paramId,
                             CameraCommand& commandId,
                             float& value);

    /// Decode and execute command.
    virtual bool decodeAndExecuteCommand(uint8_t* data, int size) = 0;
};
```

getVersion method

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

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

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

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

Console output:

```
Camera class version: 2.4.0
```

openCamera method

openCamera(...) method initializes camera controller. This method can be used instead of **initCamera(...)** method. Method declaration:

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

Parameter	Value
initString	Initialization string. Particular camera controller can have unique initialization string format. But it is recommended to use '**;**' symbol to divide part of initialization string. Recommended camera controller initialization string for controllers which uses serial port: **"/dev/ttyUSB0;9600;100"** ("/dev/ttyUSB0" - serial port name, "9600" - baudrate, "100" - serial port read timeout).

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

initCamera method

initCamera(...) method initializes camera controller by list of parameters. This method can be used instead of **openCamera(...)** method (**CameraParams** class includes **initString**) when you need initialize camera controller with not default parameters values. Method declaration:

```
virtual bool initCamera(CameraParams& params) = 0;
```

Parameter	Value
params	Parameters (**CameraParams** class). CameraParams class includes initString wich used in **openCamera(...)** method. See description of **CameraParams** class.

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

closeCamera method

closeCamera() method designed to close connection to camera. Method declaration:

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

isCameraOpen method

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

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

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

isCameraConnected method

isCameraConnected() method returns camera connection status. Connection status shows if the camera controller has data exchange with camera equipment. For example, if camera has serial port and camera controller connected to serial port (opens serial port file in OS) but camera may be not active (no power). In this case connection status shows that camera controller doesn't have data exchange with camera equipment (methos will return FALSE). If camera controller has data exchange with camera equipment the method will return TRUE. If camera controller not initialize the connection status always FALSE. Method declaration:

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

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

setParam method

setParam(...) method sets new camera controller parameters value. The particular implementation of the camera controller must provide thread-safe **setParam(...)** method call. This means that the **setParam(...)** method can be safely called from any thread. Method declaration:

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

Parameter	Description
id	Camera controller parameter ID according to CameraParam enum (see description of **CameraParam** enum).
value	Camera controller parameter value. Value depends on parameter ID (see description of **CameraParam** enum).

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

getParam method

getParam(...) method returns controller parameter value. The particular implementation of the camera controller must provide thread-safe **getParam(...)** method call. This means that the **getParam(...)** method can be safely called from any thread. Method declaration:

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

Parameter	Description
id	Camera controller parameter ID according to CameraParam enum (see description of CameraParam enum).

Returns: parameter value or **-1** of the parameters doesn't exist in particular camera controller.

getParams method

getParams(...) method designed to obtain camera parameters. The particular implementation of the camera controller must provide thread-safe **getParams(...)** method call. This means that the **getParams(...)** method can be safely called from any thread. Method declaration:

```
virtual void getParams(CameraParams& params) = 0;
```

Parameter	Description
params	Reference to CameraParams object to store params.

executeCommand method

executeCommand(...) method executes camera controller command. The particular implementation of the camera controller must provide thread-safe **executeCommand(...)** method call. This means that the **executeCommand(...)** method can be safely called from any thread. Method declaration:

```
virtual bool executeCommand(CameraCommand id) = 0;
```

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

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

encodeSetParamCommand method

encodeSetParamCommand(...) static method 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.
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

encodeCommand(...) static method 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

decodeCommand(...) static method 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.

decodeAndExecuteCommand method

decodeAndExecuteCommand(...) method decodes and executes command on camera controller side. The particular implementation of the camera controller must provide thread-safe **decodeAndExecuteCommand(...)** method call. This means that the **decodeAndExecuteCommand(...)** method can be safely called from any thread. Method declaration:

```
virtual bool decodeAndExecuteCommand(uint8_t* data, int size) = 0;
```

Parameter	Description
data	Pointer to input command.
size	Size of command. Must be 11 bytes for SET_PARAM or 7 bytes for COMMAND.

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

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. Some commands may be unsupported by particular camera controller.

Command	Description
RESTART	Restart camera controller.
NUC	Do NUC (Calibration). For thermal cameras.
APPLY_PARAMS	Apply settings.
SAVE_PARAMS	Save params in camera memory.
MENU_ON	Menu on.

Command	Description
MENU_OFF	Menu off.
MENU_SET	Menu set.
MENU_UP	Menu move up.
MENU_DOWN	Menu move down.
MENU_LEFT	Menu move left.
MENU_RIGHT	Menu move right.
FREEZE	Freeze image.
DEFREEZE	Defreeze image.

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. Value depends on implementation but it is recommended
    /// to keep default values: 0 - Off. 1 - On.
    DISPLAY_MODE,
    /// Video output type. Value depends on implementation.
    VIDEO_OUTPUT,
    /// Logging mode. Values: 0 - Disable, 1 - Only file,
    /// 2 - Only terminal (console), 3 - File and terminal.
    LOG_MODE,
    /// Exposure mode. Value depends on implementation but it is recommended to
    /// keep default values: 0 - Manual, 1 - Auto (default),
    /// 2 - Shutter priority, 3 - Aperture priority.
    EXPOSURE_MODE,
    /// Exposure time of the camera sensor. The exposure time is limited by the
    /// frame interval. Camera controller should interpret the values as 100 µs
    /// units, where the value 1 stands for 1/10000th of a second, 10000 for
    /// 1 second and 100000 for 10 seconds.
    EXPOSURE_TIME,
    /// White balance mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual, 1 - Auto.
    WHITE_BALANCE_MODE,
    /// White balance area. Value depends on implementation.
    WHITE_BALANCE_AREA,
    /// White dynamic range mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Off, 1 - On.
    WIDE_DYNAMIC_RANGE_MODE,
    /// Image stabilization mode. Value depends on implementation but it is
```

```
    /// recommended to keep default values: 0 - Off, 1 - On.
    STABILIZATION_MODE,
    /// ISO sensitivity. Value depends on implementation.
    ISO_SENSITIVITY,
    /// Scene mode. Value depends on implementation.
    SCENE_MODE,
    /// FPS.
    FPS,
    /// Brightness mode. Value depends on implementation but it is recommended
    /// to keep default values: 0 - Manual, 1 - Auto.
    BRIGHTNESS_MODE,
    /// Brightness. Value 0 - 100%.
    BRIGHTNESS,
    /// Contrast. Value 1 - 100%.
    CONTRAST,
    /// Gain mode. Value depends on implementation but it is recommended to keep
    /// default values: 0 - Manual, 1 - Auto.
    GAIN_MODE,
    /// Gain. Value 1 - 100%.
    GAIN,
    /// Sharpening mode. Value depends on implementation but it is recommended
    /// to keep default values: 0 - Manual, 1 - Auto.
    SHARPENING_MODE,
    /// Sharpening. Value 1 - 100%.
    SHARPENING,
    /// Palette. Value depends on implementation but it is recommended to keep
    /// default values for thermal cameras: 0 - White hot, 1 - Black hot.
    PALETTE,
    /// Analog gain control mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual, 1 - Auto.
    AGC_MODE,
    /// Shutter mode. Value depends on implementation but it is recommended to
    /// keep default values: 0 - Manual, 1 - Auto.
    SHUTTER_MODE,
    /// Shutter position. 0 (full close) - 65535 (full open).
    SHUTTER_POSITION,
    /// Shutter speed. Value: 0 - 100%.
    SHUTTER_SPEED,
    /// Digital zoom mode. Value depends on implementation but it is recommended
    /// to keep default values: 0 - Off, 1 - On.
    DIGITAL_ZOOM_MODE,
    /// Digital zoom. Value 1.0 (x1) - 20.0 (x20).
    DIGITAL_ZOOM,
    /// Exposure compensation mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Off, 1 - On.
    EXPOSURE_COMPENSATION_MODE,
    /// Exposure compensation position. Value depends on particular camera
    /// controller.
    EXPOSURE_COMPENSATION_POSITION,
    /// Defog mode. Value depends on implementation but it is recommended to
    /// keep default values: 0 - Off, 1 - On.
    DEFOG_MODE,
    /// Dehaze mode. Value depends on implementation but it is recommended to
    /// keep default values: 0 - Off, 1 - On.
    DEHAZE_MODE,
```

```
    /// Noise reduction mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Off, 1 - 2D, 3 - 3D.
    NOISE_REDUCTION_MODE,
    /// Black and white filter mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Off, 1 - On.
    BLACK_WHITE_FILTER_MODE,
    /// Filter mode. Value depends on implementation.
    FILTER_MODE,
    /// NUC mode for thermal cameras. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual, 1 - Auto.
    NUC_MODE,
    /// Auto NUC interval for thermal cameras. Value in milliseconds
    /// from 0 (Off) to 100000.
    AUTO_NUC_INTERVAL_MSEC,
    /// Image flip mode. Value depends on implementation but it is recommended
    /// to keep default values: 0 - Off, 1 - Horizontal, 2 - Vertical,
    /// 3 - Horizontal and vertical.
    IMAGE_FLIP,
    /// DDE mode. Value depends on implementation but it is recommended to keep
    /// default values: 0 - Off, 1 - On.
    DDE_MODE,
    /// DDE level. Value depends on implementation.
    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. Value depends on implementation.
    ALC_GATE,
    /// Sensor sensitivity. Value depends on implementation.
    SENSITIVITY,
    /// Changing mode (day / night). Value depends on implementation.
    CHANGING_MODE,
    /// Changing level (day / night). Value depends on implementation.
    CHANGING_LEVEL,
    /// Chroma level. Values: 0 - 100%.
    CHROMA_LEVEL,
    /// Details, enhancement. Values: 0 - 100%.
    DETAIL,
    /// Camera settings profile. Value depends on implementation.
    PROFILE,
    /// Connection status (read only). Shows if we have respone 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. Value depends on implementation.
    TYPE,
```

```
    /// Camera custom param. Value depends on implementation.
    CUSTOM_1,
    /// Camera custom param. Value depends on implementation.
    CUSTOM_2,
    /// Camera custom param. Value depends on implementation.
    CUSTOM_3
};
```

Table 3 - Camera params description. Some params may be unsupported by particular camera controller.

Parameter	Access	Description
WIDTH	read / write	Video frame width. Value from 0 to 16384.
HEIGHT	read / write	Video frame height Value from 0 to 16384.
DISPLAY_MODE	read / write	Display menu mode. Value depends on implementation but it is recommended to keep default values: 0 - Off. 1 - On.
VIDEO_OUTPUT	read / write	Video output type. Value depends on implementation.
LOG_MODE	read / write	Logging mode. Values: 0 - Disable, 1 - Only file, 2 - Only terminal (console), 3 - File and terminal.
EXPOSURE_MODE	read / write	Exposure mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual, 1 - Auto (default), 2 - Shutter priority, 3 - Aperture priority.
EXPOSURE_TIME	read / write	Exposure time of the camera sensor. The exposure time is limited by the frame interval. Camera controller should interpret the values as 100 µs units, where the value 1 stands for 1/10000th of a second, 10000 for 1 second and 100000 for 10 seconds.
WHITE_BALANCE_MODE	read / write	White balance mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual, 1 - Auto.
WHITE_BALANCE_AREA	read / write	White balance area. Value depends on implementation.
WIDE_DYNAMIC_RANGE_MODE	read / write	White dynamic range mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - On.

Parameter	Access	Description
STABILIZATION_MODE	read / write	Image stabilization mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - On.
ISO_SENSITIVITY	read / write	ISO sensitivity. Value depends on implementation.
SCENE_MODE	read / write	Scene mode. Value depends on implementation.
FPS	read / write	FPS.
BRIGHTNESS_MODE	read / write	Brightness mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual, 1 - Auto.
BRIGHTNESS	read / write	Brightness. Value 0 - 100%.
CONTRAST	read / write	Contrast. Value 1 - 100%.
GAIN_MODE	read / write	Gain mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual, 1 - Auto.
GAIN	read / write	Gain. Value 0 - 100%.
SHARPENING_MODE	read / write	Sharpening mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual, 1 - Auto.
SHARPENING	read / write	Sharpening. Value 1 - 100%.
PALETTE	read / write	Palette. Value depends on implementation but it is recommended to keep default values for thermal cameras: 0 - White hot, 1 - Black hot.
AGC_MODE	read / write	Analog gain control mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual, 1 - Auto.
SHUTTER_MODE	read / write	Shutter mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual, 1 - Auto.
SHUTTER_POSITION	read / write	Shutter position. 0 (full close) - 65535 (full open).

Parameter	Access	Description
SHUTTER_SPEED	read / write	Shutter speed. Value: 0 - 100%.
DIGITAL_ZOOM_MODE	read / write	Digital zoom mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - On.
DIGITAL_ZOOM	read only	Digital zoom. Value 1.0 (x1) - 20.0 (x20).
EXPOSURE_COMPENSATION_MODE	read only	Exposure compensation mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - On.
EXPOSURE_COMPENSATION_POSITION	read / write	Exposure compensation position. Value depends on particular camera controller.
DEFOG_MODE	read / write	Defog mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - On.
DEHAZE_MODE		Dehaze mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - On.
NOISE_REDUCTION_MODE	read / write	Noise reduction mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - 2D, 3 - 3D.
BLACK_WHITE_FILTER_MODE	read only	Black and white filter mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - On.
FILTER_MODE	read / write	Filter mode. Value depends on implementation.
NUC_MODE	read / write	NUC mode for thermal cameras. Value depends on implementation but it is recommended to keep default values: 0 - Manual, 1 - Auto.
AUTO_NUC_INTERVAL	read / write	Auto NUC interval for thermal cameras. Value in milliseconds from 0 (Off) to 100000.
IMAGE_FLIP	read / write	Image flip mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - Horizontal, 2 - Vertical, 3 - Horizontal and vertical.
DDE_MODE	read / write	DDE mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - On.

Parameter	Access	Description
DDE_LEVEL	read / write	DDE level. Value depends on implementation.
ROI_X0	read / write	ROI top-left horizontal position, pixels.
ROI_Y0	read / write	ROI top-left vertical position, pixels.
ROI_X1	read / write	ROI bottom-right horizontal position, pixels.
ROI_Y1	read / write	ROI bottom-right vertical position, pixels.
TEMPERATURE	read only	Camera temperature, degree.
ALC_GATE	read / write	ALC gate. Value depends on implementation.
SENSETIVITY	read / write	Sensor sensitivity. Value depends on implementation.
CHANGING_MODE	read / write	Changing mode (day / night). Value depends on implementation.
CHANGING_LEVEL	read / write	Changing level (day / night). Value depends on implementation.
CHROMA_LEVEL	read / write	Chroma level. Values: 0 - 100%.
DETAIL	read / write	Details, enhancement. Values: 0 - 100%.
PROFILE	read / write	Camera settings profile. Value depends on implementation.
IS_CONNECTED	read only	Connection status. Value: 0 - no camera responses, 1 - connected.
IS_OPEN	read only	Open status (read only): 1 - camera control port open, 0 - not open.
TYPE	read / write	Camera type. Value depends on implementation.
CUSTOM_1	read / write	Camera custom param. Value depends on implementation.
CUSTOM_2	read / write	Camera custom param. Value depends on implementation.

Parameter	Access	Description
CUSTOM_3	read / write	Camera custom param. Value depends on implementation.

CameraParams class description

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

Class declaration

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

```
class CameraParams
{
public:
    /// Initialization string. Formats depends on implementation.
    std::string initString{"/dev/ttyUSB0;9600;20"};
    /// 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. Value depends on implementation but it is recommended
    /// to keep default values: 0 - Off. 1 - On.
    int displayMode{0};
    /// Video output type. Value depends on implementation.
    int videoOutput{0};
    /// Logging mode. Values: 0 - Disable, 1 - Only file,
    /// 2 - Only terminal (console), 3 - File and terminal.
    int logMode{0};
    /// Exposure mode. Value depends on implementation but it is recommended to
    /// keep default values: 0 - Manual, 1 - Auto (default),
    /// 2 - Shutter priority, 3 - Aperture priority.
    int exposureMode{1};
    /// Exposure time of the camera sensor. The exposure time is limited by the
    /// frame interval. Camera controller should interpret the values as 100 µs
    /// units, where the value 1 stands for 1/10000th of a second, 10000 for
    /// 1 second and 100000 for 10 seconds.
    int exposureTime{0};
    /// White balance mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual, 1 - Auto.
    int whiteBalanceMode{1};
    /// White balance area. Value depends on implementation.
    int whiteBalanceArea{0};
    /// White dynamic range mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Off, 1 - On.
    int wideDynamicRangeMode{0};
    /// Image stabilization mode. Value depends on implementation but it is
```

```
    /// recommended to keep default values: 0 - Off, 1 - On.
    int stabilisationMode{0};
    /// ISO sensitivity. Value depends on implementation.
    int isoSensetivity{0};
    /// Scene mode. Value depends on implementation.
    int sceneMode{0};
    /// FPS.
    float fps{0.0f};
    /// Brightness mode. Value depends on implementation but it is recommended
    /// to keep default values: 0 - Manual, 1 - Auto.
    int brightnessMode{1};
    /// Brightness. Value 0 - 100%.
    int brightness{0};
    /// Contrast. Value 1 - 100%.
    int contrast{0};
    /// Gain mode. Value depends on implementation but it is recommended to keep
    /// default values: 0 - Manual, 1 - Auto.
    int gainMode{1};
    /// Gain. Value 1 - 100%.
    int gain{0};
    /// Sharpening mode. Value depends on implementation but it is recommended
    /// to keep default values: 0 - Manual, 1 - Auto.
    int sharpeningMode{0};
    /// Sharpening. Value 1 - 100%.
    int sharpening{0};
    /// Palette. Value depends on implementation but it is recommended to keep
    /// default values for thermal cameras: 0 - White hot, 1 - Black hot.
    int palette{0};
    /// Analog gain control mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual, 1 - Auto.
    int agcMode{1};
    /// Shutter mode. Value depends on implementation but it is recommended to
    /// keep default values: 0 - Manual, 1 - Auto.
    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. Value depends on implementation but it is recommended
    /// to keep default values: 0 - Off, 1 - On.
    int digitalZoomMode{0};
    /// Digital zoom. Value 1.0 (x1) - 20.0 (x20).
    float digitalZoom{1.0f};
    /// Exposure compensation mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Off, 1 - On.
    int exposureCompensationMode{0};
    /// Exposure compensation position. Value depends on particular camera
    /// controller.
    int exposureCompensationPosition{0};
    /// Defog mode. Value depends on implementation but it is recommended to
    /// keep default values: 0 - Off, 1 - On.
    int defogMode{0};
    /// Dehaze mode. Value depends on implementation but it is recommended to
    /// keep default values: 0 - Off, 1 - On.
    int dehazeMode{0};
```

```
    /// Noise reduction mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Off, 1 - 2D, 3 - 3D.
    int noiseReductionMode{0};
    /// Black and white filter mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Off, 1 - On.
    int blackAndWhiteFilterMode{0};
    /// Filter mode. Value depends on implementation.
    int filterMode{0};
    /// NUC mode for thermal cameras. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual, 1 - Auto.
    int nucMode{0};
    /// Auto NUC interval for thermal cameras. Value in milliseconds
    /// from 0 (Off) to 100000.
    int autoNucIntervalMsec{0};
    /// Image flip mode. Value depends on implementation but it is recommended
    /// to keep default values: 0 - Off, 1 - Horizontal, 2 - Vertical,
    /// 3 - Horizontal and vertical.
    int imageFlip{0};
    /// DDE mode. Value depends on implementation but it is recommended to keep
    /// default values: 0 - Off, 1 - On.
    int ddeMode{0};
    /// DDE level. Value depends on implementation.
    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. Value depends on implementation.
    int alcGate{0};
    /// Sensor sensitivity. Value depends on implementation.
    float sensitivity{0};
    /// Changing mode (day / night). Value depends on implementation.
    int changingMode{0};
    /// Changing level (day / night). Value depends on implementation.
    float changingLevel{0.0f};
    /// Chroma level. Values: 0 - 100%.
    int chromaLevel{0};
    /// Details, enhancement. Values: 0 - 100%.
    int detail{0};
    /// Camera settings profile. Value depends on implementation.
    int profile{0};
    /// Connection status (read only). Shows if we have respons from camera.
    /// Value: false - not connected, true - connected.
    bool isConnected{false};
    /// Open status (read only):
    /// true - camera control port open, false - not open.
    bool isOpen{false};
    /// Camera type. Value depends on implementation.
    int type{0};
```

```
    /// Camera custom param. Value depends on implementation.
    float custom1{0.0f};
    /// Camera custom param. Value depends on implementation.
    float custom2{0.0f};
    /// Camera custom param. Value depends on implementation.
    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. Particular camera controller can have unique init string format. But it is recommended to use ';' symbol to divide part of initialization string. Recommended camera controller initialization string for controllers which uses serial port: **"/dev/ttyUSB0;9600;100"** ("/dev/ttyUSB0" - serial port name, "9600" - baudrate, "100" - serial port read timeout).
width	int	Video frame width. Value from 0 to 16384.
height	int	Video frame height Value from 0 to 16384.
displayMode	int	Display menu mode. Value depends on implementation but it is recommended to keep default values: 0 - Off. 1 - On.
videoOutput	int	Video output type. Value depends on implementation.

Field	type	Description
logMode	int	Logging mode. Values: 0 - Disable, 1 - Only file, 2 - Only terminal (console), 3 - File and terminal.
exposureMode	int	Exposure mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual, 1 - Auto (default), 2 - Shutter priority, 3 - Aperture priority.
exposureTime	int	Exposure time of the camera sensor. The exposure time is limited by the frame interval. Camera controller should interpret the values as 100 µs units, where the value 1 stands for 1/10000th of a second, 10000 for 1 second and 100000 for 10 seconds.
whiteBalanceMode	int	White balance mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual, 1 - Auto.
whiteBalanceArea	int	White balance area. Value depends on implementation.
wideDynamicRangeMode	int	White dynamic range mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - On.
stabilisationMode	int	Image stabilization mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - On.
isoSensetivity	int	ISO sensitivity. Value depends on implementation.
sceneMode	int	Scene mode. Value depends on implementation.
fps	float	FPS.
brightnessMode	int	Brightness mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual, 1 - Auto.
brightness	int	Brightness. Value 0 - 100%.
contrast	int	Contrast. Value 1 - 100%.
gainMode	int	Gain mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual, 1 - Auto.
gain	int	Gain. Value 0 - 100%.
sharpeningMode	int	Sharpening mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual, 1 - Auto.
sharpening	int	Sharpening. Value 1 - 100%.

Field	type	Description
palette	int	Palette. Value depends on implementation but it is recommended to keep default values for thermal cameras: 0 - White hot, 1 - Black hot.
agcMode	int	Analog gain control mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual, 1 - Auto.
shutterMode	int	Shutter mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual, 1 - Auto.
shutterPos	int	Shutter position. 0 (full close) - 65535 (full open).
shutterSpeed	int	Shutter speed. Value: 0 - 100%.
digitalZoomMode	int	Digital zoom mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - On.
digitalZoom	float	Digital zoom. Value 1.0 (x1) - 20.0 (x20).
exposureCompensationMode	int	Exposure compensation mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - On.
exposureCompensationPosition	int	Exposure compensation position. Value depends on particular camera controller.
defogMode	int	Defog mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - On.
dehazeMode	int	Dehaze mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - On.
noiseReductionMode	int	Noise reduction mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - 2D, 3 - 3D.
blackAndWhiteFilterMode	int	Black and white filter mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - On.
filterMode	int	Filter mode. Value depends on implementation.
nucMode	int	NUC mode for thermal cameras. Value depends on implementation but it is recommended to keep default values: 0 - Manual, 1 - Auto.
autoNucIntervalMsec	int	Auto NUC interval for thermal cameras. Value in milliseconds from 0 (Off) to 100000.

Field	type	Description
imageFlip	int	Image flip mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - Horizontal, 2 - Vertical, 3 - Horizontal and vertical.
ddeMode	int	DDE mode. Value depends on implementation but it is recommended to keep default values: 0 - Off, 1 - On.
ddeLevel	float	DDE level. Value depends on implementation.
roiX0	int	ROI top-left horizontal position, pixels.
roiY0	int	ROI top-left vertical position, pixels.
roiX1	int	ROI bottom-right horizontal position, pixels.
roiY1	int	ROI bottom-right vertical position, pixels.
temperature	int	Camera temperature, degree.
alcGate	int	ALC gate. Value depends on implementation.
sensitivity	int	Sensor sensitivity. Value depends on implementation.
changingMode	int	Changing mode (day / night). Value depends on implementation.
changingLevel	int	Changing level (day / night). Value depends on implementation.
chromaLevel	float	Chroma level. Values: 0 - 100%.
detail	int	Details, enhancement. Values: 0 - 100%.
profile	int	Camera settings profile. Value depends on implementation.
isConnected	bool	Connection status. Value: false - no camera responses, true - connected.
isOpen	bool	Open status (read only): true - camera control port open, false - not open.
type	int	Camera type. Value depends on implementation.
custom1	float	Camera custom param. Value depends on implementation.
custom2	float	Camera custom param. Value depends on implementation.
custom3	float	Camera custom param. Value depends on implementation.

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

Serialize camera params

CameraParams class provides method **encode(...)** to serialize camera params (fields of CameraParams class, see Table 4). 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 params from JSON file and write 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 **Camera** library:

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

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

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

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

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

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

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

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(${PARENT}_SUBMODULE_CAMERA                          ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_CAMERA)
    SET(${PARENT}_CAMERA                                ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_CAMERA_TEST                           OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_CAMERA_EXAMPLE                        OFF CACHE BOOL "" FORCE)
endif()

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

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

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

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 **Camera** library in your **src/CMakeLists.txt** file:

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

Done!

How to make custom implementation

The **Camera** class provides only an interface, data structures, and methods for encoding and decoding commands and params. To create your own implementation of the camera controller, you must include the Camera repository in your project (see **Build and connect to your project** section). The catalogue **example** (see **Library files** section) includes an example of the design of the custom camera controller. You must implement all the methods of the Camera interface class. Custom camera class declaration:

```
class CustomCamera: public Camera
{
public:

    /// Class constructor.
    CustomCamera();

    /// Class destructor.
    ~CustomCamera();

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

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

    /// Init camera controller by structure.
    bool initCamera(CameraParams& params);
```

```
    /// Close camera connection.
    void closeCamera();

    /// Get camera open status.
    bool isCameraOpen();

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

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

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

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

    /// Execute camera controller command.
    bool executeCommand(CameraCommand id);

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

private:

    /// Parameters structure (default params).
    CameraParams m_params;
};
```