# FlirTau2Parser C++ library

**v1.0.1**

# Table of contents



# Overview

The **FlirTau2Parser** C++ library designed for encoding control commands and decoding responses from Flir Tau 2 camera. The library includes basic methods for preparing commands (encoding) and interpreting the camera responses (decoding). It uses C++17 standard. The library provides simple interface and doesn't have third party dependencies. Also, the library provides test application to check communication with cameras via serial ports. The library doesn't have third-party dependencies but test application depends on SerialPort library (provides methods to work with serial ports, source code included, Apache 2.0 license).

# Versions

**Table 1** - Library versions.

| Version | Release date | What's new |
|---|---|---|
| 1.0.0 | 04.05.2023 | - First version. |
| 1.0.1 | 08.03.2024 | - Documentation added.<br>- Code optimized. |

# Library files

The library is supplied only by source code. The user is 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.
src  ------------------------------ Folder with library source code.
    CMakeLists.txt ---------------- CMake file.
    FlirTau2Parser.h -------------- Main library header file.
    FlirTau2ParserVersion.h ------- Header file with library version.
    FlirTau2ParserVersion.h.in ---- File for CMake to generate version header.
    FlirTau2Parser.cpp ------------ C++ implementation file.
test ------------------------------ Folder for test application files.
    3rdparty ---------------------- Folder with third-party libraries.
        CMakeLists.txt ------------ CMake file to include third-party libraries.
        SerialPort ---------------- Folder with files of SerialPort library.
    CMakeLists.txt ---------------- CMake file for test application.
    main.cpp ---------------------- Source C++ file of test application.
```

# FlirTau2Parser class description

## Class declaration

```
class FlirTau2Parser
{
public:

    /// Encode flir tau 2 command.
    bool getCommand(uint8_t* data,
                    FlirTau2Command id,
                    int8_t &size,
                    int arg1 = 0,
                    int arg2 = 0,
                    int arg3 = 0,
```

```
                    int arg4 = 0,
                    int arg5 = 0,
                    int arg6 = 0,
                    int arg7 = 0,
                    int arg8 = 0,
                    int arg9 = 0,
                    int arg10 = 0,
                    int arg11 = 0,
                    int arg12 = 0);

    /// Decode flir tau 2 response.
    FlirTau2Response decodeResponse(uint8_t nextByte,
                                    std::vector<int> &args,
                                    uint8_t len);
    /// Get library version.
    static std::string getVersion();
};
```

## getCommand method

The **getCommand(...)** method encodes (prepares data) control command for camera. The camera does not send data on its own, but only responds to requests. Method declaration:

```
bool getCommand(uint8_t* data,
                FlirTau2Command id,
                int8_t &size,
                int arg1 = 0,
                int arg2 = 0,
                int arg3 = 0,
                int arg4 = 0,
                int arg5 = 0,
                int arg6 = 0,
                int arg7 = 0,
                int arg8 = 0,
                int arg9 = 0,
                int arg10 = 0,
                int arg11 = 0,
                int arg12 = 0);
```

| Parameter | Value |
| --- | --- |
| data | Address of buffer for encoded data (command). Size of buffer should be >= 8. |
| id | Command ID according to FlirTau2Command enum. |
| size | Size of encoded command. |
| arg1-arg12 | Arguments of command. The value of arguments depends on command ID (see FlirTau2Command enum). Some commands don't have arguments. |

**Returns:** TRUE if commands encoded or FALSE if not (not valid ID, not valid parameters or not valid address).

## decodeResponse method

The **decodeResponse(...)** method decodes response from camera. See the Decoding example. Once the data from the serial port has been read the user must pass it to the method byte-by-byte (each byte separately in sequence). The library has an internal ring buffer. When the user passes another byte to the method, the library shifts the ring buffer by one byte and copies the new byte to the end of the buffer. If the library detects a valid response in the ring buffer, it returns the response ID (see FlirTau2Response) and returns the response arguments. In all other cases, the library returns a **NONE** identifier. Method declaration:

```
FlirTau2Response decodeResponse(uint8_t nextByte, std::vector<int> &args, uint8_t len);
```

| Parameter | Value |
|---|---|
| nextByte | Next read byte from serial port. |
| args | Response arguments from camera. The number of arguments and their values depend on response ID (see FlirTau2Response enum). Some responses don't have arguments. |
| len | Length of camera response buffer. |

**Returns:** Response FlirTau2Response if response detected or **NONE** ID if not.

## getVersion method

The **getVersion()** method return string of current **FlirTau2Parser** class version. Method declaration:

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

Method can be used without **FlirTau2Parser** class instance:

```
cout << "FlirTau2Parser version: " << FlirTau2Parser::getVersion() << endl;
```

Console output:

```
FlirTau2Parser version: 1.0.1
```

# Data structures

# FlirTau2Command

The command IDs are described in the **FlirTau2Command** structure in the **FlirTau2Parser.h** file. The definition of the structure is as follows (part of definition):

```
enum class FlirTau2Command
{
    ////////////////////////// Main commands ///////////////////////////////////

    /// No operation command, no arguments.
    NO_OP,

    /// Command to set all settings as power-on defaults, no arguments.
    SET_DEFAULTS,

    /// Command to reset to default modes, no arguments.
    RESET,

    /// Reset camera with factory reset defaults, no arguments.
    RESET_FACTORY_DEFAULTS,

    /// Get serial number of camera and sensor, no arguments.
    GET_SERIAL_NUMBER,

    /// Get serial number of firmware and software, no arguments.
    GET_REVISION,

    /// Command to set gain mode
    /// arg1 : 0x00 - Automatic
    ///        0x01 - Low gain only
    ///        0x02 - High gain only
    ///        0x03 - Manual gain
    SET_GAIN_MODE,

    /// Command to get gain mode, no arguments.
    GET_GAIN_MODE,

    /// Command to set FFC mode
    /// arg1 : 0x00 - manual
    ///        0x01 - Auto
    ///        0x02 - External
    SET_FFC_MODE,

    /// Command to get FFC mode, no arguments.
    GET_FFC_MODE,

    /// Command to set DO FFC
    /// arg1 : 0x00 - short FFC
    ///        0x01 - long FFC
    SET_DO_FFC,

    /// Command to get DO FFC mode, no arguments.
    GET_DO_FFC,

    /// Command to set FFC period.
```

```
    /// for current gain state
    ///     arg1 = FFC interval
    /// for high and low gain states
    ///     arg1 = FFC interval, high gain
    ///     arg2 = FFC interval, low gain
    SET_FFC_PERIOD,

    /// Command to get FFC period
    /// arg1 : 0x00 - current state
    ///        0x01 - high and low gain state
    GET_FFC_PERIOD,

    /// Command to set FFC temp difference to trigger auto FFC.
    /// for current gain state
    ///     arg1 = Temp delta
    /// for high and low gain states
    ///     arg1 = FFC temp delta, high gain
    ///     arg2 = FFC temp delta, low gain
    SET_FFC_TEMP,

    /// Command to get FFC period
    /// arg1 : 0x00 - current state
    ///        0x01 - high and low gain state
    GET_FFC_TEMP,

    /// Command to set video mode
    /// arg1 : 0x00 - real time
    ///        0x01 - freeze frame
    ///        0x04 - 2x zoom
    ///        0x08 - 4x zoom (Tau 320 only)
    SET_VIDEO_MODE,

    /// Command to get video mode, no arguments.
    GET_VIDEO_MODE,

    /// Command to set analog video LUT or intensity transform.
    /// arg1 : 0x00 - White hot
    ///        0x01 - Black hot
    ///        0x02 - Fusion
    ///        0x03 - Rainbow
    ///        0x04 - Globow
    ///        0x05 - Ironbow1
    ///        0x06 - Ironbow2
    ///        0x07 - Sepia
    ///        0x08 - Color1
    ///        0x09 - Color2
    ///        0x0A - ice and fire
    ///        0x0B - Rain
    ///        0x0C - CEM custom
    SET_VIDEO_LUT,

    /// Get video LUT, no arguments.
    GET_VIDEO_LUT,

    /// Command to set analog video orientation.
    /// arg1 : 0x00 - normal
```

```
    ///        0x01 - invert
    ///        0x02 - revert
    ///        0x03 - invert+revert
    SET_VIDEO_ORIENTATION,

    /// Command to get analog video orientation, no arguments.
    GET_VIDEO_ORIENTATION,

    /// Command to set digital output chanel mode
    /// XP chanel settings
    ///     arg1 : 0x00 - disabled
    ///            0x01 - BT.656 (post AGC)
    ///            0x02 - CMOS 14 bit (pre AGC)
    ///            0x03 - CMOS 8 bit (post AGC)
    ///            0x04 - CMOS 14 bit inverted
    ///            0x05 - CMOS 8 bit inverted
    /// LVDS chanel settings
    ///     arg2 : 0x00 - 14 bit data
    ///            0x01 - 8 bit data
    ///            0x02 - digital off
    ///            0x03 - 14 bit un-filtered
    ///            0x04 - 8 bit inverted
    ///            0x05 - 14 bit inverted
    ///            0x06 - 14 bit inverted un-filtered
    SET_DIGITAL_OUTPUT_MODE,

    /// Command to get digital output mode, no arguments.
    GET_DIGITAL_OUTPUT_MODE,

    /// Set agc type (image optimization)
    /// arg1 : 0x0000 = plateau histogram
    ///        0x0001 = once bright
    ///        0x0002 = auto bright
    ///        0x0003 = manual
    ///        0x0004 = not defined (returns error)
    ///        0x0005 = linear
    SET_AGC_MODE,

    /// Command to get AGC mode, no arguments.
    GET_AGC_MODE,

    /// Command to set contrast.
    /// arg1 = contrast (0x00 - 0xFF)
    SET_CONTRAST,

    /// Command to get contrast, no arguments.
    GET_CONTRAST,

    /// Command to set brightness.
    /// arg1 = brightness (0x00 - 0x3FFF)
    SET_BRIGHTNESS,

    /// Command to get brightness, no arguments.
    GET_BRIGHTNESS,

    /// Set brightness bias
```

```
    /// arg1 = bias value (0 - 0x3FF)
    SET_BRIGHTNESS_BIAS,

    /// Get brightness bias, no argument
    GET_BRIGHTNESS_BIAS,

    /// Set spot-meter mode
    /// arg1 : 0x00 - disabled
    ///        0x01 - on, fahrenheit scale
    ///        0x02 - on, centigrade scale
    SET_SPOT_METER_MODE,

    /// Get spot meter mode, no arguments.
    GET_SPOT_METER_MODE,

    /// Get the FPA temp in celsius*10 or raw counts
    /// 512 represents 51.2 celsius.
    /// arg1 : 0x00 - temp in celsius * 10
    ///        0x01 - raw temp value
    READ_SENSOR,

    /// Enable/disable external sync feature.
    /// arg1 : 0x00 - disabled
    ///        0x01 - slave
    ///        0x02 - master
    SET_EXTERNAL_SYNC,

    /// Get external sync, no arguments required.
    GET_EXTERNAL_SYNC,

    /// Set isotherm mode on/off.
    /// arg1 : 0x00 - disabled.
    ///        0x01 - enabled.
    SET_ISOTHERM_ON_OFF,

    /// Get isotherm on/off, no arguments.
    GET_ISOTHERM_ON_OFF,

    /// Set isotherm threshold in percentage.
    /// arg1 = lower threshold
    /// arg2 = upper threshold
    SET_ISOTHERM_THRESHOLD_PER,

    /// Get isotherm threshold in percentage. No arguments.
    GET_ISOTHERM_THRESHOLD_PER,

    /// Set isotherm threshold in degree celsius.
    /// arg1 = lower threshold
    /// arg2 = upper threshold
    SET_ISOTHERM_THRESHOLD,

    /// Get isotherm threshold in degree celsius. No arguments.
    GET_ISOTHERM_THRESHOLD,

    /// Set test pattern mode.
    /// arg1 : 0x00 - test pattern off
```

```
    ///        0x01 - ascending ramp
    ///        0x03 - big vertical
    ///        0x04 - horizontal shade
    ///        0x06 - color bars
    ///        0x08 - ramp with steps
    SET_TEST_PATTERN,

    /// Get test pattern, no arguments.
    GET_TEST_PATTERN,

    /// Get spot meter value in degrees celsius. No arguments required.
    GET_SPOT_METER_VALUE,

    /// Set spot meter display mode.
    /// arg1 : 0x00 - display off
    ///        0x01 - numeric only
    ///        0x02 - thermometer only
    ///        0x03 - numeric & thermometer
    SET_SPOT_METER_DISPLAY,

    /// Get spot meter display mode, no arguments.
    GET_SPOT_METER_DISPLAY,

    /// Set time to display the FFC.
    /// arg1 = time in frames (data range 0-600 frames)]
    SET_FFC_WARN_TIME,

    /// Get FFC warn time
    GET_FFC_WARN_TIME,

    /// Set agc ITT filter value.
    /// arg1 : 0x00 - immediate
    ///        (0x01-0xFF) -  numerator
    SET_AGC_FILTER_VALUE,

    /// Get AGC ITT filter value
    GET_AGC_FILTER_VALUE,

    /// Set the plateau level for plateau AGC.
    /// arg1 = level in range 0-1000
    SET_PLATEAU_LEVEL,

    /// Get the plateau level for plateau AGC. No arguments.
    GET_PLATEAU_LEVEL,

    /// Set ROI for AGC
    /// arg1 = left normal roi
    /// arg2 = top normal roi
    /// arg3 = right normal roi
    /// arg4 = bottom normal roi
    /// arg5 = left 2x roi
    /// arg6 = top 2x roi
    /// arg7 = right 2x roi
    /// arg8 = bottom 2x roi
    /// arg9 = left 4x roi
    /// arg10 = top 4x roi
```

```
    /// arg11 = right 4x roi
    /// arg12 = bottom 4x roi
    SET_AGC_ROI,

    /// Get ROI for AGC, no arguments.
    GET_AGC_ROI,

    /// Set ITT midpoint  offset.
    /// arg1 = midpoint offset in range 0-255
    SET_ITT_MIDPOINT,

    /// Get ITT midpoint offset, no arguments required.
    GET_ITT_MIDPOINT,

    /// Get camera part number, no arguments.
    GET_CAMERA_PART_NUM,

    /// Set max value of video gain
    /// arg1  = gain in range 0 - 2048
    SET_AGC_MAX_GAIN,

    /// Get AGC max gain, no arguments.
    GET_AGC_MAX_GAIN,

    /// Set pan and tilt position
    /// arg1 = pan position in range -82 and 82
    /// arg2 = tilt position in range -68 and 68
    SET_PAN_TILT_POS,

    /// Get pan position, no arguments.
    GET_PAN_TILT_POS,

    /// Set video standard.
    /// arg1 : 0x00 - NTSC
    ///        0X01 - PAL
    SET_VIDEO_STANDARD,

    /// Get video standard, no arguments.
    GET_VIDEO_STANDARD,

    /// Set shutter position.
    /// arg1 = 0x00 - open
    ///        0x01 - close
    SET_SHUTTER_ON_OFF,

    /// Get shutter on/off, no arguments.
    GET_SHUTTER_ON_OFF,

    /// Set the gain of the DDE filter.
    /// arg1 = gain value in range 0x00 - 0xFF
    SET_DDE_GAIN,

    /// Get the gain of DDE filter, no arguments.
    GET_DDE_GAIN,

    /// Set the threshold of the DDE filter.
```

```
    /// arg1 = threshold value in range 0x00 - 0xFF
    SET_DDE_THRESHOLD,

    /// Get the threshold of DDE filter, no arguments.
    GET_DDE_THRESHOLD,

    /// Set Spatial threshold and mode.
    /// arg1 = threshold in range 0 - 15
    /// arg2 = : 0x00 - manual
    ///          0x01 - Auto
    SET_SPATIAL_THRESHOLD_AND_MODE,

    /// Set Spatial threshold and mode, no arguments
    GET_SPATIAL_THRESHOLD_AND_MODE,

    /// Set low to high threshold and population.
    /// arg1 =  high to low threshold
    /// arg2 =  high to low population
    /// arg3 =  low to high threshold
    /// arg4 =  low to high population
    SET_SWITCH_PARAMS,

    /// Get gain switch params, no arguments.
    GET_SWITCH_PARAMS
};
```

## FlirTau2Response

The response IDs are described in the **FlirTau2Response** structure in the **FlirTau2Parser.h** file. The definition of the structure is as follows (part of definition):

```
enum class FlirTau2Response
{
    // No response detected
    NONE,

    //Acknowledgement detected for sent command
    ACK,

    // No operation command response
    NO_OP,

    /// Serial number of camera and sensor
    /// data1 = high word of serial number of camera
    /// data2 = low word of serial number of camera
    /// data3 = high word of serial number of sensor
    /// data4 = low word of serial number of sensor
    SERIAL_NUMBER,

    /// Revision number, firmware and software version
    /// data1 = software major version
    /// data2 = software minor version
    /// data3 = firmware major version
```

```
    /// data4 = firmware minor version
    REVISION_NUMBER,

    /// Gain mode, dynamic range control mode.
    /// data1 : 0x00 - automatic
    ///         0x01 - low gain only
    ///         0x02 - high gain only
    ///         0x03 - manual (no switching)
    GAIN_MODE,

    /// FFC mode
    /// data1 : 0x00 - manual
    ///        0x01 - Auto
    ///        0x02 - External
    FFC_MODE,

    /// DO FFC
    /// data1 : 0x00 - short FFC
    ///        0x01 - long FFC
    DO_FFC,

    /// FFC period.
    /// for current gain state
    ///     data1 = FFC interval
    /// for high and low gain states
    ///     data1 = FFC interval, high gain
    ///     data2 = FFC interval, low gain
    FFC_PERIOD,

    /// FFC temp difference to trigger auto FFC.
    /// for current gain state
    ///     data1 = Temp delta
    /// for high and low gain states
    ///     data1 = FFC temp delta, high gain
    ///     data2 = FFC temp delta, low gain
    FFC_TEMP,

    /// Video mode
    /// data1 : 0x00 - real time
    ///        0x01 - freeze frame
    ///        0x04 - 2x zoom
    ///        0x08 - 4x zoom (Tau 320 only)
    VIDEO_MODE,

    /// analog video LUT or intensity transform.
    /// data1 : 0x00 - White hot
    ///        0x01 - Black hot
    ///        0x02 - Fusion
    ///        0x03 - Rainbow
    ///        0x04 - Globow
    ///        0x05 - Ironbow1
    ///        0x06 - Ironbow2
    ///        0x07 - Sepia
    ///        0x08 - Color1
    ///        0x09 - Color2
    ///        0x0A - ice and fire
```

```
    ///        0x0B - Rain
    ///        0x0C - CEM custom
    VIDEO_LUT,

    /// analog video orientation.
    /// data1 : 0x00 - normal
    ///        0x01 - invert
    ///        0x02 - revert
    ///        0x03 - invert+revert
    VIDEO_ORIENTATION,

    /// digital output chanel mode
    /// XP chanel settings
    ///     data1 : 0x00 - disabled
    ///             0x01 - BT.656 (post AGC)
    ///             0x02 - CMOS 14 bit (pre AGC)
    ///             0x03 - CMOS 8 bit (post AGC)
    ///             0x04 - CMOS 14 bit inverted
    ///             0x05 - CMOS 8 bit inverted
    /// LVDS chanel settings
    ///     data2 : 0x00 - 14 bit data
    ///             0x01 - 8 bit data
    ///             0x02 - digital off
    ///             0x03 - 14 bit un-filtered
    ///             0x04 - 8 bit inverted
    ///             0x05 - 14 bit inverted
    ///             0x06 - 14 bit inverted un-filtered
    DIGITAL_OUTPUT_MODE,

    /// agc type (image optimization)
    /// data1 : 0x0000 = plateau histogram
    ///        0x0001 = once bright
    ///        0x0002 = auto bright
    ///        0x0003 = manual
    ///        0x0004 = not defined (returns error)
    ///        0x0005 = linear
    AGC_MODE,

    /// contrast.
    /// data1 = contrast (0x00 - 0xFF)
    CONTRAST,

    /// brightness.
    /// data1 = brightness (0x00 - 0x3FFF)
    BRIGHTNESS,

    /// brightness bias
    /// data1 = bias value (0 - 0x3FF)
    BRIGHTNESS_BIAS,

    /// spot-meter mode
    /// data1 : 0x00 - disabled
    ///        0x01 - on, fahrenheit scale
    ///        0x02 - on, centigrade scale
    SPOT_METER_MODE,
```

```
    /// FPA temp in celsius*10 or raw counts
    /// data1 : temp value
    FPA_TEMP,

    /// external sync feature.
    /// data1 : 0x00 - disabled
    ///         0x01 - slave
    ///         0x02 - master
    EXTERNAL_SYNC,

    /// isotherm mode on/off.
    /// data1 : 0x00 - disabled.
    ///         0x01 - enabled.
    ISOTHERM_ON_OFF,

    /// isotherm threshold in percentage.
    /// data1 = lower threshold
    /// data2 = upper threshold
    ISOTHERM_THRESHOLD_PER,

    /// isotherm threshold.
    /// data1 = lower threshold
    /// data2 = upper threshold
    ISOTHERM_THRESHOLD,

    /// test pattern mode.
    /// data1 : 0x00 - test pattern off
    ///         0x01 - ascending ramp
    ///         0x03 - big vertical
    ///         0x04 - horizontal shade
    ///         0x06 - color bars
    ///         0x08 - ramp with steps
    TEST_PATTERN,

    /// spot meter value in degrees celsius.
    SPOT_METER_VALUE,

    /// spot meter display mode.
    /// data1 : 0x00 - display off
    ///         0x01 - numeric only
    ///         0x02 - thermometer only
    ///         0x03 - numeric & thermometer
    SPOT_METER_DISPLAY,

    /// time to display the FFC.
    /// data1 = time in frames (data range 0-600 frames)]
    FFC_WARN_TIME,

    /// agc ITT filter value.
    /// data1 : 0x00 - immediate
    ///         (0x01-0xFF) -  numerator
    AGC_FILTER_VALUE,

    /// plateau level for plateau AGC.
    /// data1 = level in range 0-1000
    PLATEAU_LEVEL,
```

```
    /// ROI for AGC,
    /// data1 = left
    /// data2 = top
    /// data3 = right
    /// data4 = bottom
    AGC_ROI,

    /// ITT midpoint  offset.
    /// data1 = midpoint offset in range 0-255
    ITT_MIDPOINT,

    /// max value of video gain
    /// data1  = gain in range 0 - 2048
    AGC_MAX_GAIN,

    /// pan and tilt position
    /// data1 = pan position in range -82 and 82
    /// data2 = tilt position in range -68 and 68
    PAN_TILT_POS,

    /// video standard.
    /// data1 : 0x00 - NTSC
    ///         0X01 - PAL
    VIDEO_STANDARD,

    /// shutter position.
    /// data1 = 0x00 - open
    ///         0x01 - close
    SHUTTER_ON_OFF,

    /// gain of the DDE filter.
    /// data1 = gain value in range 0x00 - 0xFF
    DDE_GAIN,

    /// threshold of the DDE filter.
    /// data1 = threshold value in range 0x00 - 0xFF
    DDE_THRESHOLD,

    /// Spatial threshold and mode.
    /// data1 = threshold in range 0 - 15
    /// data2 = : 0x00 - manual
    ///           0x01 - Auto
    SPATIAL_THRESHOLD_AND_MODE,

    /// Switch parameters.
    /// data1 =  high to low threshold
    /// data2 =  high to low population
    /// data3 =  low to high threshold
    /// data4 =  low to high population
    SWITCH_PARAMS
};
```

# Test application

Folder **FlirTau2Parser/test** contains the test application files. The test application allows you to generate any command, send it to the camera over the serial port, receive and decode the response. Once started, the user must enter the serial port name (full name for Linux or just the port number for Windows), baud rate (default 9600):

```
================================================
FlirTau2Parser tester v1.0.1
================================================

Set serial port name: /dev/serial/by-id/usb-FTDI_USB-RS232_Cable_Tau_2-if00-port0
Set baudrate (default : 9600): 9600
```

After user can choose command (enter command ID). If no necessary command has been displayed use can set ID according to FlirTau2Command enum (test application supports all commands):

```
Commands:
6 - SET GAIN MODE
7 - GET GAIN MODE
8 - SET FFC MODE
9 - GET FFC MODE
26 - SET CONTRAST
27 - GET CONTRAST
28 - SET BRIGHTNESS
29 - GET BRIGHTNESS

For rest of commands check FlirTau2Parser.h file

Choose command:
```

Then, corresponding command will be generated and sent to camera. Camera will return related response such as ACK (4 bytes) or general response (7 bytes).

# Examples

## Response decoding example

Below is an example of decoding response from camera.

```
// Get data from serial port.
uint8_t buffer[128];
int bytes = serialPort.readData(buffer, 128);
std::vector<int> readArguments;

// Decoding response byte-by-byte.
```

```
for (int i = 0; i < bytes; ++i)
{
    response = parser.decodeResponse(buffer[i], readArguments, bytes);
    switch (response)
    {

    case FlirTau2Response::GAIN_MODE:
    {
        cout << "Gain mode : " << readArguments.at(0) << endl;
        break;
    }

        ///...
    }
}
```

The response IDs are described in the FlirTau2Response structure in the **FlirTau2Parser.h** file. The declaration of the structure is as follows.

## Command encoding example

Below is an example of encoding command.

```
// Init variables.
uint8_t buffer[24];
int size;

parser.getCommand(buffer, FlirTau2Command::FOCUS_NEAR, size);

// Send data.
if (serialPort.write(buffer, size) != size)
{
    cout << "ERROR: Can't send data" << endl;
    continue;
}

int arg1; // gain mode value.
parser.getCommand(buffer, FlirTau2Command::SET_GAIN_MODE, size, arg1);

// Send data.
if (serialPort.write(buffer, size) != size)
{
    cout << "ERROR: Can't send data" << endl;
    continue;
}
```

# Build and connect to your project

Typical commands to build **FlirTau2Parser** library:

```
cd FlirTau2Parser
git submodule update --init --recursive
mkdir build
cd build
cmake ..
make
```

If you want to connect **FlirTau2Parser** 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 **FlirTau2Parser** repository there. New structure of your repository:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    FlirTau2Parser
```

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(${PARENT}_SUBMODULE_FLIR_TAU2_PARSER              ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_FLIR_TAU2_PARSER)
    SET(${PARENT}_FLIR_TAU2_PARSER                    ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_FLIR_TAU2_PARSER_TEST               OFF CACHE BOOL "" FORCE)
endif()

################################################################################
## INCLUDING SUBDIRECTORIES
## Adding subdirectories according to the 3rd-party configuration
################################################################################
if (${PARENT}_SUBMODULE_FLIR_TAU2_PARSER)
    add_subdirectory(FlirTau2Parser)
endif()
```

File **3rdparty/CMakeLists.txt** adds folder **FlirTau2Parser** 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
    FlirTau2Parser
```

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} FlirTau2Parser)
```

Done!