FujiProtocolParser C++ library

v1.0.1

Table of contents

Overview

The **FujiProtocolParser** C++ library designed for encoding control commands and checking responses for Fujinon CCTV lenses. The library includes basic methods for preparing commands (encoding) and interpreting the camera responses (checking command arguments). It uses C++17 standard. The library provides simple interface and doesn't have third party dependencies. Also, the library includes demo application to test communication with lenses via serial ports.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	27.11.2023	- First version.
1.0.1	31.01.2024	- Documentation added. - 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.
src  ---------------------------------- Folder with library source code.
    CMakeLists.txt -------------------- CMake file.
    FujiProtocolParser.h -------------- Main library header file.
    FujiProtocolParserVersion.h ------- Header file with library version.
    FujiProtocolParserVersion.h.in ---- File for CMake to generate version header.
    FujiProtocolParser.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.
```

The test application depends on open source SerialPort C++ library (source code included in repository, Apache 2.0 license).

FujiProtocolParser class description

Class declaration

```
class FujiProtocolParser
{
public:
    /// Get library version.
    static std::string getVersion();

    /// Encode Fuji lens command.
    bool getCommand(uint8_t* data,
```

```
                    int& size,
                    FujiCommand id,
                    int arg1 = 0,
                    int arg2 = 0);

    /// Check Fuji lens response.
    bool checkResponse(uint8_t* data,
                       int size,
                       FujiCommand id,
                       int& arg1,
                       int& arg2);
};
```

getCommand method

getCommand(...) method encodes (prepares data) control command for lens. The lens does not send data on its own, but only responds to requests. To decode responses to some requests, the library needs to know which request was sent to the lens. For this purpose, the user must provide the following principle of data exchange with the lens: the user prepares a command for the lens (with getCommand(...) method) and sends it, then waits for a response (standard response time is 50 msec), then reads data from the serial port (the source code of the library for working with serial ports is included in the repository, SerialPort) and passes it to the library for decoding. Method declaration:

```
bool getCommand(uint8_t* data, int& size, FujiCommand id, int arg1 = 0, int arg2 = 0)
```

Parameter	Value
data	Pointer to buffer for encoded data (command). Size of buffer should be >= 18 bytes.
size	Size of encoded command's buffer.
id	Command ID from FujiCommand enum.
arg1	First argument of command. The value of argument depends on command ID (see FujiCommand enum). Some commands don't have arguments.
arg2	Second optional argument of command. The value of argument depends on command ID (see FujiCommand enum). Some commands don't have arguments.

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

checkResponse method

checkResponse(...) 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 as buffer. If the library detects a valid response in the buffer, it returns TRUE and returns the response arguments. Method declaration:

```
bool checkResponse(uint8_t* data, int size, FujiCommand id, int& arg1, int& arg2);
```

Parameter	Value
data	Pointer to buffer for data that read from lens.
size	Size of buffer.
id	Previosly sent command ID from FujiCommand enum.
arg1	First argument of detected command. The value of argument depends on command ID (see FujiCommand enum). Some commands don't have arguments.
arg2	Second argument of detected command. The value of argument depends on command ID (see FujiCommand enum). Some commands don't have arguments.

Returns: TRUE if command response detected or FALSE if not.

getVersion method

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

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

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

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

Console output:

```
FujiProtocolParser version: 1.0.1
```

Data structures

FujiCommand enum

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

```
/**
 * @brief Fuji lens commands according to control protocol specification.
 */
enum class FujiCommand
{
    /// Connection request. After first connection request the lens will
    /// make zoom/focus initialization. [Connection response]
    CONNECT = 1,
    /// Request for the first half of the lens name. [First half of the lens
    /// name response]
```

```
    GET_LENS_NAME_1,
    /// Request for the second half of the lens name. [Second half of the
    /// lens name response]
    GET_LENS_NAME_2,
    /// Request to open FN. [Response FN: 2^(8x(1 - data/0x10000))].
    REQUEST_OPEN_FN,
    /// Request for tele-end focal length. The 16 bit data divided into two
    /// parts: one is the exponent part b (4 bits), and mantissa 12 bit.
    /// "a" represents 0 to 4095 without symbols and "b" represents -8 to 7
    /// with symbol. The exponent part and mantissa shall represent the value
    /// (distance) a x 10b [m]. [Tele focal lens response].
    REQUEST_TELE_END_FOCAL_LENGTH,
    /// Request for wide-end focus length. The 16 bit data divided into two
    /// parts: one is the exponent part b (4 bits), and mantissa 12 bit.
    /// "a" represents 0 to 4095 without symbols and "b" represents -8 to 7
    /// with symbol. The exponent part and mantissa shall represent the value
    /// (distance) a x 10b [m]. [Wide focal lens response].
    REQUEST_WIDE_END_FOCAL_LENGTH,
    /// Request for mode. The 16 bit data divided into two
    /// parts: one is the exponent part b (4 bits), and mantissa 12 bit.
    /// "a" represents 0 to 4095 without symbols and "b" represents -8 to 7
    /// with symbol. The exponent part and mantissa shall represent the value
    /// (distance) a x 10b [m]. [MOD response].
    REQUEST_MOD,
    /// Set iris position. Command arguments:
    /// arg1 - 0(full close) to 65535(full open).
    /// [ACKNOWLEDGE response].
    SET_IRIS_POSITION,
    /// Set zoom position. Command arguments:
    /// arg1 - 0(full wide) to 65535(full tele).
    /// [ACKNOWLEDGE response].
    SET_ZOOM_POSITION,
    /// Set focus position. Command arguments:
    /// arg1 - 0(MOD) to 65535(infinity).
    /// [ACKNOWLEDGE response].
    SET_FOCUS_POSITION,
    /// Move zoom with speed. Command arguments:
    /// arg1:
    ///      -32767 move wide with max speed.
    ///      0 stop zoom moving.
    ///      32767 move tele with max speed.
    /// [ACKNOWLEDGE response].
    MOVE_ZOOM_WITH_SPEED,
    /// Move focus with speed. Command arguments:
    /// arg1:
    ///      -32767 move to MOD with max speed.
    ///      0 stop focus moving.
    ///      32767 move to infinity with max speed.
    /// [ACKNOWLEDGE response].
    MOVE_FOCUS_WITH_SPEED,
    /// Request for iris position. 0x0000(full close) to 0xFFFF(full open).
    /// [Response to iris position]
    REQUEST_IRIS_POSITION,
    /// Request for zoom position. Response: 0x0000(full wide) to
    /// 0xFFFF(full tele). [Response for zoom position].
    REQUEST_ZOOM_POSITION,
```

```
    /// Request for focus position. 0x0000 (MOD) to 0xFFFF (infinity).
    /// [Response for focus position].
    REQUEST_FOCUS_POSITION,
    /// Switch 0 control. Command arguments:
    /// arg1 (optical filter option):
    ///       1 - filter type 1.
    ///       2 - filter type 2.
    ///       3 - filter type 3.
    ///       4 - filter type 4.
    /// arg2 (near infra-red wavelength):
    ///       1 - no effect.
    ///       2 - n-IR 850 nm.
    ///       3 - n_IR 870 nm.
    ///       4 - n_IR 950 nm.
    /// [ACKNOWLEDGE response].
    SWITCH_0,
    /// Switch 2 control. Command arguments:
    /// arg1: 0 - IRIS auto, 1 - IRIS manual.
    /// [ACKNOWLEDGE response].
    SWITCH_2,
    /// Switch 3 control. Command arguments:
    /// arg1 (Extended magnification):
    /// 0 - x1.0 (default).
    /// 1 - x2.0.
    /// [ACKNOWLEDGE response].
    SWITCH_3,
    /// Switch 6 control. Command arguments:
    /// arg1 (Stabilization mode):
    /// 0 - stabilizer ON.
    /// 1 - stabilizer OFF.
    /// [ACKNOWLEDGE response].
    SWITCH_6,
    /// Request for switch 0 control.
    /// Response parameters:
    /// arg1 (optical filter option):
    ///       1 - filter type 1.
    ///       2 - filter type 2.
    ///       3 - filter type 3.
    ///       4 - filter type 4.
    /// arg2 (near infra-red wavelength):
    ///       1 - no effect.
    ///       2 - n-IR 850 nm.
    ///       3 - n_IR 870 nm.
    ///       4 - n_IR 950 nm.
    REQUEST_SWITCH_0,
    /// Request for switch 2 control.
    /// Response arguments: 0 - IRIS auto, 1 - IRIS manual.
    REQUEST_SWITCH_2,
    /// Request for switch 3 control. Response arguments:
    /// arg1 (Extended magnification):
    /// 1 - x1.0 (default).
    /// 2 - x2.0.
    REQUEST_SWITCH_3,
    /// Request for switch 6 control. Response arguments:
    /// arg1 (Stabilization mode):
    /// 0 - stabilizer ON.
```

```
    /// 1 - stabilizer OFF.
    REQUEST_SWITCH_6,
    /// Auto iris control. Hunting mitigation parameters in auto iris. When
    /// hunting is occurred by the camera change this parameter.
    /// Command arguments:
    /// arg1:
    ///      100 (default).
    ///      10 - 100 setting range.
    SET_AUTO_IRIS_PARAMS,
    /// Request for auto iris parameters. Hunting mitigation parameters in
    /// auto iris. When hunting is occurred by the camera change this
    /// parameter. Response arguments:
    /// arg1:
    ///      100 (default).
    ///      10 - 100 setting range.
    REQUEST_AUTO_IRIS_PARAMS,
    /// AF switch 0 control. Command arguments:
    /// arg1 (search range):
    ///      0 - full range search.
    ///      1 - 1/2 range search.
    ///      2 - 1/4 range search.
    ///      3 - 1/8 range search.
    ///      4 - 1/16 range search.
    ///      5 - 1/32 range search.
    ///      6 - 1/64 range search.
    ///      7-15 - full range search.
    /// arg2 (AF mode):
    ///      0 - AF ON.
    ///      1 - AF OFF.
    /// [ACKNOWLEDGE response].
    AF_SWITCH_0,
    /// AF video delay control. Adjustment parameters of the video signal
    /// delay. When focusing position is shifted by camera, change this
    /// parameter. Command parameters:
    /// arg1:
    ///      6 (default).
    ///      0 - 128 setting range.
    /// If a large value is set, the driving time will be longer. If the
    /// value is not appropriate, it may become our of focus.
    SET_AF_VIDEO_DELAY,
    /// Request AF switch 0 position. Response arguments:
    /// arg1:
    ///      0 - AF active.
    ///      1 - AF not active.
    REQUEST_AF_SWITCH_0_POSITION,
    /// Request AF video delay parameter. Response parameters:
    /// arg1:
    ///      0x06 (default).
    ///      0x00 - 0x80 setting range.
    /// If a large value is set, the driving time will be longer. If the
    /// value is not appropriate, it may become our of focus.
    REQUEST_AF_VIDEO_DELAY
};
```

Test application

Folder **FujiProtocolParser/test** contains the test application files. The test application allows you to generate any command, send it to the lens 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).

```
==========================================
Fuji Protocol Parser v1.0.1
==========================================

Enter serial port name :
```

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

```
Commands:
1 CONNECT
2 GET_LENS_NAME_1
3 GET_LENS_NAME_2
4 REQUEST_OPEN_FN
5 REQUEST_TELE_END_FOCAL_LENGTH
6 REQUEST_WIDE_END_FOCAL_LENGTH
7 REQUEST_MOD
8 SET_IRIS_POSITION
9 SET_ZOOM_POSITION
10 SET_FOCUS_POSITION
11 MOVE_ZOOM_WITH_SPEED
12 MOVE_FOCUS_WITH_SPEED
13 REQUEST_IRIS_POSITION
14 REQUEST_ZOOM_POSITION
15 REQUEST_FOCUS_POSITION
16 SWITCH_0
17 SWITCH_2
18 SWITCH_3
19 SWITCH_6
20 REQUEST_SWITCH_0
21 REQUEST_SWITCH_2
22 REQUEST_SWITCH_3
23 REQUEST_SWITCH_6
24 SET_AUTO_IRIS_PARAMS
25 REQUEST_AUTO_IRIS_PARAMS
26 AF_SWITCH_0
27 SET_AF_VIDEO_DELAY
28 REQUEST_AF_SWITCH_0_POSITION
29 REQUEST_AF_VIDEO_DELAY
Choose command:
```

Then, corresponding command will be generated and sent to lens. Lens will return related response such as ACK or general response. In terminal, sent buffer and received buffer will be printed.

Examples

Response decoding example

Below is an example of decoding cameras response.

```
// Init variables.
const int dataBufferSize = 512;
uint8_t dataBuffer[dataBufferSize];
FujiProtocolParser parser;
int size = 0;
int arg1, arg2; // arguments that lens will return.
FujiCommand id; // id of previously sent command.

// Read data.
size = g_serialPort.read(dataBuffer, dataBufferSize);

// Check response size.
if (size <= 0)
{
    cout << " ERROR: no response" << endl;
    return false;
}

// Check response.
if (!parser.checkResponse(dataBuffer, size, id, arg1, arg2))
{
    cout << " ERROR: checkResponse(...)" << endl;
    return false;
}

/// Check id and read arguments ...
```

The command IDs are described in the FujiCommand structure in the **FujiProtocolParser.h** file.

Command encoding example

Below is an example of encoding command.

```
// Init variables.
const int dataBufferSize = 512;
uint8_t dataBuffer[dataBufferSize];
FujiProtocolParser parser;
int size = 0;
int arg1; // argument 1 you want to send to lens
int arg2; // argument 2 you want to send to lens
FujiCommand id; // id of command to send.
```

```
// Encode command.
if (!parser.getCommand(dataBuffer, size, id, arg1, arg2))
{
    return false;
}

// Send command.
if (g_serialPort.write(dataBuffer, size) != size)
{
    return false;
}
```

Build and connect to your project

Typical commands to build **FujiProtocolParser** library:

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

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

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

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(${PARENT}_SUBMODULE_FUJI_PROTOCOL_PARSER         ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_FUJI_PROTOCOL_PARSER)
    SET(${PARENT}_FUJI_PROTOCOL_PARSER               ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_FUJI_PROTOCOL_PARSER_TEST          OFF CACHE BOOL "" FORCE)
endif()

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

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

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} FujiProtocolParser)
```

Done!