FujiSxParser C++ library

v2.0.0

Table of contents

Overview

The **FujiSxParser** C++ library designed for encoding control commands and decoding responses from Fujinon Sx series CCTV cameras (block daylight cameras with integrated zoom lenses). Fujinon Sx cameras use custom Pelco-D communication protocol. 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 demo application to test communication with cameras via serial ports.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	04.05.2023	- First version.
1.0.1	10.05.2023	- decodeResponse bug is fixed.
2.0.0	29.12.2023	- Code reviewed. - Documentation updated. - Interface changed.

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
docs ----------------------------------------------- folder with camera specifications
    pelco-d_protocol_specification_for_sx800.pdf ---- control protocol specification
src  ----------------------------------------------- folder with library source code
    CMakeLists.txt ----------------------------------- CMake file
    FujiSxParser.h ----------------------------------- main library header file
    FujiSxParserVersion.h ---------------------------- header file with library version
    FujiSxParserVersion.h.in ------------------------- file for CMake to generate version
header
    FujiSxParser.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** (source code included in repository, Apache 2.0 license).

FujiSxParser class description

Class declaration

```
class FujiSxParser
{
public:
    /// Encode fuji command.
    bool getCommand(uint8_t data[7], int address, FujiSxCommand id, int arg1 = 0, int
arg2 = 0);

    /// Decode fuji response.
    FujiSxResponse decodeResponse(uint8_t nextByte, int& arg1, int& arg2);

    /// Get library version.
    static std::string getVersion();
};
```

getCommand method

getCommand(...) method encodes (prepares data) control command for camera. The camera 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 camera. For this purpose, the user must provide the following principle of data exchange with the camera: the user prepares a command for the camera (with getCommand(...) method) and sends it, then waits for a response (standard response time is 40 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 address, FujiSxCommand id, int arg1 = 0, int arg2 =
0);
```

Parameter	Value
data	Pointer to buffer for encoded data (command). Size of buffer should be >= 7 bytes.
address	Address of camera. **Default value 7**.
id	Command ID from FujiSxCommand enum.
arg1	First argument of command. The value of argument depends on command ID (see FujiSxCommand enum). Some commands don't have arguments.
arg2	Second optional argument of command. The value of argument depends on command ID (see FujiSxCommand 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

decodeResponse(...) 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). All messages to be exchanged with the camera are 7 bytes in size (according to the protocol specification). The library has an internal ring buffer of 7 bytes. 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 FujiSxResponse) and returns the response arguments. In all other cases, the library returns a **NONE** identifier. Method declaration:

```
FujiSxResponse decodeResponse(uint8_t nextByte, int& arg1, int& arg2);
```

Parameter	Value
nextByte	Next read byte from serial port.
arg1	Response argument 1. The value of argument depends on response ID (see FujiSxResponse enum). Some responses don't have arguments.
arg2	Response argument 2. The value of argument depends on response ID (see FujiSxResponse enum). Some responses don't have arguments.

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

getVersion method

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

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

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

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

Console output:

```
FujiSxParser version: 2.0.0
```

Data structures

FujiSxCommand

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

```
/**
 * @brief Fuji lens commands according to control protocol specification.
 */
enum class FujiSxCommand
{
    ////////////////////////// Main commands ////////////////////////////////

    /// Camera on, but not supported by Sx800.
    /// No arguments required.
    CAMERA_ON,

    /// Camera off, but not supported by Sx800.
    /// No arguments required.
    CAMERA_OFF,

    /// Iris close command.
    /// No arguments required.
    IRIS_CLOSE,

    /// Iris open command.
    /// No arguments required.
    IRIS_OPEN,

    /// Focus near command.
    /// No arguments required.
    FOCUS_NEAR,

    /// Focus far command.
    /// No arguments required.
    FOCUS_FAR,

    /// Zoom wide command.
    /// No arguments required.
    ZOOM_WIDE,

    /// Zoom tele command.
    /// No arguments required.
    ZOOM_TELE,

    /// No arguments required.
    DOWN,

    /// No arguments required.
    UP,

    /// No arguments required.
    LEFT,

    /// No arguments required.
    RIGHT,
```

```
    ////////////////////////// Extended commands ////////////////////////////

    /// Set zoom speed.
    /// Arguments:
    /// arg1 : 0x00 - Slowest speed. (=Lowest medium speed)
    ///         0x01 - Low medium speed.
    ///         0x02 - High medium speed.
    ///         0x03 - Highest speed. (=High medium speed)
    ///         others - Low medium speed
    SET_ZOOM_SPEED,

    /// Set focus speed.
    /// Arguments:
    /// arg1 : 0x00 - Slowest speed. (=High medium speed)
    ///         0x01 - Low medium speed. (=High medium speed)
    ///         0x02 - High medium speed.
    ///         0x03 - Highest speed. (=High medium speed)
    ///         others - Low medium speed
    SET_FOCUS_SPEED,

    /// Set auto focus on/off
    /// Arguments:
    /// arg1 : 0x00 - AF on
    ///         0x01 - AF off (MF)
    ///         0x02 - Quick AF
    ///         others - AF on
    AUTO_FOCUS_ON_OFF,

    /// ...

};
```

FujiSxResponse

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

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

    //Acknowledgement detected for sent command
    ACK,

    ///Time second is returned
    ///data1 = seconds
    TIME_SECOND,

    ///Time hour and minute are returned
    /// data1 = hour, data2 = minutes
    TIME_HOUR_MINUTE,
```

```
    ///Time month and day are returned
    /// data1 = month, data2 = date
    TIME_MONTH_DAY,

    /// Time year is returned
    /// data1 = year
    TIME_YEAR,

    ///Focus position is returned
    /// data1 = focus position
    FOCUS_POSITION,

    ///Zoom position is returned
    /// data1 = zoom position
    ZOOM_POSITION,

    ///Firmware version
    /// data1 = major firmware version , data2 = minor version
    FW_VERSION,

    ///Lens status
    /// data1 : 0x00 - No error
    ///         0x01 - Lens error
    LENS_STATUS,

    /// ...
};
```

Test application

Folder **FujiSxParser/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) and camera address (default 7):

```
================================================
FujiSxParser tester v2.0.0
================================================

Set COM port num (1,2,3,...): 2
Set baudrate (default : 9600): 9600
Enter the camera adress (default : 7): 7
```

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

```
Commands:
2 - IRIS_CLOSE
3 - IRIS_OPEN
4 - FOCUS_NEAR
```

```
5 - FOCUS_FAR
6 - ZOOM_WIDE
7 - ZOOM_TELE
12 - SET_ZOOM_SPEED
13 - SET_FOCUS_SPEED
14 - AUTO_FOCUS_ON_OFF
15 - AUTO_IRIS_ON_OFF
18 - SET_ZOOM_POSITION
28 - GET_ZOOM_POSITION
29 - GET_FOCUS_POSITION
31 - GET_FW_VERSION
32 - GET_LENS_STATUS
33 - SET_FOCUS_POSITION

For rest of commands check FujiSxParser.h file

Chose 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 the platform zoom tele command and an example of the focus near command.

```
// Get data from serial port.
uint8_t buffer[128];
int bytes = serialPort.readData(buffer, 128);

// Decoding response byte-by-byte.
for (int i = 0; i < bytes; ++i)
{
    // Put next byte to parser.
    FujiSxResponse response = parser.decodeResponse(buffer[i], data1, data2);

    // Check result.
    switch (response)
    {
    case FujiSxResponse::TIME_SECOND:
        cout << "Time second : " << data1 << endl;
        break;

        ///...
    }
}
```

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

Command encoding example

Below is an example of the platform zoom tele command and an example of the focus near command.

```
// Init variables.
uint8_t packet[7];
uint8_t cameraAddress = 7;

// Prepare zoom tele.
parser.getCommand(packet, cameraAddress, FujiSxCommand::ZOOM_TELE);

// Prepare zoom tele.
parser.getCommand(packet, cameraAddress, FujiSxCommand::FOCUS_NEAR);
```

Build and connect to your project

Typical commands to build **FujiSxParser** library:

```
cd FujiSxParser
mkdir build
cd build
cmake ..
make
```

If you want connect FujiSxParser 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 **FujiSxParser** as submodule by command (or copy repository files):

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

In you repository folder will be created folder **3rdparty/FujiSxParser** which contains files of **FujiSxParser** repository. New structure of your repository:

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

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(${PARENT}_SUBMODULE_FUJI_SX_PARSER                  ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_FUJI_SX_PARSER)
    SET(${PARENT}_FUJI_SX_PARSER                        ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_FUJI_SX_PARSER_TEST                   OFF CACHE BOOL "" FORCE)
endif()

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

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

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

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

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

Done!