StingrayParser C++ library

v2.0.0

Table of contents

- Overview
- Versions
- Library files
- StingrayParser class description
  - StingrayParser class declaration
  - getVersion method
  - getCommand method
  - decodeResponse method
- StingrayCommand enum
- Test application
- Build and connect to your project

Overview

StingrayParser C++ library version **2.0.0** is designed to help developers control Stingray **SWIR lenses**. The library includes basic methods for preparing commands (encoding) and interpreting the lens responses (decoding). It uses C++17 standard and compatible with any OS and CPU. The library provides simple interface and doesn't have third party dependencies. Also, the library provides demo application to test communication with lens via serial ports.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	20.06.2023	First version.
2.0.0	29.12.2023	- StingrayParser class interface updated. - Test application updated.

Library files

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

```
CMakeLists.txt ------------------------ Main CMake file
README.md ----------------------------- Documentation
docs ---------------------------------- Folder with STINGRAY datasheet
src ----------------------------------- Folder with library source code
    CMakeLists.txt -------------------- CMake file
    StingrayParser.h ------------------ Main library header file
    StingrayParser.cpp ---------------- C++ implementation file
    StingrayParserVersion.h ----------- Header file with library version
    StingrayParserVersion.h.in -------- File for CMake to generate version header
test ---------------------------------- Folder for test application files
    3rdaprty -------------------------- Folder with 3rdparty libraries
        CMakeLists.txt ---------------- CMake file for 3rdparty folder
        SerialPort -------------------- Serial port service library
    CMakeLists.txt -------------------- CMake file for test app
    main.cpp -------------------------- Source C++ file of demo app
```

StingrayParser class description

StingrayParser class declaration

StingrayParser .h file contains **StingrayParser** class declaration.

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

    /// Method to encode Stingray lens command.
    bool getCommand(uint8_t* data, int& size, StingrayCommand id, int arg = 0);

    /// Process response for STATUS command.
    void decodeResponse(uint8_t* data, int size,int &zoom, int &focus);
};
```

getVersion method

getVersion() method returns string of current version of **StingrayParser** class. Method declaration:

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

Method can be used without **StingrayParser** class instance. Example:

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

Console output:

```
StingrayParser version: 2.0.0
```

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. Method declaration:

```
bool getCommand(uint8_t* data, int& size, StingrayCommand id, int arg = 0);
```

Parameter	Description
data	Pointer to buffer for encoded data.
size	Encoded command size.
id	Command ID from **StingrayCommand** enum
arg	Command argument. The value of argument depends on command ID (see **StingrayCommand** enum). Some commands don't have arguments.

Returns: TRUE if command is prepared or FALSE if not.

Bellow there is an example of zoom tele command and an example of the focus to absolute position command.

```
// Init variables.
uint8_t buffer[256];
uint8_t size;

// Prepare zoom tele command.
StingrayParser parser;
parser.getCommand(buffer, size, StingrayCommand::ZOOM_TELE);

// Send command.
serialPort.write(buffer, size);

// Prepare focus to abs pos command.
parser.getCommand(buffer, size, StingrayCommand::FOCUS_TO_ABS_POS, 5000);
```

```
// Send command.
serialPort.write(buffer, size);
```

decodeResponse method

decodeResponse(...) method process response and give zoom or focus position if available. In version 2.0.0 of the library only zoom and focus position response decoding implemented. Method declaration:

```
 void decodeResponse(uint8_t* data, int size,int &zoom, int &focus);
```

Parameter	Description
data	Pointer to data.
size	Data size.
zoom	Returned **zoom position** value or **-1** if no zoom position in response.
focus	Returned **focus position** value or **-1** if no focus position in response.

To retrieve the zoom, focus position, or both, you must initially send `REQUEST_ZOOM_POS`, `REQUEST_FOCUS_POS`, or `REQUEST_STATUS` respectively. The data buffer received in response should be passed to the **decodeResponse** function. This function updates the **zoom** variable with the new zoom value, or sets it to **-1** if the received buffer does not contain zoom data. The same process applies to the **focus** variable. Please note that the buffer might contain only a portion of the response or multiple responses. In the case of partial data, **decodeResponse** will update the **zoom** or **focus** variables upon its next call, which should contain the end of the response. Bellow there is example of sending REQUEST_STATUS request and getting zoom and focus position:

```
uint8_t buffer[256];
int size = 0;
StingrayParser parser;

while (true)
{
    // Prepare reaquest.
    parser.getCommand(buffer, size, StingrayCommand::REQUEST_STATUS);

    // Send request.
    serialPort.write(buffer, size);

    // Wait some time. Recommended 40 msec.

    // Read response from serial port. You can wait 40 msec.
    int bytes = serialPort.read(buffer, 256);

    // Decode response.
    int zoomPos = 0;
    int focusPos = 0;
    parser.decodeResponse(buffer, bytes, zoomPos, focusPos);
```

```
    // Check if zoom updated
    if (zoomPos != -1) {
        // zoomPos contains updated value of zoom
    }

    // Check if focus updated
    if (focusPos != -1) {
        // focusPos contains updated value of focus
    }
}
```

StingrayCommand enum

```
enum class StingrayCommand
{
    /// Focus to absolute position. Argument: focus position 0 - 10000.
    FOCUS_TO_ABS_POS = 1,
    /// Focus to relative position. Argument: focus increment +-(0 - 10000).
    FOCUS_TO_RELATIVE_POS,
    /// Move focus far. Argument: focus speed 0 - 255.
    FOCUS_FAR,
    /// Move focus far. Argument: focus speed 0 - 255.
    FOCUS_NEAR,
    /// Focus stop. No arguments.
    FOCUS_STOP,
    /// Request focus position. No arguments.
    REQUEST_FOCUS_POS,
    /// Store current focus position for next system boot/reset. No arguments.
    STORE_FOCUS_POS,
    /// Zoom to absolute position. Argument: zoom position 0 - 10000.
    ZOOM_TO_ABS_POS,
    /// Zoom to relative position. Argument: focus increment +-(0 - 10000).
    ZOOM_TO_RELATIVE_POS,
    /// Move zoom wide. Argument: zoom speed 0 - 255.
    ZOOM_WIDE,
    /// Move zoom tele. Argument: zoom speed 0 - 255.
    ZOOM_TELE,
    /// Zoom stop. No arguments.
    ZOOM_STOP,
    /// Request zoom position. No arguments.
    REQUEST_ZOOM_POS,
    /// Store current zoom position for next system boot/reset. No arguments.
    STORE_ZOOM_POS,
    /// Request status. No arguments.
    REQUEST_STATUS,
    /// Request system information. No arguments.
    REQUEST_SYSTEM_INFO,
    /// Change verbose level. Arguments: 0, 1 or 2.
    CHANGE_VERBOSE_LEVEL,
    /// Reset. No arguments.
    RESET,
```

```
    /// Stop zoom and focus motion. No arguments.
    STOP,
    /// Query zoom/focus motors tolerance. No arguments.
    REQUEST_TOLERANCE,
    /// Command Acknowledgement Toggle. Argument: 0 or 1.
    SET_ACK_RESPONSE
};
```

Table 2 - StingrayCommand enum class description.

Parameter	Description	Argument
FOCUS_TO_ABS_POS	Move **focus** to absolute position	Focus position: **0 - 10000**
FOCUS_TO_RELATIVE_POS	Move **focus** to relative position	Focus increment: **-10000 - +10000**
FOCUS_FAR	Move **focus** far	Focus speed: **0 - 255**
FOCUS_NEAR	Move **focus** near	Focus speed: **0 - 255**
FOCUS_STOP	Stop **focus** movement	No arguments
REQUEST_FOCUS_POS	Request **focus** position	No arguments
STORE_FOCUS_POS	Store current **focus** position for next system boot/reset	No arguments
ZOOM_TO_ABS_POS	Move **zoom** to absolute position	Zoom position: **0 - 10000**
ZOOM_TO_RELATIVE_POS	Move **zoom** to relative position	Zoom increment: **-10000 - +10000**
ZOOM_WIDE	Move **zoom** wide	Zoom speed: **0 - 255**
ZOOM_TELE	Move **zoom** tele	Zoom speed: **0 - 255**
ZOOM_STOP	Stop **zoom** movement	No arguments
REQUEST_ZOOM_POS	Request **zoom** position	No arguments
STORE_ZOOM_POS	Store current **zoom** position for next system boot/reset	No arguments
REQUEST_STATUS	Request status. Current zoom and focus position will be returned.	No arguments
REQUEST_SYSTEM_INFO	Request system information	No arguments

Parameter	Description	Argument
CHANGE_VERBOSE_LEVEL	Change verbose level	Arguments: **0, 1** or **2** 0 - Default. Simple output for computer use 1 - Focus will report step count when reaching destination instead of NOERROR, otherwise same as 0. 2 - Provide error codes in plain English
RESET	Reset.	No arguments
STOP	Stop **zoom** and **focus** motion	No arguments
REQUEST_TOLERANCE	Query **zoom & focus** motors tolerance	No arguments
SET_ACK_RESPONSE	Acknowledgement Toggle. When this command is on most commands that don't normally have a response (i.e. 'F=XXXX' will now respond with 'COMMAND=ACK' to signal the lens driver successfully received the command.	Argument: **0** or **1** 0 - off 1 - on

Test application

Folder **StingrayParser/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):

```
==========================================
StingrayParser test v2.0.0
==========================================

Set COM port num (1,2,3,...): 2
Serial port open!
```

After user can chose command (enter command ID):

```
==========================================

1 - FOCUS_TO_ABS_POS
2 - FOCUS_TO_RELATIVE_POS
3 - FOCUS_FAR
4 - FOCUS_NEAR
5 - FOCUS_STOP
6 - REQUEST_FOCUS_POS
```

```
7 - STORE_FOCUS_POS
8 - ZOOM_TO_ABS_POS
9 - ZOOM_TO_RELATIVE_POS
10 - ZOOM_WIDE
11 - ZOOM_TELE
12 - ZOOM_STOP
13 - REQUEST_ZOOM_POS
14 - STORE_ZOOM_POS
15 - REQUEST_STATUS
16 - REQUEST_SYSTEM_INFO
17 - CHANGE_VERBOSE_LEVEL
18 - RESET
19 - STOP
20 - REQUEST_TOLERANCE
21 - SET_ACK_RESPONSE
==========================================

Chose command:
```

Then, corresponding command will be generated and sent to lens.

Build and connect to your project

Typical commands to build **StingrayParser** library:

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

If you want to connect **StingrayParser** library to your CMake project as source code, you can do the following. For example, if your repository has structure:

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

You can add repository **StingrayParser** as git submodule by commands (only if you have access to GitHub repository):

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

In your repository folder, a new **3rdparty/StingrayParser** folder will be created, which contains files from **StingrayParser** repository. If you don't have access to GitHub repository, copy **StingrayParser** repository folder to **3rdparty** folder to your repository. The new structure of your repository will be as follows:

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

Create CMakeLists.txt file in **3rdparty** folder. CMakeLists.txt should be containing:

```
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(${PARENT}_SUBMODULE_STRINGRAY_PARSER                ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_STRINGRAY_PARSER)
    SET(${PARENT}_STRINGRAY_PARSER                      ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_STRINGRAY_PARSER_TEST                 OFF CACHE BOOL "" FORCE)
endif()

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

File **3rdparty/CMakeLists.txt** adds folder **StingrayParser** to your project and excludes test applications and examples from compiling. The new structure of your repository will be:

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

Next, you need to include the '3rdparty' folder in the main **CMakeLists.txt** file of your repository. Add the following string at the end of your main **CMakeLists.txt**:

```
add_subdirectory(3rdparty)
```

Next, you have to include **StingrayParser** library in your **src/CMakeLists.txt** file:

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

Done!