ViscaParser C++ library

v3.0.1

Table of contents

Overview

ViscaParser C++ library version **3.0.0** is designed for parsing VISCA (trademark of Sony Corporation) protocol messages. The library is designed according to the VISCA protocol description for the camera **Sony FCB-EV7520A**. The library allows you to use it to control other cameras that support the VISCA protocol. Before encoding and decoding specific commands, please check the source code of the library for the corresponding command according to the technical description of your camera. The library provides: VISCA messages encoding, VISCA messages decoding and searching for messages in a continuous data set. Folder **examples** contains examples how to use the library. The library has been developed with C++17 standard and doesn't have third party dependencies.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	20.01.2022	First version
2.0.0	02.06.2022	- New commands added. - Bugs fixed.

Version	Release date	What's new
3.0.0	03.04.2023	- Class name changed from ViscaProtocolParser to ViscaParser. - ViscaProtocolParserDataStructures.h file excluded. - Documentation updated.
3.0.1	11.02.2024	- Documentation 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.
src -------------------------------- Folder with library source code.
    CMakeLists.txt ----------------- CMake file of the library.
    ViscaParser.h ------------------ Main library header file.
    ViscaParser.cpp ---------------- C++ implementation file.
    ViscaParserVersion.h ----------- Header file with library version.
    ViscaParserVersion.h.in -------- CMake file to generate version header.
examples --------------------------- Folder with simple example files.
    CMakeLists.txt ----------------- CMake file to include applications.
    ViscaParserDemoApplication ----- Forlder with ViscaParser demo application.
        CMakeLists.txt ------------- CMake file of the application.
        json.hpp ------------------- Source file of library to work with JSON files.
        main.cpp ------------------- Main source file of the application.
        SerialPort.cpp ------------- Source code file of the serial port library.
        SerialPort.h --------------- Header file of the serial port library.
        SerialPortVersion.h -------- Header file with version of the serial port library.
    ViscaParserTest ---------------- Forlder with ViscaParser demo application.
        CMakeLists.txt ------------- CMake file of the application.
        main.cpp ------------------- Main source file of the application.
        SerialPort.cpp ------------- Source code file of the serial port library.
        SerialPort.h --------------- Header file of the serial port library.
        SerialPortVersion.h -------- Header file with version of the serial port library.
```

Additionally demo application depends on OpenCV which provides user interface functions.

ViscaParser class description

Class declaration

ViscaParser.h contains main class **ViscaParser** declaration and the file library contains a declaration of **ViscaPackets** enum, which contains list commands and messages supported by VISCA protocol. Class declaration:

```
class ViscaParser
```

```
{
public:

    /// Encode COMMAND or ENQUIRY COMMAND.
    bool encodeCommand(
        visca::ViscaPackets commandId, uint8_t* packet,
        uint32_t& packetSize, uint32_t cameraAddress,
        uint32_t param1 = 0, uint32_t param2 = 0,
        uint32_t param3 = 0, uint32_t param4 = 0,
        uint32_t param5 = 0, uint32_t param6 = 0,
        uint32_t param7 = 0, uint32_t param8 = 0,
        uint32_t param9 = 0, uint32_t param10 = 0);

        /// Decode input data byte-by-byte.
        visca::ViscaPackets decodeData(
            uint8_t nextByte, uint32_t cameraAddress,
            uint32_t& param1, uint32_t& param2,
            uint32_t& param3, uint32_t& param4,
            uint32_t& param5, uint32_t& param6,
            uint32_t& param7, uint32_t& param8,
            uint32_t& param9, uint32_t& param10);

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

getVersion method

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

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

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

```
std::cout << "ViscaParser class version: " << ViscaParser::getVersion();
```

Console output:

```
ViscaParser class version: 3.0.1
```

encodeCommand method

encodeCommand(...) method intended to encode VISCA command to send to camera. Method declaration:

```
bool encodeCommand(
        visca::ViscaPackets commandId, uint8_t* packet,
        uint32_t& packetSize, uint32_t cameraAddress,
        uint32_t param1 = 0, uint32_t param2 = 0,
        uint32_t param3 = 0, uint32_t param4 = 0,
        uint32_t param5 = 0, uint32_t param6 = 0,
        uint32_t param7 = 0, uint32_t param8 = 0,
        uint32_t param9 = 0, uint32_t param10 = 0);
```

Parameter	Value
commandId	ID of VISCA command according to ViscaPackets enum (declared in ViscaParser.h file). ViscaPackets enum repeats names of commands from VISCA protocol specification.
packetSize	Pointer to the packet buffer to be filled by the method. The size of the buffer must be at least **24 bytes**.
cameraAddress	Camera address. Usually equals 1 or can be reassigned with the command **visca::ViscaPackets::COMMAND_AddressSet**. The camera address can be in the range from 1 to 7.
param1	Parameter 1. The value of each parameter depends on the **commandId** (see ViscaPackets enum).
param2	Parameter 2. The value of each parameter depends on the **commandId** (see ViscaPackets enum).
param3	Parameter 3. The value of each parameter depends on the **commandId** (see ViscaPackets enum).
param4	Parameter 4. The value of each parameter depends on the **commandId** (see ViscaPackets enum).
param5	Parameter 5. The value of each parameter depends on the **commandId** (see ViscaPackets enum).
param6	Parameter 6. The value of each parameter depends on the **commandId** (see ViscaPackets enum).
param7	Parameter 7. The value of each parameter depends on the **commandId** (see ViscaPackets enum).
param8	Parameter 8. The value of each parameter depends on the **commandId** (see ViscaPackets enum).
param9	Parameter 9. The value of each parameter depends on the **commandId** (see ViscaPackets enum).
param10	Parameter 10. The value of each parameter depends on the **commandId** (see ViscaPackets enum).

Returns: TRUE if the command encoded or FALSE if not.

Example:

```
// Encoding
uint8_t packetData[24];
uint32_t packetSize = 0;
visca_parser.encodeCommand(cr::visca::ViscaPackets::INQUIRY_CAM_ZoomPosInq, packetData,
packetSize, 1, 1);

// Sending to serial port.
serialPort.sendData(packetData, packetSize);
```

decodeData method

decodeData(...) method intended for decoding input data received from the camera, processing it byte by byte. This process is essential for identifying connected packets within the data stream obtained from the serial port. To be detected right user should send request first. in **encodeCommand(...)** method the library remember last encoded messages and uses it for decoding. Method declaration:

```
visca::ViscaPackets decodeData(
            uint8_t nextByte, uint32_t cameraAddress,
            uint32_t& param1, uint32_t& param2,
            uint32_t& param3, uint32_t& param4,
            uint32_t& param5, uint32_t& param6,
            uint32_t& param7, uint32_t& param8,
            uint32_t& param9, uint32_t& param10);
```

Parameter	Value
nextByte	Next byte in data to decode.
cameraAddress	Camera address. Usually equals 1 or can be reassigned with the command **visca::ViscaPackets::COMMAND_AddressSet**. The camera address can be in the range from 1 to 7.
param1	Output parameter 1. The value of each parameter depends on the **commandId** (see ViscaPackets enum).
param2	Output parameter 2. The value of each parameter depends on the **commandId** (see ViscaPackets enum).
param3	Output parameter 3. The value of each parameter depends on the **commandId** (see ViscaPackets enum).
param4	Output parameter 4. The value of each parameter depends on the **commandId** (see ViscaPackets enum).
param5	Output parameter 5. The value of each parameter depends on the **commandId** (see ViscaPackets enum).
param6	Output parameter 6. The value of each parameter depends on the **commandId** (see ViscaPackets enum).
param7	Output parameter 7. The value of each parameter depends on the **commandId** (see ViscaPackets enum).

Parameter	Value
param8	Output parameter 8. The value of each parameter depends on the **commandId** (see ViscaPackets enum).
param9	Output parameter 9. The value of each parameter depends on the **commandId** (see ViscaPackets enum).
param10	Output parameter 10. The value of each parameter depends on the **commandId** (see ViscaPackets enum).

Returns: ID of message according to enumeration **ViscaPackets** enum. If the method decoded a message, it returns the message identifier. If no message is decoded, the method returns the identifier **visca::ViscaPackets::UNDEFINED_PACKET**.

Example:

```
// Read data from serial port.
int numberOfBytes = serial_port.readData(packetData, packetSize);

// Decode data.
if (numberOfBytes <= 0)
{
    std::cout << "No response" << std::endl;
}
else
{
    for (int i = 0; i < numberOfBytes; ++i)
    {
        uint32_t param1, param2, param3, param4, param5, param6, param7, param8, param9,
param10 = 0;
        switch(visca_parser.decodeData(packetData[i], 1, param1, param2, param3, param4,
param5,
                                       param6, param7, param8, param9, param10))
        {
        case cr::visca::ViscaPackets::ACKNOWLEDGE:
            std::cout << "ACKNOWLEDGE" << std::endl;
            break;
        case cr::visca::ViscaPackets::COMPLETION_COMMANDS:
            std::cout << "COMPLETION_COMMANDS" << std::endl;
            break;
        case cr::visca::ViscaPackets::COMPLETION_INQUIRY:
            std::cout << "COMPLETION_INQUIRY" << std::endl;
            break;
        case cr::visca::ViscaPackets::REPLY_CAM_ZoomPos:
            std::cout << "ZOOM POSITION" << std::endl;
            break;
        default:
            break;
        }
    }
}
```

ViscaPackets enum

ViscaPackets enum declared in ViscaParser.h file. Enum represents all commands and replies described in VISCA protocol. The enum delaration:

```
enum class ViscaPackets
{
    /// Not VISCA packets.
    UNDEFINED_PACKET,

    /// ERROR MESSAGES.
    ERROR_Message_Length,
    ERROR_Syntax,
    ERROR_Command_Buffer_Full,
    ERROR_Command_Canceled,
    ERROR_No_Socket,
    ERROR_Command_Not_Executable,

    /// COMMAND MESSAGES.

    /// Set address value. Parameters:
    /// camera address.
    COMMAND_AddressSet,

    /// Clears the command buffers in the camera and cancels the command
    /// currently being executed. No parameters.
    COMMAND_IF_Clear_Broadcast,
    /// Clears the command buffers in the camera and cancels the command
    /// currently being executed. Parameters: camera address.
    COMMAND_IF_Clear,

    /// Cancels the command currently being executed. Parameters:
    /// camera address
    /// param_1 - socket number (1 or 2).
    COMMAND_CommandCancel,

    /// Camera power On. Parameters:
    /// camera address.
    COMMAND_CAM_Power_On,
    /// Camera power Off (Standby). Parameters:
    /// camera address.
    COMMAND_CAM_Power_Off,
...
```