RIFTEK
Sensors & Instruments

RF627 SOFTWARE DEVELOPMENT KIT AND SERVICE PROTOCOL

Developer guide

22, Logoisky tract, Minsk
220090, Republic of Belarus
tel/fax: +375 17 281 36 57
info@riftek.com
www.riftek.com

Contents

1. General information

This document provides technical guidance for using the RF627 SDK and the service protocol for laser scanners RF627 series.

The RF627 SDK (Software Development Kit) and the service protocol make it possible for the user to create custom applications to work with laser scanners RF627 series manufactured by RIFTEK.

2. RF627 SDK

2.1. System requirements

- Operating system Windows 7 and later.
- Microsoft Visual C++ Runtime Redistributable for Windows x64/x86.

2.2. SDK contents

The RF627 Software Development Kit includes:

Directory	Description
doc	Developer Guide.
include	Header files.
x64	LIB and DLL files for x64 and x86.
x86	

2.3. Class name

Class name: **rf627**.

2.4. SDK functions

2.4.1. Library initialization

```
static bool init();
```
Library initialization. This method must be called once before using the library.
Returns *true* on success.

```
static void cleanup();
```
This method must be called at the end of the program to cleanup the memory allocated by the library.

2.4.2. Searching for scanners

```
static rf627_list search(bool *ok = nullptr);
```
Performs the search for scanners over network. Returns a list of found scanners.
`ok`: optional pointer to *boolean*, which is set to *false* if error occurs.

2.4.3. Connecting / disconnecting a scanner

```
bool connect();
```
This method creates and configures the UDP sockets for data exchange with the scanner. Returns *false* on error.

```
void disconnect();
```
This method closes sockets opened by the *connect* method.

2.4.4. Scanner parameters

2.4.4.1. Reading / writing configuration parameters

```
bool read_params(rf627_user_params_t* pparams = nullptr);
```
Reads configuration parameters from RAM. Returns *false* on error.
`pparams`: a pointer to the configuration parameters structure to store.

```
bool write_params(rf627_user_params_t* pparams = nullptr);
```
Writes configuration parameters to RAM. Returns *false* on error.
`pparams`: a pointer to the configuration parameters structure to write.

2.4.4.2. Saving parameters to flash memory

```
bool flush_params();
```
Saves configuration parameters to flash memory. Returns *false* on error.

2.4.4.3. Restoring default parameters

```
bool reset_params();
```
Restores configuration parameters to factory set. Returns *false* on error.

2.4.4.4. Reading system parameters

```
bool read_sysmon_params(rf627_sysmon_params_t* psysmon = nullptr);
```
Reads system monitor values from the scanner memory. This is also being read by *read_params* method as part of *ref627_user_params* structure. Returns *false* on error.
`psysmon`: a pointer to *sysmon* structure to store.

2.4.5. Receiving profiles

```
bool get_result(rf627_profile& profile);
```
Reads profile data from stream. Returns *false* on error.
`profile`: reference to the *profile* structure.

2.4.6. Receiving image

```
bool get_image(uint8_t* ppixmap);
```
Requests and reads an image from scanner. Returns *false* on error.
`ppixmap`: a pointer to 8-bit pixel array. Array size is *RF627_IMAGE_SIZE*. Each byte represents a pixel brightness in range of 0-255 (black to bright).

2.4.7. Firmware update

```
bool write_firmware_image(const char* file_name);
```
Writes the firmware image to memory. Returns *false* on error.
`file_name`: firmware file name (.rf627).

```
bool flush_firmware_image();
```
Saves the firmware image transferred by *write_firmware_image* to flash. The scanner will be rebooted on success to boot the new firmware. Returns *false* on error.

2.4.8. Reboot

```
bool reboot();
```
Reboots the scanner. Returns *false* on error.

2.4.9. Log

```
bool read_log(uint32_t nstart_line, rf627_log_record_t *plog_entries, int
nlines);
```
Reads log entries from the scanner memory. Returns *false* on error.
`nstart_line`: a number of the first line to read.
`plog_entries`: a pointer to array of *rf627_log_record_t* structures.
`nlines`: a number of lines to read, the maximum value is RF627_MAX_LOG_ENTRIES_PER_PAYLOAD.

```
uint32_t read_log_record_count();
```
Gets a total number of log entries from the scanner. Returns a number of entries, or 0 on error.

2.4.10. Error messages

```
const char *error_msg();
```
Gets a pointer to the last error message.

2.4.11. SDK version

```
static uint32_t version();
```
Returns the SDK version.

2.5. Data structures

```
struct rf627_point
{
    double x;
    double z;
};
Coordinates of a profile point (millimeters)

struct rf627_profile
{
    rf627_stream_msg_t header;
    std::vector<rf627_point> points;
};
Profile read from stream

typedef enum: uint8_t
{
    DTY_PixelsNormal        = 0x10,   // Pixels (up to 648 points)
    DTY_ProfileNormal       = 0x11,   // Profile (up to 648 points)
    DTY_PixelsInterpolated  = 0x12,   // 2x interpolated pixels (up to 1296)
    DTY_ProfileInterpolated = 0x13    // 2x interpolated profile (up to 1296)
} data_type_t;
Profile type

#pragma pack(push,1)

typedef struct
{
    data_type_t data_type;     // Profile type (one of data_type_t enum)
    uint8_t     flags;         // Flags (bit 7 indicates that packet requires
acknowledgment from host)
    uint16_t    device_type;   // 627
    uint32_t    serial_number;
    uint64_t    system_time;

    uint8_t     proto_version_major;
    uint8_t     proto_version_minor;
    uint8_t     hardware_params_offset;
```

```
    uint8_t     data_offset;
    uint32_t    packet_count;  // Seq. number of packet emitted
    uint32_t    measure_count; // Seq. number of measurement

    uint16_t    zmr;           // Measurement range by Z
    uint16_t    xemr;          // Range by X at end of Z range
    uint16_t    discrete_value;
    uint8_t     reserved_0[14];

    uint32_t    exposure_time; // Exposure time used to take a measurement
    uint32_t    laser_value;
    uint32_t    step_count;    // STEP value in STEP/DIR mode
    uint8_t     dir;           // DIR value
    uint8_t     reserved_1[3];
}
rf627_stream_msg_t;
#pragma pack(pop)
A header of profile packet

#pragma pack(push,1)
typedef struct
{
    char        name[64];      // Readable name of scanner
    uint16_t    device_id;     // 627
    uint32_t    serial_number;
    uint32_t    firmware_version;
    uint32_t    hardware_version;
    uint32_t    config_version;
    uint32_t    fsbl_version;
    uint32_t    z_begin;       // Beginning of Z range
    uint32_t    z_range;       // Measurement range by Z
    uint32_t    x_smr;         // Meas. range by X at start of Z range
    uint32_t    x_emr;         // Meas. range by X at end of Z range
    uint8_t     reserved_0[36];

    uint16_t    eth_speed;     // 100 or 1000 (Mbps)
    uint32_t    ip_address;
    uint32_t    net_mask;
    uint32_t    gateway_ip;
    uint32_t    host_ip;       // IP address of host receiving data stream
    uint16_t    stream_port;   // Host port number of data stream
    uint16_t    http_port;
    uint16_t    service_port;
    uint16_t    eip_broadcast_port;
    uint16_t    eip_port;
    uint8_t     hardware_address[6];
    uint8_t     reserved_1[26];

    uint32_t    max_payload_size;
    uint8_t     reserved_2[32];

    uint8_t     stream_enabled;        // Nonzero if data stream is enabled
    uint8_t     stream_format;         // Profile type (data_type_t & 0x0F)
    uint8_t     reserved_3[32];

    uint8_t     reserved_4[256];
}
rf627_device_info_t;
#pragma pack(pop)
Scanner information
```

```
#pragma pack(push,1)
typedef struct
{
    char        name[64];      // Readable scanner name
    uint8_t            reserved[128];
}
rf627_general_params_t;
#pragma pack(pop)

#pragma pack(push,1)
typedef struct
{
    int16_t            fpga_temp;      // FPGA temperature (Celsius * 10)
    uint8_t            reserved[80];
}rf627_sysmon_params_t;
#pragma pack(pop)

#pragma pack(push,1)
typedef struct
{
    uint8_t            dhs;            // Enable double speed mode
    uint8_t            gain_analog;
    uint8_t            gain_digital;
    uint32_t    exposure;              // Exposure time, ns
    uint32_t    max_exposure;
    uint32_t    frame_rate;            // Frame rate limitation
    uint32_t    max_frame_rate;
    uint8_t     reserved_0;
    uint8_t     auto_exposure;         // Enable auto-exposure
    uint8_t     frame_by_request;
    uint8_t            reserved_1[61];
}rf627_sensor_params_t;
#pragma pack(pop)

#pragma pack(push,1)
typedef struct
{
    uint8_t            enable;         // Switch scanner to RF625 emulation
mode
    uint16_t    tcp_port;
    uint8_t            reserved[32];
}rf627_rf625compat_params_t;
#pragma pack(pop)

#pragma pack(push,1)
typedef struct
{
    uint8_t            enable;         // Manually enable region of interest
    uint8_t            active;
    uint16_t    window_height;
    uint8_t            position_mode;  // 0 - manual, nonzero - auto
    uint16_t    window_top;
    uint16_t    current_window_top;    // Current ROI top in auto mode
    uint16_t    profile_size;          // Required profile size
    uint8_t            reserved[80];
}rf627_roi_params_t;
#pragma pack(pop)

#pragma pack(push,1)
typedef struct
```

```
{
    uint16_t    speed;                          // 100 or 1000 Mbps
    uint8_t             autonegotiation;        // Auto-detect speed
    uint32_t    ip_address;
    uint32_t    net_mask;
    uint32_t    gateway_ip;
    uint32_t    host_ip;
    uint16_t    stream_port;
    uint16_t    http_port;
    uint16_t    service_port;
    uint16_t    eip_broadcast_port;
    uint16_t    eip_port;
    uint8_t             reserved[64];
}rf627_network_params_t;
#pragma pack(pop)

#pragma pack(push,1)
typedef struct
{
    uint8_t             enable;         // Enable data stream
    uint8_t             format;         // Profile type (data_type_t & 0x0F)
    uint8_t             ack;            // Each data packet requires
acknowledgment (0 is off, default)
    uint8_t             reserved[32];
}rf627_stream_params_t;
#pragma pack(pop)

#pragma pack(push,1)
typedef struct
{
    uint32_t    brightness_threshold;
    uint8_t             stg1_filter_width;
    uint8_t             stg1_processing_mode;
    uint8_t             stg2_reduce_noise;
    uint32_t    frame_rate;
    uint8_t             reserved[60];
}rf627_image_processing_params_t;
#pragma pack(pop)

#pragma pack(push,1)
typedef struct
{
    uint8_t             enable;         // 0 to turn laser off
    uint8_t             auto_level;
    uint16_t    level;
    uint8_t             reserved[32];
}rf627_laser_params_t;
#pragma pack(pop)

#pragma pack(push,1)
typedef struct
{
    uint16_t    params_mask;            // Each bit indicates if following
parameters are customizable in selected mode (LSB is in1_enable, etc.)
    uint8_t             in1_enable;
    uint8_t             in1_mode;
    uint32_t    in1_delay;
    uint8_t             in1_decimation;
    uint8_t             in2_enable;
    uint8_t             in2_mode;
```

```
    uint8_t             in2_invert;
    uint8_t             in3_enable;
    uint8_t             in3_mode;
    uint8_t             reserved[12];
}rf627_inputs_preset_t;
#pragma pack(pop)

#pragma pack(push,1)
typedef struct
{
    uint8_t                 preset_index;     // Selected mode
    rf627_inputs_preset_t   params[12];
    uint8_t                 reserved[32];
}rf627_inputs_params_t;
#pragma pack(pop)

#pragma pack(push,1)
typedef struct
{
    uint8_t             out1_enable;
    uint8_t             out1_mode;
    uint32_t    out1_delay;
    uint32_t    out1_pulse_width;
    uint8_t             out1_invert;
    uint8_t             out2_enable;
    uint8_t             out2_mode;
    uint32_t    out2_delay;
    uint32_t    out2_pulse_width;
    uint8_t             out2_invert;
    uint8_t             reserved[32];
}rf627_outputs_params_t;
#pragma pack(pop)

#pragma pack(push,1)
typedef struct
{
    rf627_general_params_t          general;
    rf627_sysmon_params_t           sysmon;
    rf627_rf625compat_params_t      rf625_compat;
    rf627_sensor_params_t           sensor;
    rf627_roi_params_t              roi;
    rf627_network_params_t          network;
    rf627_stream_params_t           stream;
    rf627_image_processing_params_t image_processing;
    rf627_laser_params_t            laser;
    rf627_inputs_params_t           inputs;
    rf627_outputs_params_t          outputs;
    uint8_t                         reserved[283];
}rf627_user_params_t;
#pragma pack(pop)
```

Triggering modes:

0 - Internal Clock
1 - External Trigger
2 - 1-phase Encoder
3 - 1-phase Encoder w/Zero
4 - 2-phase Encoder
5 - 2-phase Encoder w/Zero
6 - Step/Dir
7 - Ext. Trigger/Int. Clock
8 - Software Request

3. Service protocol

3.1. Profile transmission format

The result of processing the image frames coming from the CMOS sensor is the profile of the laser beam reflected from the controlled surface, presented in one of four formats. Each profile is transmitted by the scanner as a separate UDP packet to the network address and port specified on the web page of the scanner in the **Network** tab.

The user data area (payload) of the UDP packet contains data from the scanner, divided into the header and the profile data.

The groups of header fields:

- **Message fingerprint** - Fields of this group form a unique message identifier used to confirm delivery of the UDP packet.
- **Application specific data** - Specific data for a particular application of the scanner, intended to accommodate information on customer's request (for example, additional counters, the state of the inputs at the time of receiving the profile, ROI parameters, etc.).
- **Hardware parameters** - Scanner hardware parameters at the moment of starting the frame exposure.

The header has the following structure:

Address	Type	Field name	Comment	Group
0	uint8_t	Data_type	Data type marker: 0x10 - «raw profile»; 0x11 - «calibrated profile»; 0x12 - «extended raw profile»; 0x13 - «extended calibrated profile».	Message fingerprint
1	uint8_t	Flags	Packet flags: Flags[7] – packet delivery confirmation; Flags[6:0] – reserve.	
2	uint16_t	Device_ID	Device type identifier (627 always).	
4	uint32_t	Serial_number	Scanner serial number.	
8	uint64_t	System_time	System time (time after turning on the scanner), ns. The start time of the frame exposure.	
16	uint8_t	Protocol_Version_Major	The current version of the protocol.	
17	uint8_t	Protocol_Version_Minor	The current version of the protocol revision.	
18	uint8_t	Hardware_params_shift	Start address of hardware parameters during measurement. It's used when resizing the "Application specific data" area.	
19	uint8_t	Data_shift	Start address of data in packet. It's used when resizing the "Hardware parameters" area.	
20	uint32_t	Software_packets_counter	Counter of sent packets with profiles.	
24	uint32_t	Measures_counter	Hardware counter of performed measurements.	
28	uint16_t	ZMR	Measurement range by Z axis, 0.1 * mm.	
30	uint16_t	XEMR	Measurement range by X axis in the end of the range, 0.1 * mm.	
32	uint16_t	Discrete_Value	Discretization of measurement range. For profiles: 16384, for pixels: 32.	
34	uint16_t	Reserved		Application specific data
36	uint16_t	Reserved		
38	uint16_t	Reserved		
40	uint16_t	Reserved		

Address	Type	Field name	Comment	Group
42	uint16_t	Reserved		
44	uint16_t	Reserved		
46	uint16_t	Reserved		
48	uint32_t	Exposure_time	CMOS exposure time, ns.	Hardware parameters
52	uint32_t	Laser_value	Laser power level, ns.	
56	uint32_t	Step_counter	Step value in STEP/DIR mode, or encoder value.	
60	uint8_t	Dir	Dir value in STEP/DIR mode.	
61	uint8_t[3]	Reserved		

Depending on the format, profile data have the following structure.

Raw profile format:

Address	Data type	Field name	Comment
64	uint16_t	Z[0]	Z value (discrete) of point 1.
66	uint16_t	Z[1]	Z value (discrete) of point 2.
68	uint16_t	Z[2]	Z value (discrete) of point 3.
70	uint16_t	Z[3]	Z value (discrete) of point 4.
...	...	...	...
1358	uint16_t	Z[647]	Z value (discrete) of point 648.

Calibrated profile format:

Address	Data type	Field name	Comment
64	uint16_t	Z[0]	Z value (discrete) of point 1.
66	uint16_t	Z[1]	Z value (discrete) of point 2.
68	uint16_t	Z[2]	Z value (discrete) of point 3.
70	uint16_t	Z[3]	Z value (discrete) of point 4.
...	...	...	...
2654	uint16_t	Z[1295]	Z value (discrete) of point 1296.

Extended raw profile format:

Address	Data type	Field name	Comment
64	int16_t	X[0]	X value (discrete) of point 1.
66	uint16_t	Z[0]	Z value (discrete) of point 1.
68	int16_t	X[1]	X value (discrete) of point 2.
70	uint16_t	Z[1]	Z value (discrete) of point 2.
...	...	...	...
2652	int16_t	X[647]	X value (discrete) of point 648.
2654	uint16_t	Z[647]	Z value (discrete) of point 648.

Extended calibrated profile format:

Address	Data type	Field name	Comment
64	int16_t	X[0]	X value (discrete) of point 1.
66	uint16_t	Z[0]	Z value (discrete) of point 1.
68	int16_t	X[1]	X value (discrete) of point 2.
70	uint16_t	Z[1]	Z value (discrete) of point 2.
...	...	...	...
5180	int16_t	X[1295]	X value (discrete) of point 1296.
5182	uint16_t	Z[1295]	Z value (discrete) of point 1296.

Since the type of transmitted data is *integer* (uint16_t, int16_t), the following formulas should be used to convert the position of profile points to *float* format:

- for the **Raw profile** format and the **Extended raw profile** format in subpixel values:

PointZ[N] = Z[N] / Discrete_Value;
PointX[N] = N;

- for the **Calibrated profile** format and the **Extended calibrated profile** format in millimeters:

*PointZ[N] = Z[N] * ZMR / Discrete_Value;*
*PointX[N] = X[N] * XEMR / Discrete_Value.*

Delivery confirmation for the packet with profile ensures delivery of each measurement made. As a confirmation, it is necessary to send the UDP packet to the network address of the scanner, the port is the same as the host port for receiving data (50001 by default). The packet must contain a copy of the first 16 bytes of the profile packet header (fields of the **Message fingerprint** group).

3.2. Message types

Configuration and control of scanner parameters is possible not only using the built-in web page, but also using the service protocol based on the exchange of UDP packets (called messages) between the scanner and the host computer (or other network device).

Message types:

- **Command** - Requires some action from the device to which it is directed (for example, to set / restore parameters), may require confirmation and has a unique identifier.
- **Command confirmation** - Confirms the receipt of the command, contains a unique command identifier, and if the actions do not require lengthy preparation of the result (reading FLASH memory, frame capture, etc.), then the confirmation may contain the requested data.
- **Answer** - Contains the data requested by the command. The answer to the command is sent as soon as the data is ready and contains its own unique identifier; it may require confirmation.

Messages in the form of UDP packets must be sent to the network address of the scanner (192.168.1.30 for factory settings) and the port specified in the **Service port** field in the **Network** tab of the web page (50011 for factory settings). The scanner sends messages with confirmation and answer to the network address and port from which the command was sent.

Message types are defined as follows:

```
/*=============== MESSAGE TYPES ============================================*/
#define              OP_CODE_COMMAND            (uint8_t)0x01
#define              OP_CODE_CONFIRM            (uint8_t)0x02
#define              OP_CODE_ANSWER             (uint8_t)0x03
#define              OP_FLAGS_CONFIRM_BIT       (uint8_t)0x08
#define              OP_FLAGS_FINAL_BIT         (uint8_t)0x04

typedef      uint8_t                            SrvcMsgOperation_Type;
enum
{
      MSG_COMMAND                = (OP_CODE_COMMAND << 4),
      MSG_CONFIRM                = (OP_CODE_CONFIRM << 4),
      MSG_ANSWER                 = (OP_CODE_ANSWER << 4),
      MSG_COMMAND_CNFRM_FINAL    = (OP_CODE_COMMAND << 4) | OP_FLAGS_CONFIRM_BIT |
OP_FLAGS_FINAL_BIT,
      MSG_COMMAND_CNFRM          = (OP_CODE_COMMAND << 4) | OP_FLAGS_CONFIRM_BIT,
      MSG_COMMAND_FINAL          = (OP_CODE_COMMAND << 4) | OP_FLAGS_FINAL_BIT,
      MSG_CONFIRM_FINAL          = (OP_CODE_CONFIRM << 4) | OP_FLAGS_FINAL_BIT,
      MSG_ANSWER_CNFRM           = (OP_CODE_ANSWER << 4) | OP_FLAGS_CONFIRM_BIT,
```

```
        MSG_ANSWER_FINAL            = (OP_CODE_ANSWER << 4) | OP_FLAGS_FINAL_BIT,
        MSG_ANSWER_CNFRM_FINAL      = (OP_CODE_ANSWER << 4) | OP_FLAGS_CONFIRM_BIT |
OP_FLAGS_FINAL_BIT
};
/*=============================================================================*/
```

The OP_CODE_COMMAND, OP_CODE_CONFIRM, OP_CODE_ANSWER, OP_FLAGS_CONFIRM_BIT, OP_FLAGS_FINAL_BIT constants should be used to check the corresponding bits of the **Message type** field and must not be written to this field.

The following constants must be used to set the message type:

Constant	Description
MSG_COMMAND	Command that does not require confirmation and is not the last in the chain of commands.
MSG_CONFIRM	Command confirmation, which is not the last in the chain.
MSG_ANSWER	Answer to the command. It does not require confirmation and is not the last in the chain.
MSG_COMMAND_CNFRM_FINAL	Command that requires delivery confirmation and is the last in the chain of commands.
MSG_COMMAND_CNFRM	Command that requires delivery confirmation and is not the last in the chain of commands.
MSG_COMMAND_FINAL	Command that does not require confirmation and is the last in the chain of commands.
MSG_CONFIRM_FINAL	Command confirmation, which is the last in the chain.
MSG_ANSWER_CNFRM	Answer to the command that requires confirmation and is not the last in the chain.
MSG_ANSWER_FINAL	Answer to the command that does not require confirmation and is the last in the chain of answers.
MSG_ANSWER_CNFRM_FINAL	Answer to the command that requires confirmation and is the last in the chain.

3.3. Message structure

Message structure:

Index	Function	Size, byte	Description
0	Message type	1	Bits 7-4: b0001 - command; b0010 - command confirmation; b0011 - answer. Bit 3 - command confirmation or answer confirmation: b0 - confirmation is not required; b1 - confirmation is required. Bit 2 - last command sign: b0 - command/answer is not the last; b1 - command/answer is the last in the chain. The rest (bits 1 and 0) - reserve.
1	Message parameters	3	Commands: not used. Confirmations, answers: byte [0] contains the result of the command execution (0 - success, otherwise - error).
4	Device identifier	4	Unique device identifier (scanner serial number) to which the message was sent. Can be broadcast: 0xFFFFFFFF. Contains the scanner identifier when the scanner sends a message or confirmation.

Index	Function	Size, byte	Description
8	Message identifier (for confirmation)	2	Counter. Must be unique for each message, the repetition period of 65536 messages is quite sufficient for operation.

10	Command identifier	1	Module identifier:
			{
			MID_SYSTEM = 0x50,
			MID_USER_PARAMS = 0x5E,
			MID_FRAME_CAPTURE = 0x53
			}

11		1	Command code is specific to each module.
12	Data area size	2	Data size in the payload.
14-(N+14)	Command attributes or answer data	N	In the case of sending a command, the field contains the attributes (data that must be applied). In the case of confirmation or answer, the field contains the data requested by the command.

When developing custom software, the message structure can be represented as follows:

```
/*============== MESSAGE STRUCTURE-HEADER ==============================*/
#pragma pack(push, 1)
volatile typedef struct
{
	union{
		struct{
			uint8_t		OpFlags	: 4;
			uint8_t		OpCode	: 4;
		};
		SrvcMsgOperation_Type	Operation;
	};
	uint8_t			MsgParams[3];
	union{
		struct{
			uint32_t	DevID;
			uint16_t	UniqID;
			ModuleID_Type	ModuleID;
			uint8_t		Cmd;
		};
		uint64_t		MsgID;
	};
	uint16_t			PayloadLen;
}SrvcMsgHeader_Type;
/*======================================================================*/

/*============== MESSAGE STRUCTURE =====================================*/
#define		MSG_MAX_PAYLOAD	32768 - sizeof(SrvcMsgHeader_Type)
typedef struct
{
	union{
		struct{
			union{
				SrvcMsgHeader_Type	Header;
				uint8_t			HeaderData[sizeof(SrvcMsgHeader_Type)];
			};
			uint8_t			Payload[MSG_MAX_PAYLOAD];
		};
		uint8_t		RawData[MSG_MAX_PAYLOAD+sizeof(SrvcMsgHeader_Type)];
	};
}SrvcMsg_Type;
/*======================================================================*/
```

The **Command identifier** field contains the module identifier to which the command is addressed and the command code, this allows for the user to functionally divide the commands into groups and have up to 65536 commands in total.

The following modules are available to the user:

- **SYSTEM** - 0x50 identifier, the system module that controls the global parameters of the scanner (receiving and writing all settings in one package, saving parameters, rebooting, etc.).
- **USER_PARAMS** - 0x5E identifier, the user parameters control module that designed to read and write the groups of parameters and individual scanner parameters.
- **FRAME_CAPTURE** - 0x53 identifier, the module provides frame acquisition from the image formed by the CMOS sensor.

3.3.1. SYSTEM module

The **SYSTEM** module contains the following commands:

Command name:	CMD_GET_USER_PARAMS
Command code:	0x02
Description:	Getting the structure with user parameters.
Command attributes:	–
Answer:	Confirmation and data packet: typedef struct { .... }UserParams_TypeDef. **Note:** the UserParams_TypeDef structure is described in p. 3.4.
Command name:	CMD_SET_USER_PARAMS
Command code:	0x03
Description:	Writing user parameters to current parameters without saving to flash memory.
Command attributes:	typedef struct { .... }UserParams_TypeDef.
Answer:	Confirmation packet.
Command name:	CMD_SAVE_PARAMS
Command code:	0x10
Description:	Saving the current scanner configuration to flash memory. All parameters will be saved.
Command attributes:	–
Answer:	Confirmation packet.
Command name:	CMD_SAVE_AS_DEFAULT_PARAMS
Command code:	0x11
Description:	Saving the current scanner configuration to the recovery area in flash memory. All parameters will be saved.
Command attributes:	–
Answer:	Confirmation packet.
Command name:	CMD_RESET
Command code:	0x12
Description:	Rebooting the scanner. Parameters saved in flash memory will be used after rebooting.

Command attributes:	–
Answer:	Confirmation packet.
Command name:	CMD_LOAD_DEFAULT_PARAMS
Command code:	0x13
Description:	Setting all parameters from the recovery area and saving them to flash memory.
Command attributes:	–
Answer:	Confirmation packet.

3.3.2. USER_PARAMS module

The **USER_PARAMS** module contains the following commands:

Command name:	CMD_U_GENERAL_HELLO
Command code:	0x00
Description:	The command to transfer the packet with the main parameters of the device. This command is used when searching for devices on the network.
Command attributes:	–
Answer:	Confirmation and data packet: typedef struct { .... }HelloAnswer_TypeDef. **Note:** the HelloAnswer_TypeDef structure is described in p. 3.5.1.
Command name:	CMD_U_GENERAL_GET
Command code:	0x01
Description:	Request for general user parameters.
Command attributes:	–
Answer:	Confirmation and data packet: typedef struct { … }User_General_TypeDef;
Command name:	CMD_U_GENERAL_SET
Command code:	0x02
Description:	Setting the general user parameters.
Command attributes:	typedef struct { … }User_General_TypeDef;
Answer:	Confirmation packet.
Command name:	CMD_U_SYSMONITOR_GET
Command code:	0x03
Description:	Request for parameters and measurement results by the system monitor.
Command attributes:	–
Answer:	Confirmation and data packet: typedef struct

	{ … }User_SystemMonitor_TypeDef;
Command name:	CMD_U_SYSMONITOR_SET
Command code:	0x04
Description:	Setting the system monitor parameters. Fields containing measurement results are ignored.
Command attributes:	typedef struct { … }User_SystemMonitor_TypeDef;
Answer:	Confirmation packet.
Command name:	CMD_U_COMPATIBILITY_GET
Command code:	0x05
Description:	Request for compatibility mode parameters.
Command attributes:	–
Answer:	Confirmation and data packet: typedef struct { … }User_Compatibility_TypeDef;
Command name:	CMD_U_COMPATIBILITY_SET
Command code:	0x06
Description:	Setting the compatibility mode parameters.
Command attributes:	typedef struct { … }User_Compatibility_TypeDef;
Answer:	Confirmation packet.
Command name:	CMD_U_SENSOR_GET
Command code:	0x07
Description:	Request for CMOS sensor parameters.
Command attributes:	–
Answer:	Confirmation and data packet: typedef struct { … }User_Sensor_TypeDef;
Command name:	CMD_U_SENSOR_SET
Command code:	0x08
Description:	Setting the CMOS sensor parameters.
Command attributes:	typedef struct { … }User_Sensor_TypeDef;

Answer:	Confirmation packet.
Command name:	CMD_U_ROI_GET
Command code:	0x09
Description:	Request for ROI mode parameters.
Command attributes:	–
Answer:	Confirmation and data packet: typedef struct { … }User_ROI_TypeDef;
Command name:	CMD_U_ROI_SET
Command code:	0x0A
Description:	Setting the ROI mode parameters.
Command attributes:	typedef struct { … }User_ROI_TypeDef;
Answer:	Confirmation packet.
Command name:	CMD_U_NETWORK_GET
Command code:	0x0B
Description:	Request for network parameters.
Command attributes:	–
Answer:	Confirmation and data packet: typedef struct { … }User_Network_TypeDef;
Command name:	CMD_U_NETWORK_SET
Command code:	0x0C
Description:	Setting the network parameters.
Command attributes:	typedef struct { … }User_Network_TypeDef;
Answer:	Confirmation packet.
Command name:	CMD_U_STREAMS_GET
Command code:	0x0D
Description:	Request for parameters of data streams.
Command attributes:	–
Answer:	Confirmation and data packet: typedef struct { … }User_Streams_TypeDef;

Command name:	CMD_U_STREAMS_SET
Command code:	0x0E
Description:	Setting parameters of data streams.
Command attributes:	typedef struct { … }User_Streams_TypeDef;
Answer:	Confirmation packet.
Command name:	CMD_U_PROCESSING_GET
Command code:	0x0F
Description:	Request for image processing parameters.
Command attributes:	–
Answer:	Confirmation and data packet: typedef struct { … }User_Processing_TypeDef;
Command name:	CMD_U_PROCESSING_SET
Command code:	0x10
Description:	Setting the image processing parameters.
Command attributes:	typedef struct { … }User_Processing_TypeDef;
Answer:	Confirmation packet.
Command name:	CMD_U_LASER_GET
Command code:	0x11
Description:	Request for laser parameters.
Command attributes:	–
Answer:	Confirmation and data packet: typedef struct { … }User_Laser_TypeDef;
Command name:	CMD_U_LASER_SET
Command code:	0x12
Description:	Setting the parameters of the laser.
Command attributes:	typedef struct { … }User_Laser_TypeDef;
Answer:	Confirmation packet.
Command name:	CMD_U_INPUTS_GET
Command code:	0x13

Description:	Request for parameters of input channels.
Command attributes:	–
Answer:	Confirmation and data packet: typedef struct { … }User_Inputs_TypeDef;
Command name:	CMD_U_INPUTS_SET
Command code:	0x14
Description:	Setting the parameters of input channels.
Command attributes:	typedef struct { … }User_Inputs_TypeDef;
Answer:	Confirmation packet.
Command name:	CMD_U_OUTPUTS_GET
Command code:	0x15
Description:	Request for parameters of output channels.
Command attributes:	–
Answer:	Confirmation and data packet: typedef struct { … }User_Outputs_TypeDef;
Command name:	CMD_U_OUTPUTS_SET
Command code:	0x16
Description:	Setting the parameters of output channels.
Command attributes:	typedef struct { … }User_Outputs_TypeDef;
Answer:	Confirmation packet.

3.3.3. FRAME_CAPTURE module

The **FRAME_CAPTURE** module contains the following commands:

Command name:	CMD_U_FRAME_CAPTURE_GET_FRAME
Command code:	0x10
Description:	Frame request. After receiving the command, the scanner will capture one image frame and transmit it in fragments.
Command attributes:	uint8_t: 0 – do not require confirmation of fragments delivery; other – require confirmation of fragments delivery. Delivery of the last fragment always requires confirmation.
Answer:	Confirmation packet. After capturing a frame, it will be transferred in fragments in the following format: uint32_t: data offset in frame; uint8_t[…]: brightness of points.

3.4. Structure of user parameters

The **UserParams_TypeDef** structure contains all parameters available to the user and has the following fields (data alignment within the structure - bytes):

```
volatile typedef struct __attribute__ ((__packed__))
{
	User_General_TypeDef			General;
	User_SystemMonitor_TypeDef		SystemMonitor;
	User_Compatibility_TypeDef		Compatibility;
	User_Sensor_TypeDef			Sensor;
	User_ROI_TypeDef			ROI;
	User_Network_TypeDef			Network;
	User_Streams_TypeDef			Streams;
	User_Processing_TypeDef			Processing;
	User_Laser_TypeDef			Laser;
	User_Inputs_TypeDef			Inputs;
	User_Outputs_TypeDef			Outputs;
/*-------Reserve-----------------------------------------------*/
	uint8_t					Reserved[283];
/*-------------------------------------------------------------*/
}UserParams_TypeDef;
```

Each field of the **UserParams_TypeDef** structure is an instance of the structure containing specific parameters of one of the subsystems of the scanner.

3.4.1. General parameters

The **User_General_TypeDef** structure:

```
typedef struct __attribute__ ((__packed__))
{
	uint8_t				Name[64];
	uint8_t				Reserved[128];
}User_General_TypeDef;
```

Field name	Description
Name[64]	User-assigned name of the scanner. It can be used to simplify the identification of scanners on the network.
Reserved[128]	Reserved for new parameters.

3.4.2. System monitor parameters

The **User_SystemMonitor_TypeDef** structure contains parameters and results of the system monitor operation, which allow to evaluate the current state of the scanner:

```
typedef struct __attribute__ ((__packed__))
{
	int16_t				FPGA_Temp;
	uint8_t				ParamsChanged;
	uint8_t				Reserved[80];
}User_SystemMonitor_TypeDef;
```

Field name	Description
FPGA_Temp	FPGA temperature, Celsius * 10. For example, if the temperature is 51.2°C, then the value is 512.
ParamsChanged	Parameters have been changed, but not saved: 0 – no changes; 1 – changes, not saved.
Reserved[80]	Reserved for new parameters.

3.4.3. Compatibility mode parameters

The **User_Compatibility_TypeDef** structure contains parameters used in the emulation modes (for example, RF625 emulation):

```
typedef struct __attribute__ ((__packed__))
{
	uint8_t		RF625_Enabled;
	uint16_t	RF625TCPPort;
	uint8_t		Reserved[32];
}User_Compatibility_TypeDef;
```

Field name	Factory value	Description
RF625_Enabled	0	Enable the RF625 emulation mode: 0 – disabled; other – enabled.
RF625TCPPort	620	TCP port number used in the RF625 emulation mode.
Reserved[32]		Reserved for new parameters.

3.4.4. CMOS sensor parameters

The **User_Sensor_TypeDef** structure contains the CMOS sensor parameters:

```
typedef struct __attribute__ ((__packed__))
{
	uint8_t		DoubleSpeedMode;
	uint8_t		Gain_Analog;
	uint8_t		Gain_Digital;
	uint32_t	Exposure;
	uint32_t	MaxExposure;
	uint32_t	FrameRate;
	uint32_t	MaxFrameRate;
	uint8_t		Reserved_0;
	uint8_t		AutoExposure;
	uint8_t		Reserved[62];
}User_Sensor_TypeDef;
```

Field name	Factory value	Description
DoubleSpeedMode	0	Enable the double speed mode. In the double speed mode, the accuracy is reduced from ±0.05% to ±0.085% from the range. 0 – disabled; other – enabled.
Gain_Analog	6	Analog gain of the signal generated by each pixel of the image. Valid values: from 1 to 15.
Gain_Digital	108	Digital gain of the signal generated by each pixel of the image. Valid values: from 96 to 114.
Exposure	300000	Exposure time (signal accumulation time) in nanoseconds, step – 10 ns. The minimum value is 100 ns, the maximum value is defined by the required frequency of profiles.
MaxExposure	1443298	The maximum possible exposure time for the current operation mode of the scanner (nanoseconds).
FrameRate	485	The current rate of the frames generated by the CMOS sensor. It equals to the current frequency of profiles.
MaxFrameRate	485	The maximum possible frame rate for the current operation mode of the scanner (taking into account the double speed mode, ROI parameters, etc.).
Reserved_0		Reserve.

Field name	Factory value	Description
AutoExposure	0	Enable the auto exposure mode: 0 – disabled; other – enabled.
Reserved[62]		Reserved for new parameters.

3.4.5. ROI mode parameters

The **User_ROI_TypeDef** structure contains parameters of the frame area used to obtain the profile. These parameters significantly affect the frequency of profiles.


```
typedef struct __attribute__ ((__packed__))
{
	uint8_t		Enabled;
	uint8_t		Active;
	uint16_t	Size;
	uint8_t		PositionMode;
	uint16_t	FixedPosition;
	uint16_t	AutoPosition;
	uint16_t	RequiredProfileSize;
	uint8_t		Reserved[80];
}User_ROI_TypeDef;
```

Field name	Factory value	Description
Enabled	0	Enable the ROI mode: 0 – disabled; other – enabled.
Active	0	ROI mode activity flag. If the mode is enabled and the profile is detected in the search area, then the value will be «1», otherwise – «0». This flag allows to evaluate the stability of the ROI mode operation with the current settings.
Size	64	The size of the analyzed area in the lines of the CMOS sensor. Valid values: from 24 to 480 (step – 8 lines).
PositionMode	0	This mode controls the position of the analyzed area: 0 – manual control: the position is fixed and set by the operator in the scanner settings; other – automatic control of the position of the analyzed area based on profile analysis.
FixedPosition	300	The position of the analyzed area in the manual mode. Valid values: from 0 to (488-Size).
AutoPosition	100	The current position of the analyzed area in the automatic mode.
RequiredProfileSize	324	The number of points in the profile required for the ROI mode. If the number of points in the profile is less than this number, the scanner will switch to the profile search mode with image analysis from the entire CMOS sensor. Valid values: from 1 to 648 (up to 1296 in the "extended..." modes).
Reserved[80]		Reserved for new parameters.

3.4.6. Network interface parameters

The structure **User_Network_TypeDef** contains the network parameters of the scanner:

```
typedef struct __attribute__ ((__packed__))
{
	uint16_t	Speed;
	uint8_t	Autonegotiation;
	uint8_t	IP[4];
	uint8_t	Mask[4];
	uint8_t	Gateway[4];
	uint8_t	Host_IP[4];
	uint16_t	Host_Data_Port;
	uint16_t	HTTP_Port;
	uint16_t	ServiceCtrlPort;
	uint16_t	EIP_BroadcastRcvPort;
	uint16_t	EIP_ListeningTCPPort;
	uint8_t	Reserved[64];
}User_Network_TypeDef;
```

Field name	Factory value	Description
Speed	1000	Connection speed. In the Autonegotiation mode, this value cannot be changed. If the Autonegotiation mode is disabled, this parameter sets the connection speed: 100 – 100 Mbps; 1000 – 1000 Mbps.
Autonegotiation	1	Enable / disable the Autonegotiation mode (automatic negotiation of the connection speed): 0 – disabled; other – enabled.
IP[4]	192.168.1.30	Network address of the scanner.
Mask[4]	255.255.255.0	Subnet mask.
Gateway[4]	192.168.1.1	Gateway address.
Host_IP[4]	192.168.1.2	Network address of the computer (or other network device) receiving profiles.
Host_Data_Port	50001	Port number of the computer (or other network device), to which the scanner must send UDP packets with profiles.
HTTP_Port	80	Scanner port number for the HTTP connection to access the web page of the scanner.
ServiceCtrlPort	50011	Scanner port number for service protocol.
EIP_BroadcastRcvPort	44818	Service port Ethernet IP.
EIP_ListeningTCPPort	44818	Service port Ethernet IP.
Reserved[64]		Reserved for new parameters.

3.4.7. Data stream parameters

The **User_Streams_TypeDef** structure contains the data stream parameters:

```
typedef struct __attribute__ ((__packed__))
{
	uint8_t	UDP_Profiles_Enabled;
	uint8_t	Profiles_Format;
	uint8_t	Profiles_Confirmation;
	uint8_t	Reserved[32];
}User_Streams_TypeDef;
```

Field name	Factory value	Description
UDP_Profiles_Enabled	1	Enable / disable the stream of UDP packets with profiles: 0 – disabled; other – enabled.
Profiles_Format	1	Set the profile transfer format: 0x10 - «raw profile»; 0x11 - «calibrated profile»; 0x12 - «extended raw profile»; 0x13 - «extended calibrated profile». other – must not be used.
Profiles_Confirmation	0	Enable / disable the requirement to confirm delivery of UDP packets with profiles: 0 – disabled; other – enabled.
Reserved[32]		Reserved for new parameters.

3.4.8. Image processing algorithm parameters

The **User_Processing_TypeDef** structure contains the image processing algorithm parameters:

```
typedef struct __attribute__ ((__packed__))
{
	uint32_t	Threshold;
	uint8_t	Stg1_FilterWidth;
	uint8_t	Stg1_ProcessingMode;
	uint8_t	Stg2_ReduceProfileNoise;
	uint32_t	ProfilesPerSecond;
	uint8_t	Reserved[60];
}User_Processing_TypeDef;
```

Field name	Factory value	Description
Threshold	2000	Filtering threshold for the points that are not bright enough. It is applied after the initial filtering of image points and therefore has wide variation limits: from 0 to 1632000. Points that have the brightness less than the threshold value are excluded from processing.
Stg1_FilterWidth	25	Primary processing filter width. Valid values: from 1 to 25.
Stg1_ProcessingMode	2	Additional math processing mode for results of primary filtering: 0 – without additional processing; 1 – additive processing by the central core; 2 – multiplicative processing by the central core; 3 – mixed processing; other – same as for "2" value.
Stg2_ReduceProfileNoise	0	Filtering mode for points unstable in time. Enabling this mode eliminates unstable profile points. 0 – disabled; other – enabled.
ProfilesPerSecond	–	Actual frequency of profiles per second after processing.
Reserved[60]		Reserved for new parameters.

3.4.9. Laser brightness parameters

The **User_Laser_TypeDef** structure contains parameters that control the laser brightness (currently not used):

```
typedef struct __attribute__ ((__packed__))
{
	uint8_t		Enabled;
	uint8_t		AutoMode;
	uint16_t	Value;
	uint8_t		Reserved[32];
}User_Laser_TypeDef;
```

Field name	Factory value	Description
Enabled	1	Turn on/off the laser. 0 – off; other – on.
AutoMode	0	Enable/disable automatic laser brightness control mode. 0 – disabled; other – enabled.
Value	10	Laser brightness value. It's used when automatic control mode is turned off. Values: from 0 (laser is off) to 100 (maximum brightness).
Reserved[32]		Reserved for new parameters.

3.4.10. Parameters that control the operation of the scanner input channels

The **User_InputsPreset_TypeDef** structure:

```
typedef struct __attribute__ ((__packed__))
{
	uint16_t	ParamsMask;
	uint8_t		In1_Enabled;
	Input1Mode	In1_Mode;
	uint32_t	In1_Delay;
	uint8_t		In1_Divider;
	uint8_t		In2_Enabled;
	Input2Mode	In2_Mode;
	uint8_t		In2_Inverse;
	uint8_t		In3_Enabled;
	Input3Mode	In3_Mode;
	uint8_t		Reserved[12];
}User_InputsPreset_TypeDef;
```

Field name	Factory value	Description
ParamsMask		The mask of parameters used in the preset (the low-order bit - the first field, etc.).
In1_Enabled	0	Enable / disable the use of Input #1: 0 – disabled; other – enabled.
In1_Mode	0	Operation mode of Input #1: 0 – start the frame accumulation by the CMOS sensor on the pulse rise at the input; 1 – start the frame accumulation by the CMOS sensor on the pulse fall at the input; 2 – gating of internal pulse generator by the logical "1" at the input; 3 – gating of internal pulse generator by the logical "0" at the input; other – must not be used.

Field name	Factory value	Description
In1_Delay	100	The delay in the start of frame accumulation by the CMOS sensor relative to the input event in nanoseconds. Step – 10 ns.
In1_Divider	0	Divider value. If the value is "0", then a frame will be formed for each event; if the value is "1", then a frame will be formed through one trigger event; if the value is "2", then a frame will be formed through two trigger events, etc.
In2_Enabled	0	Enable / disable the use of Input #2: 0 – disabled; other – enabled.
In2_Mode	0	Operation mode of Input #2: 0 – determining the moving direction by the level at the input; 1 – determining the moving direction according to the ratio of the phases at Input #1 and Input #2; other – must not be used.
In2_Inverse	0	Inversion mode for Input #2.
In3_Enabled	0	Enable / disable the use of Input #3: 0 – disabled; other – enabled.
In3_Mode	0	Operation mode of Input #3: 0 – reset of the internal profile counter on the pulse rise at the input; 1 – reset of the internal profile counter on the pulse fall at the input; other – must not be used.
Reserved[12]		Reserved for new parameters.

The **User_Inputs_TypeDef** structure:

```
typedef struct __attribute__ ((__packed__))
{
	uint8_t                                 PresetIdx;
	User_InputsPreset_TypeDef               PresetParams[12];
	uint8_t                                 Reserved[32];
}User_Inputs_TypeDef;
```

Field name	Description
PresetIdx	The index of the preset used. It defines the number of the structure from the PresetParams[12] array.
PresetParams[12]	Array of structures with presets. The first 9 structures (with indexes 0...8) are reserved by the manufacturer and should not be changed by the user.
Reserved[32]	Reserved for new parameters.

3.4.11. Parameters that control the operation of the scanner output channels

The **User_Outputs_TypeDef** structure contains parameters that control the operation of the scanner output channels:

```
typedef struct __attribute__ ((__packed__))
{
	uint8_t                         Out1_Enabled;
	OutputMode                      Out1_Mode;
	uint32_t                        Out1_Delay;
	uint32_t                        Out1_PulseWidth;
	uint8_t                         Out1_Inverse;
	uint8_t                         Out2_Enabled;
	OutputMode                      Out2_Mode;
	uint32_t                        Out2_Delay;
	uint32_t                        Out2_PulseWidth;
```

```
    uint8_t                         Out2_Inverse;
    uint8_t                         Reserved[32];
}User_Outputs_TypeDef;
```

Field name	Factory value	Description
Out1_Enabled, Out2_Enabled	0, 0	Enable / disable the use of the outputs: 0 – disabled; other – enabled.
Out1_Mode, Out2_Mode	1, 1	Operation modes of outputs: 0 – pulse generation at the moment of starting the frame accumulation by the CMOS sensor; 1 – repetition of the logic level at Input #1; 2 – pulse generation at the moment of changing the logic level from «0» to «1» at Input #1; 3 – pulse generation at the moment of changing the logic level from «1» to «0» at Input #1; 4 – repetition of the logic level at Input #2; 5 – pulse generation at the moment of changing the logic level from «0» to «1» at Input #2; 6 – pulse generation at the moment of changing the logic level from «1» to «0» at Input #2; 7 – repetition of the logic level at Input #3; 8 – pulse generation at the moment of changing the logic level from «0» to «1» at Input #3; 9 – pulse generation at the moment of changing the logic level from «1» to «0» at Input #3; other – must not be used.
Out1_Delay, Out2_Delay	500, 50	Pulse delay at the output (nanoseconds) relative to the triggering event (taking into account the output operation mode). Step – 10 ns.
Out1_PulseWidth, Out2_PulseWidth	1000, 100	Pulse width at the output (nanoseconds). Step – 10 ns.
Out1_Inverse, Out2_Inverse	0, 0	Enable / disable inversion of the output value. 0 – disabled; other – enabled.
Reserved[32]		Reserved for new parameters.

3.5. Examples of controlling the scanner via the service protocol

3.5.1. Searching for scanners on the network

The search for scanners is performed by sending the UDP packet with the CMD_U_GENERAL_HELLO command to the USER_PARAMS module to the broadcast network address (192.168.1.255 for factory settings) to the port of the service protocol (50011 for factory settings). An example of the packet is shown below:

```
0000  ff ff ff ff ff ff f8 32  e4 bb 8a 91 08 00 45 00
0010  00 2a 7c a7 00 00 80 11  39 ca c0 a8 01 02 c0 a8
0020  01 ff ff 6e c3 5b 00 16  3e a5 1c 00 00 00 ff ff
0030  ff ff 00 00 5e 00 00 00
```

In response, the scanner will send the UDP packet containing the message with the answer to the CMD_U_GENERAL_HELLO command and, as a payload, the HelloAnswer_TypeDef structure:

```
0000  f8 32 e4 bb 8a 91 00 0a  35 3b 56 45 08 00 45 00
0010  02 36 dd ef 00 00 ff 11  58 56 c0 a8 01 1e c0 a8
0020  01 02 c0 01 c3 5b 02 22  c1 04 24 00 00 00 00 3b
0030  56 45 00 00 5e 00 0c 02  52 46 36 32 37 20 32 44
0040  20 4c 61 73 65 72 20 73  63 61 6e 6e 65 72 00 00
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0070  00 00 00 00 00 00 00 00  73 02 00 3b 56 45 04 01
0080  01 01 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00c0  00 00 e8 03 c0 a8 01 1e  ff ff ff 00 c0 a8 01 01
00d0  c0 a8 01 02 51 c3 50 00  5b c3 12 af 12 af 00 00
00e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 05
0100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0120  00 00 01 01 00 00 00 00  00 00 00 00 00 00 00 00
0130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0200  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0210  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0220  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0230  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0240  00 00 00 00
```

The fields of the structure have the following values:

```
typedef struct
{
	struct
	{
		uint8_t			Name[64];
		uint16_t		DeviceID;
		uint32_t		Serial;
		uint32_t		FirmWareVer;
		uint8_t			Reserved[64];
	}General;
	struct
	{
		uint16_t		Speed;
		uint32_t		IP;
		uint32_t		Mask;
		uint32_t		Gateway;
		uint32_t		HostIP;
		uint16_t		HostProfilesPort;
		uint16_t		HTTPPort;
		uint16_t		ServicePort;
		uint16_t		EIPBroadcastRcvPort;
		uint16_t		EIPListeningTCPPort;
		uint8_t			Reserved[32];
	}Network;
	struct
	{
		uint32_t		MaxPayloadSize;
		uint8_t			Reserved[32];
```

```
        }ServiceProtocol;
        struct
        {
                uint8_t                 ProfilesEnabled;
                uint8_t                 ProfilesFormat;
                uint8_t                 Reserved[32];
        }Streams;
        uint8_t                         Reserved[256];
}HelloAnswer_TypeDef;
```

Field name	Factory value	Description
Name	RF627 2D Laser scanner	User-assigned name of the scanner. It can be used to simplify the identification of scanners on the network.
DeviceID	627	Scanner type identifier. Always "627" for scanners RF627.
Serial	----	Scanner serial number. It's assigned by the manufacturer and is unique to each scanner.
FirmWareVer	----	Current firmware version.
Speed	----	Current connection speed: 100 – 100 Mbps; 1000 – 1000 Mbps.
IP	192.168.1.30, uint32_t: 0x1e01a8c0	Scanner network address.
Mask	255.255.255.0, uint32_t: 0x00ffffff	Scanner subnet mask.
Gateway	192.168.1.1, uint32_t: 0x0101a8c0	Gateway address.
HostIP	192.168.1.2, uint32_t: 0x0201a8c0	Network address of the computer (or other network device) receiving UDP packets with profiles.
HostProfilesPort	50001	Port number of the computer (or other network device), to which the scanner must send UDP packets with profiles.
HTTPPort	80	Port number for the HTTP connection.
ServicePort	50011	Port number of the computer (or other network device) used to transfer the messages of the service protocol.
EIPBroadcastRcvPort	44818	Service port Ethernet IP.
EIPListeningTCPPort	44818	Service port Ethernet IP.
MaxPayloadSize	----	Maximum payload size in service protocol messages (bytes).
ProfilesEnabled	1	Permission to send UDP packets with profiles: 0 – not allowed; other – allowed.
ProfilesFormat	1	Profile transfer formats: 0 – «plain measure»; 1 – «plain profile»; 2 – «extended measure»; 3 – «extended profile»; other – must not be used.

3.5.2. Setting the exposure time

To set the exposure time (for example, 50000 ns, i.e. 50 µs), it is necessary to send a message with the CMD_U_SENSOR_SET command to the USER_PARAMS module, and the User_Sensor_TypeDef structure in the payload to the network address of the scanner, to the port of the service protocol:

```
SrvcMsg_Type         Msg;
Msg.Header.Operation = MSG_COMMAND_CNFRM_FINAL;
             Msg.Header.DevID            = Factory.General.Serial;
             Msg.Header.ModuleID         = MID_USER_PARAMS;
             Msg.Header.Cmd                    = CMD_U_SENSOR_SET;
             Msg.Header.PayloadLen       = sizeof(User_Sensor_TypeDef);
             User_Sensor_TypeDef* Data   = (User_Sensor_TypeDef*)Msg.Payload;
             Data->DoubleSpeedMode       = 0;
             Data->Gain_Analog           = 6;
             Data->Gain_Digital          = 108;
             Data->Exposure                    = 50000;
             Data->FrameRate                   = 485;
             Data->AutoExposure          = 0;
             udpServiceSocket.send_data(Msg.RawData,     sizeof(SrvcMsgHeader_Type)     +
Msg.Header.PayloadLen);
```

The corresponding UDP packet:

```
0000  00 0a 35 64 c6 e0 f8 32  e4 bb 8a 91 08 00 45 00
0010  00 7d 42 05 00 00 80 11  74 fa c0 a8 01 02 c0 a8
0020  01 1e c3 5b c3 5b 00 69  b1 e7 1c 00 00 00 e0 c6
0030  64 00 00 00 5e 08 53 00  00 06 6c 50 c3 00 00 00
0040  00 00 00 e5 01 00 00 00  00 00 00 00 00 00 00 00
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0080  00 00 00 00 00 00 00 00  00 00 00
```

The scanner will confirm receipt and execution of this command:

```
0000  f8 32 e4 bb 8a 91 00 0a  35 64 c6 e0 08 00 45 08
0010  00 2a 32 e9 00 00 80 11  84 61 c0 a8 01 1e c0 a8
0020  01 02 c3 5b c3 5b 00 16  2e ca 24 00 00 00 e0 c6
0030  64 00 00 00 5e 08 00 00  00 00 00 00
```

3.5.3. Getting the network settings of the scanner

To get the network parameters of the scanner, you need to send a message with the CMD_U_NETWORK_GET command to the USER_PARAMS module as a UDP packet:

```
0000  00 0a 35 3b 56 45 f8 32  e4 bb 8a 91 08 00 45 00
0010  00 2a 6c 2f 00 00 80 11  4b 23 c0 a8 01 02 c0 a8
0020  01 1e c3 5b c3 5b 00 16  23 0e 1c 00 00 00 00 3b
0030  56 45 02 00 5e 0b 00 00
```

The scanner will confirm receipt of this command by a message with a payload in the form of the User_Network_TypeDef structure:

```
0000  f8 32 e4 bb 8a 91 00 0a  35 3b 56 45 08 00 45 00
0010  00 87 2f f8 00 00 ff 11  07 fd c0 a8 01 1e c0 a8
0020  01 02 c0 01 c3 5b 00 73  d6 42 24 00 00 00 00 3b
0030  56 45 02 00 5e 0b 5d 00  e8 03 01 c0 a8 01 1e ff
0040  ff ff 00 c0 a8 01 01 c0  a8 01 02 51 c3 50 00 5b
0050  c3 12 af 12 af 00 00 00  00 00 00 00 00 00 00 00
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0090  00 00 00 00 00
```

Setting and obtaining other parameters of the scanner is performed in the same way.

4. Technical support

Requests for technical assistance should be addressed at support@riftek.com, or by phone +375-17-2813513.

5. Revisions

Date	Revision	Description
16.11.2018	1.0.0	Starting document.