SOFTWARE TOOLS FOR WORKING WITH LASER SCANNERS RF627 SERIES

Developer guide

22, Logoiisky tract, Minsk
220090, Republic of Belarus
tel/fax: +375 17 357 36 57
info@riftek.com
www.riftek.com
Contents

1. General information................................................................. 3
2. RF627 SDK.................................................................................. 3
   2.1. System requirements............................................................ 3
   2.2. SDK contents...................................................................... 3
   2.3. Class name.......................................................................... 3
   2.4. SDK functions................................................................... 3
       2.4.1. Library initialization.................................................... 3
       2.4.2. Searching for scanners............................................... 3
       2.4.3. Connecting / disconnecting a scanner........................ 3
       2.4.4. Scanner parameters.................................................... 4
           2.4.4.1. Reading / writing configuration parameters............. 4
           2.4.4.2. Saving parameters to flash memory....................... 4
           2.4.4.3. Restoring default parameters............................... 4
           2.4.4.4. Reading system parameters.................................. 4
       2.4.5. Receiving profiles........................................................ 4
       2.4.6. Receiving image............................................................ 4
       2.4.7. Firmware update.......................................................... 4
       2.4.8. Reboot.......................................................................... 5
       2.4.9. Log.............................................................................. 5
       2.4.10. Error messages............................................................ 5
       2.4.11. SDK version................................................................. 5
   2.5. Data structures................................................................. 5
3. Service protocol....................................................................... 10
   3.1. Profile transmission format................................................ 10
   3.2. Message types................................................................. 12
   3.3. Message structure............................................................ 13
       3.3.1. SYSTEM module.......................................................... 15
       3.3.2. USER_PARAMS module............................................... 16
       3.3.3. FRAME_CAPTURE module.......................................... 19
3.4. Structure of user parameters............................................... 19
   3.4.1. General parameters........................................................ 19
   3.4.2. System monitor parameters............................................ 20
   3.4.3. Compatibility mode parameters..................................... 20
   3.4.4. CMOS sensor parameters.............................................. 20
   3.4.5. ROI mode parameters.................................................... 21
   3.4.6. Network interface parameters....................................... 22
   3.4.7. Data stream parameters................................................ 23
   3.4.8. Image processing algorithm parameters.......................... 23
   3.4.9. Laser brightness parameters.......................................... 24
   3.4.10. Parameters that control the operation of the scanner input channels.............................................. 24
   3.4.11. Parameters that control the operation of the scanner output channels........................................... 25
3.5. Examples of controlling the scanner via the service protocol.... 26
   3.5.1. Searching for scanners on the network............................. 26
   3.5.2. Setting the exposure time............................................... 29
   3.5.3. Getting the network settings of the scanner..................... 29
4. BIN format description.......................................................... 30
5. Technical support.................................................................... 30
6. Revisions.............................................................................. 30
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:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>Developer Guide.</td>
</tr>
<tr>
<td>include</td>
<td>Header files.</td>
</tr>
<tr>
<td>x64</td>
<td>LIB and DLL files for x64 and x86.</td>
</tr>
<tr>
<td>x86</td>
<td></td>
</tr>
</tbody>
</table>

2.3. **Class name**
Class name: rf627.

2.4. **SDK functions**

2.4.1. **Library initialization**

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

```c
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**

```c
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**

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

```c
void disconnect();
```
This method closes sockets opened by the connect method.
2.4.4. Scanner parameters

2.4.4.1. Reading / writing configuration parameters

```c
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.
```

```c
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

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

2.4.4.3. Restoring default parameters

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

2.4.4.4. Reading system parameters

```c
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 rf627_user_params structure. Returns false on error.
psysmon: a pointer to sysmon structure to store.
```

2.4.5. Receiving profiles

```c
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

```c
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

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

```c
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

```c
bool reboot();
Reboots the scanner. Returns false on error.
```
2.4.9. Log

```c
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`.

```c
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

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

2.4.11. SDK version

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

2.5. Data structures

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

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

```c
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

```c
#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;
```
Software tools for working with Laser Scanners RF627 Series

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
Software tools for working with Laser Scanners RF627 Series

```c
#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)
```

[Revision 1.0.0] 16.11.2018
Software tools for working with Laser Scanners RF627 Series

```c
{
  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
  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;
}rf627_laser_params_t;
#pragma pack(pop)
```
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:

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Field name</th>
<th>Comment</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>uint8_t</td>
<td>Data_type</td>
<td>The data type marker: 0x10 - «raw profile»; 0x11 - «calibrated profile»; 0x12 - «extended raw profile»; 0x13 - «extended calibrated profile».</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>uint16_t</td>
<td>Device_ID</td>
<td>The device type identifier (627 always).</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>uint32_t</td>
<td>Serial_number</td>
<td>The serial number of the scanner.</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>uint64_t</td>
<td>System_time</td>
<td>The system time in nanoseconds (after turning on the scanner) of the beginning of the exposure of the frame by which the profile is calculated.</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>uint8_t</td>
<td>Protocol_Version_Major</td>
<td>The current version of the protocol.</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>uint8_t</td>
<td>Protocol_Version_Minor</td>
<td>The current version of the protocol revision.</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>uint8_t</td>
<td>Hardware_params_shift</td>
<td>The shift from the header start in bytes to the area of hardware parameters. It is used when resizing the &quot;Application specific data&quot; area.</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>uint8_t</td>
<td>Data_shift</td>
<td>The shift from the header start in bytes to the data area. It is used when resizing the &quot;Hardware parameters&quot; area.</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>uint32_t</td>
<td>Software_packets_counter</td>
<td>The counter of the sent packets with profiles.</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>uint32_t</td>
<td>Measures_counter</td>
<td>The hardware counter of measurements.</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>uint16_t</td>
<td>ZMR</td>
<td>The measuring range along the Z axis, 0.1 * mm.</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>uint16_t</td>
<td>XEMR</td>
<td>The measuring range along the X axis at the end of the range, 0.1 * mm.</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>uint16_t</td>
<td>Discrete_Value</td>
<td>The discrete value by which the measuring range of the scanner is divided.</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>uint16_t</td>
<td>Reserved</td>
<td></td>
<td>Application specific data</td>
</tr>
<tr>
<td>36</td>
<td>uint16_t</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>uint16_t</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Software tools for working with Laser Scanners RF627 Series

### Hardware parameters

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Field name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>uint16_t</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>uint16_t</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>uint16_t</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>uint16_t</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>uint32_t</td>
<td>Exposure_time</td>
<td>The exposure time (ns).</td>
</tr>
<tr>
<td>52</td>
<td>uint32_t</td>
<td>Laser_value</td>
<td>The laser turn-on time during the frame exposure (ns).</td>
</tr>
<tr>
<td>56</td>
<td>uint32_t</td>
<td>Step_counter</td>
<td>The value of the pulse counter at Input #1.</td>
</tr>
<tr>
<td>60</td>
<td>uint8_t</td>
<td>Dir</td>
<td>The direction of movement (determined by Input #2), which is detected at the start of frame accumulation by the CMOS sensor. <strong>Note:</strong> The direction of movement in the profile packet is in the high bit.</td>
</tr>
<tr>
<td>61</td>
<td>uint8_t[3]</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

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

### Raw profile format:

<table>
<thead>
<tr>
<th>Address</th>
<th>Data type</th>
<th>Field name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>uint16_t</td>
<td>Z[0]</td>
<td>Z value (discrete) of point 1.</td>
</tr>
<tr>
<td>66</td>
<td>uint16_t</td>
<td>Z[1]</td>
<td>Z value (discrete) of point 2.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1358</td>
<td>uint16_t</td>
<td>Z[647]</td>
<td>Z value (discrete) of point 648.</td>
</tr>
</tbody>
</table>

### Extended raw profile format:

<table>
<thead>
<tr>
<th>Address</th>
<th>Data type</th>
<th>Field name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>uint16_t</td>
<td>Z[0]</td>
<td>Z value (discrete) of point 1.</td>
</tr>
<tr>
<td>66</td>
<td>uint16_t</td>
<td>Z[1]</td>
<td>Z value (discrete) of point 2.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>2654</td>
<td>uint16_t</td>
<td>Z[1295]</td>
<td>Z value (discrete) of point 1296.</td>
</tr>
</tbody>
</table>

### Calibrated profile format:

<table>
<thead>
<tr>
<th>Address</th>
<th>Data type</th>
<th>Field name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>int16_t</td>
<td>X[0]</td>
<td>X value (discrete) of point 1.</td>
</tr>
<tr>
<td>66</td>
<td>int16_t</td>
<td>Z[0]</td>
<td>Z value (discrete) of point 1.</td>
</tr>
<tr>
<td>68</td>
<td>int16_t</td>
<td>X[1]</td>
<td>X value (discrete) of point 2.</td>
</tr>
<tr>
<td>70</td>
<td>int16_t</td>
<td>Z[1]</td>
<td>Z value (discrete) of point 2.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>2652</td>
<td>int16_t</td>
<td>X[647]</td>
<td>X value (discrete) of point 648.</td>
</tr>
<tr>
<td>2654</td>
<td>uint16_t</td>
<td>Z[647]</td>
<td>Z value (discrete) of point 648.</td>
</tr>
</tbody>
</table>

### Extended calibrated profile format:

<table>
<thead>
<tr>
<th>Address</th>
<th>Data type</th>
<th>Field name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>int16_t</td>
<td>X[0]</td>
<td>X value (discrete) of point 1.</td>
</tr>
<tr>
<td>66</td>
<td>int16_t</td>
<td>Z[0]</td>
<td>Z value (discrete) of point 1.</td>
</tr>
<tr>
<td>68</td>
<td>int16_t</td>
<td>X[1]</td>
<td>X value (discrete) of point 2.</td>
</tr>
</tbody>
</table>
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:
  \[
  \text{Point}Z[N] = \frac{Z[N]}{\text{Discrete	ext{-}Value}}; \\
  \text{Point}X[N] = N;
  \]

- for the **Calibrated profile** format and the **Extended calibrated profile** format in millimeters:
  \[
  \text{Point}Z[N] = \frac{Z[N] \times ZMR}{\text{Discrete	ext{-}Value}}; \\
  \text{Point}X[N] = \frac{X[N] \times XEMR}{\text{Discrete	ext{-}Value}}.
  \]

Delivery confirmation for the profile packet 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:

```c
/*=============== 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_FLAGS = (OP_FLAGS_CONFIRM_BIT << 4) | (OP_FLAGS_FINAL_BIT << 4)
};
```
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:

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

3.3. Message structure

Message structure:

<table>
<thead>
<tr>
<th>Index</th>
<th>Function</th>
<th>Size, byte</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Message type</td>
<td>1</td>
<td>Bits 7-4:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>b0001 - command;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>b0010 - command confirmation;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>b0011 - answer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 3 - command confirmation or answer confirmation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>b0 - confirmation is not required;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>b1 - confirmation is required.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 2 - last command sign:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>b0 - command/answer is not the last;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>b1 - command/answer is the last in the chain.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The rest (bits 1 and 0) - reserve.</td>
</tr>
<tr>
<td>1</td>
<td>Message parameters</td>
<td>3</td>
<td>Commands: not used.</td>
</tr>
<tr>
<td>Index</td>
<td>Function</td>
<td>Size, byte</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>4</td>
<td>Device identifier</td>
<td>4</td>
<td>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.</td>
</tr>
<tr>
<td>8</td>
<td>Message identifier (for confirmation)</td>
<td>2</td>
<td>Counter. Must be unique for each message, the repetition period of 65536 messages is quite sufficient for operation.</td>
</tr>
</tbody>
</table>
| 10    | Command identifier | 1 | Module identifier:  
\{  
MID_SYSTEM = 0x50,  
MID_USER_PARAMS = 0x5E,  
MID_FRAME_CAPTURE = 0x53  
\} |
| 11    | Data area size | 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:

```c
/*=============== 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{
            SrvcMsgHeader_Type Header;
            uint8_t HeaderData[32768 - sizeof(SrvcMsgHeader_Type)];
        }
    }
}SrvMsg_Type;
/*===========================================================================*/
```
Software tools for working with Laser Scanners RF627 Series

```c
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Description</th>
<th>Attributes</th>
<th>Answer</th>
</tr>
</thead>
</table>
| CMD_GET_USER_PARAMS       | 0x02   | Getting the structure with user parameters.      | -                                 | Packet with confirmation and data: typedef struct {
                                                                                     |                                   | Note: the UserParams_TypeDef structure is described in p. 3.4. |
| CMD_SET_USER_PARAMS       | 0x03   | Writing user parameters to current parameters without saving to flash memory. | typedef struct {
                                                                                     |                                 | } UserParams_TypeDef;                  |
| CMD_SAVE_PARAMS           | 0x10   | Saving the current scanner configuration to flash memory. All parameters will be saved. | -                                 | Confirmation packet.               |
| CMD_SAVE_AS_DEFAULT_PARAMS| 0x11   | Saving the current scanner configuration to the recovery area of flash memory. All parameters will be saved. These parameters are used when the main ones are damaged. | -                                 | Confirmation packet.               |
| CMD_RESET                 | 0x12   | Restarting the scanner. When restarting the scanner, the parameters saved to flash memory will be used. | -                                 | Confirmation packet.               |
| CMD_LOAD_DEFAULT_PARAMS   | 0x13   | Setting all default parameters from the recovery area and saving them to flash memory. | -                                 | Confirmation packet.               |
### 3.3.2. USER_PARAMS module

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Description</th>
<th>Attributes</th>
<th>Answer</th>
</tr>
</thead>
</table>
| **CMD_U_GENERAL_HELLO** | 0x00 | The command for transmitting a packet with main parameters of the device. This command is used when searching for devices on the network. | –          | Packet with confirmation and data: typedef struct {
|                  |      |                                                                            |            |}HelloAnswer_TypeDef. |
|                  |      |                                                                            |            | **Note:** The HelloAnswer_TypeDef structure is described in p. 3.5.1. |
| **CMD_U_GENERAL_GET** | 0x01 | Request for general user parameters.                                       | –          | Packet with confirmation and data: typedef struct {
|                  |      |                                                                            |            |}User_General_TypeDef; |
| **CMD_U_GENERAL_SET** | 0x02 | Setting the general user parameters.                                       | Data in the following structure: typedef struct {
|                  |      |                                                                            |            | ... |
|                  |      |                                                                            |            | }User_General_TypeDef; | Confirmation packet. |
| **CMD_U_SYSMONITOR_GET** | 0x03 | Request for parameters and measurement results by the system monitor.      | –          | Packet with confirmation and data: typedef struct {
|                  |      |                                                                            |            | ... |
|                  |      |                                                                            |            | }User_SystemMonitor_TypeDef; |
| **CMD_U_SYSMONITOR_SET** | 0x04 | Setting the system monitor parameters. Fields containing measurement results are ignored. | Data in the following structure: typedef struct {
|                  |      |                                                                            |            | ... |
|                  |      |                                                                            |            | }User_SystemMonitor_TypeDef; | Confirmation packet. |
| **CMD_U_COMPATIBILITY_GET** | 0x05 | Request for compatibility mode parameters.                                | –          | Packet with confirmation and data: typedef struct {
|                  |      |                                                                            |            | ... |
|                  |      |                                                                            |            | }User_Compatibility_TypeDef; |
| **CMD_U_COMPATIBILITY_SET** | 0x06 | Setting the compatibility mode parameters.                                | Data in the following structure: typedef struct {
<p>|                  |      |                                                                            |            | ... |
|                  |      |                                                                            |            | }User_Compatibility_TypeDef; | Confirmation packet. |</p>
<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Description</th>
<th>Attributes</th>
<th>Answer</th>
</tr>
</thead>
</table>
| CMD_U_SENSOR_GET    | 0x07 | Request for CMOS sensor parameters.              | –          | Packet with confirmation and data: typedef struct {
|                     |      |                                                   |            | ... }User_Sensor_TypeDef;                                               |
| CMD_U_SENSOR_SET    | 0x08 | Setting the CMOS sensor parameters.              | Data in the following structure: typedef struct {
|                     |      |                                                   |            | ... }User_Sensor_TypeDef;                                               |
|                     |      |                                                   | –          | Confirmation packet.                                                   |
| CMD_U_ROI_GET       | 0x09 | Request for ROI mode parameters.                 | –          | Packet with confirmation and data: typedef struct {
|                     |      |                                                   |            | ... }User_ROI_TypeDef;                                                  |
| CMD_U_ROI_SET       | 0x0A | Setting the ROI mode parameters.                 | Data in the following structure: typedef struct {
|                     |      |                                                   |            | ... }User_ROI_TypeDef;                                                  |
|                     |      |                                                   | –          | Confirmation packet.                                                   |
| CMD_U_NETWORK_GET   | 0x0B | Request for network parameters.                  | –          | Packet with confirmation and data: typedef struct {
|                     |      |                                                   |            | ... }User_Network_TypeDef;                                              |
| CMD_U_NETWORK_SET   | 0x0C | Setting the network parameters.                  | Data in the following structure: typedef struct {
|                     |      |                                                   |            | ... }User_Network_TypeDef;                                              |
|                     |      |                                                   | –          | Confirmation packet.                                                   |
| CMD_U_STREAMS_GET   | 0x0D | Request for parameters of data streams.          | –          | Packet with confirmation and data: typedef struct {
|                     |      |                                                   |            | ... }User_Streams_TypeDef;                                              |
| CMD_U_STREAMS_SET   | 0x0E | Setting parameters of data streams.              | Data in the following structure: typedef struct {
|                     |      |                                                   |            | ... }User_Streams_TypeDef;                                              |
|                     |      |                                                   | –          | Confirmation packet.                                                   |
| CMD_U_PROCESSING_GET| 0x0F | Request for image processing parameters.         | –          | Packet with confirmation and data: typedef struct {
<p>|                     |      |                                                   |            | ... }                                                                   |</p>
<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Description</th>
<th>Attributes</th>
<th>Answer</th>
</tr>
</thead>
</table>
| CMD_U_PROCESSING_SET     | 0x10 | Setting the image processing parameters. | Data in the following structure: typedef struct {
|                          |      |                                      | …
|                          |      |                                      | } User_Processing_TypeDef;      | Confirmation packet.       |
| CMD_U_LASER_GET          | 0x11 | Request for laser parameters.        | –                                    | Packet with confirmation and data: typedef struct {
|                          |      |                                      | …
|                          |      |                                      | } User_Laser_TypeDef;          |                            |
| CMD_U_LASER_SET          | 0x12 | Setting the laser parameters.        | Data in the following structure: typedef struct {
|                          |      |                                      | …
|                          |      |                                      | } User_Laser_TypeDef;          | Confirmation packet.       |
| CMD_U_INPUTS_GET         | 0x13 | Request for parameters of input channels. | –                                    | Packet with confirmation and data: typedef struct {
|                          |      |                                      | …
|                          |      |                                      | } User_Inputs_TypeDef;         |                            |
| CMD_U_INPUTS_SET         | 0x14 | Setting parameters of input channels. | Data in the following structure: typedef struct {
|                          |      |                                      | …
|                          |      |                                      | } User_InputsTypeDef;          | Confirmation packet.       |
| CMD_U_OUTPUTS_GET        | 0x15 | Request for parameters of output channels. | –                                    | Packet with confirmation and data: typedef struct {
|                          |      |                                      | …
|                          |      |                                      | } User_Outputs_TypeDef;        |                            |
| CMD_U_OUTPUTS_SET        | 0x16 | Setting parameters of output channels. | Data in the following structure: typedef struct {
|                          |      |                                      | …
|                          |      |                                      | } User_Outputs_TypeDef;        | Confirmation packet.       |
3.3.3. FRAME_CAPTURE module

The FRAME_CAPTURE module contains the following commands:

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Description</th>
<th>Attributes</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMD_U_FRAME_CAPTURE_GET_FRAME</td>
<td>0x10</td>
<td>Frame request. After receiving the command, the scanner will capture one image frame and transmit it in fragments.</td>
<td>uint8_t: 0 – do not require confirmation of fragments delivery; other – require confirmation of fragments delivery. Delivery of the last fragment always requires confirmation.</td>
<td>Confirmation packet. After capturing, the frame is transmitted in fragments in the following format: uint32_t: data offset in frame; uint8_t[…]: brightness of points.</td>
</tr>
</tbody>
</table>

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):

```c
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:

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

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name[64]</td>
<td>User-assigned name of the scanner. It can be used to simplify the identification of scanners on the network.</td>
</tr>
<tr>
<td>Reserved[128]</td>
<td>Reserved for new parameters.</td>
</tr>
</tbody>
</table>
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:

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

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

3.4.3. Compatibility mode parameters

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

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

<table>
<thead>
<tr>
<th>Field name</th>
<th>Factory value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF625_Enabled</td>
<td>0</td>
<td>Enable the RF625 emulation mode: 0 – disabled; other – enabled.</td>
</tr>
<tr>
<td>RF625TCPPort</td>
<td>620</td>
<td>TCP port number used in the RF625 emulation mode.</td>
</tr>
</tbody>
</table>

3.4.4. CMOS sensor parameters

The `User_Sensor_TypeDef` structure contains the CMOS sensor parameters:

```c
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;
```
<table>
<thead>
<tr>
<th>Field name</th>
<th>Factory value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoubleSpeedMode</td>
<td>0</td>
<td>Enable the Double Speed mode. In this mode, accuracy decreases from ±0.05% to ±0.085% of the range. 0 – disabled; other – enabled.</td>
</tr>
<tr>
<td>Gain_Analog</td>
<td>6</td>
<td>Analog gain of the signal generated by each pixel of the image. Valid values: from 1 to 15.</td>
</tr>
<tr>
<td>Gain_Digital</td>
<td>108</td>
<td>Digital gain of the signal generated by each pixel of the image. Valid values: from 96 to 114.</td>
</tr>
<tr>
<td>Exposure</td>
<td>300000</td>
<td>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.</td>
</tr>
<tr>
<td>MaxExposure</td>
<td>1443298</td>
<td>The maximum possible exposure time for the current operation mode of the scanner (nanoseconds).</td>
</tr>
<tr>
<td>FrameRate</td>
<td>485</td>
<td>The current rate of the frames generated by the CMOS sensor. It equals to the current frequency of profiles.</td>
</tr>
<tr>
<td>MaxFrameRate</td>
<td>485</td>
<td>The maximum possible frame rate for the current operation mode of the scanner (taking into account the double speed mode, ROI parameters, etc.).</td>
</tr>
<tr>
<td>Reserved_0</td>
<td></td>
<td>Reserve.</td>
</tr>
<tr>
<td>AutoExposure</td>
<td>0</td>
<td>Enable the auto exposure mode: 0 – disabled; other – enabled.</td>
</tr>
</tbody>
</table>

### 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.

```c
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;
```

<table>
<thead>
<tr>
<th>Field name</th>
<th>Factory value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>0</td>
<td>Enable the ROI mode: 0 – disabled; other – enabled.</td>
</tr>
<tr>
<td>Active</td>
<td>0</td>
<td>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.</td>
</tr>
<tr>
<td>Size</td>
<td>64</td>
<td>The size of the analyzed area in the lines of the CMOS sensor. Valid values: from 24 to 480 (step – 8 lines).</td>
</tr>
<tr>
<td>PositionMode</td>
<td>0</td>
<td>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.</td>
</tr>
</tbody>
</table>
### 3.4.6. Network interface parameters

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

```c
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;
```

<table>
<thead>
<tr>
<th>Field name</th>
<th>Factory value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FixedPosition</td>
<td>300</td>
<td>The position of the analyzed area in the manual mode. Valid values: from 0 to (488-Size).</td>
</tr>
<tr>
<td>AutoPosition</td>
<td>100</td>
<td>The current position of the analyzed area in the automatic mode.</td>
</tr>
<tr>
<td>RequiredProfileSize</td>
<td>324</td>
<td>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 &quot;extended...&quot; modes).</td>
</tr>
<tr>
<td>Reserved[80]</td>
<td></td>
<td>Reserved for new parameters.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field name</th>
<th>Factory value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>1000</td>
<td>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.</td>
</tr>
<tr>
<td>Autonegotiation</td>
<td>1</td>
<td>Enable / disable the Autonegotiation mode (automatic negotiation of the connection speed): 0 – disabled; other – enabled.</td>
</tr>
<tr>
<td>Host_IP[4]</td>
<td>192.168.1.2</td>
<td>Network address of the computer (or other network device) receiving profiles.</td>
</tr>
<tr>
<td>Host_Data_Port</td>
<td>50001</td>
<td>Port number of the computer (or other network device), to which the scanner must send UDP packets with profiles.</td>
</tr>
<tr>
<td>HTTP_Port</td>
<td>80</td>
<td>Scanner port number for the HTTP connection to access the web page of the scanner.</td>
</tr>
<tr>
<td>ServiceCtrlPort</td>
<td>50011</td>
<td>Scanner port number for service protocol.</td>
</tr>
<tr>
<td>EIP_BroadcastRcvPort</td>
<td>44818</td>
<td>Service port Ethernet IP.</td>
</tr>
<tr>
<td>EIP_ListeningTCPPort</td>
<td>44818</td>
<td>Service port Ethernet IP.</td>
</tr>
<tr>
<td>Reserved[64]</td>
<td></td>
<td>Reserved for new parameters.</td>
</tr>
</tbody>
</table>
3.4.7. Data stream parameters

The `User_Streams_TypeDef` structure contains the data stream parameters:

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

<table>
<thead>
<tr>
<th>Field name</th>
<th>Factory value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP_Profiles_Enabled</td>
<td>1</td>
<td>Enable / disable the stream of UDP packets with profiles: 0 – disabled; other – enabled.</td>
</tr>
<tr>
<td>Profiles_Format</td>
<td>1</td>
<td>Set the profile transfer format: 0x10 - «raw profile»; 0x11 - «calibrated profile»; 0x12 - «extended raw profile»; 0x13 - «extended calibrated profile»; other – must not be used.</td>
</tr>
<tr>
<td>Profiles_Confirmation</td>
<td>0</td>
<td>Enable / disable the requirement to confirm delivery of UDP packets with profiles: 0 – disabled; other – enabled.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field name</th>
<th>Factory value</th>
<th>Description</th>
</tr>
</thead>
</table>

3.4.8. Image processing algorithm parameters

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

```c
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;
```

<table>
<thead>
<tr>
<th>Field name</th>
<th>Factory value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold</td>
<td>2000</td>
<td>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.</td>
</tr>
<tr>
<td>Stg1_FilterWidth</td>
<td>25</td>
<td>Primary processing filter width. Valid values: from 1 to 25.</td>
</tr>
<tr>
<td>Stg1_ProcessingMode</td>
<td>2</td>
<td>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 &quot;2&quot; value.</td>
</tr>
<tr>
<td>Stg2_ReduceProfileNoise</td>
<td>0</td>
<td>Filtering mode for points unstable in time. Enabling this mode eliminates unstable profile points.</td>
</tr>
</tbody>
</table>
3.4.9. Laser brightness parameters

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

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

<table>
<thead>
<tr>
<th>Field name</th>
<th>Factory value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>1</td>
<td>Turn on/off the laser. 0 – off; other – on.</td>
</tr>
<tr>
<td>AutoMode</td>
<td>0</td>
<td>Enable/disable automatic laser brightness control mode. 0 – disabled; other – enabled.</td>
</tr>
<tr>
<td>Value</td>
<td>10</td>
<td>Laser brightness value. It's used when automatic control mode is turned off. Values: from 0 (laser is off) to 100 (maximum brightness).</td>
</tr>
</tbody>
</table>

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

The User_InputsPreset_TypeDef structure:

```c
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;
```

<table>
<thead>
<tr>
<th>Field name</th>
<th>Factory value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParamsMask</td>
<td></td>
<td>The mask of parameters used in the preset (the low-order bit - the first field, etc.).</td>
</tr>
<tr>
<td>In1_Enabled</td>
<td>0</td>
<td>Enable / disable the use of Input #1: 0 – disabled; other – enabled.</td>
</tr>
<tr>
<td>In1_Mode</td>
<td>0</td>
<td>Operation mode of Input #1: 0 – start the frame accumulation by the CMOS sensor on the pulse rise at the input;</td>
</tr>
</tbody>
</table>
### Field name | Factory value | Description
--- | --- | ---
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.

<table>
<thead>
<tr>
<th>In1_Delay</th>
<th>100</th>
<th>The delay in the start of frame accumulation by the CMOS sensor relative to the input event in nanoseconds. Step – 10 ns.</th>
</tr>
</thead>
<tbody>
<tr>
<td>In1_Divider</td>
<td>0</td>
<td>Divider value. If the value is &quot;0&quot;, then a frame will be formed for each event; if the value is &quot;1&quot;, then a frame will be formed through one trigger event; if the value is &quot;2&quot;, then a frame will be formed through two trigger events, etc.</td>
</tr>
</tbody>
</table>
| 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. |

### The `User_Inputs_TypeDef` structure:

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

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PresetIdx</td>
<td>The index of the preset used. It defines the number of the structure from the PresetParams[12] array.</td>
</tr>
<tr>
<td>PresetParams[12]</td>
<td>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.</td>
</tr>
</tbody>
</table>

#### 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:

```c
typedef struct __attribute__ ((__packed__)) {
    uint8_t Out1_Enabled;
    OutputMode Out1_Mode;
    uint32_t Out1_Delay;
    uint32_t Out1_PulseWidth;
}User_Outputs_TypeDef;
```
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;

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

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:
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:

```c
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;
}
```

The fields of the structure have the following values:

- Name: 64-byte string
- DeviceID: 16-bit integer
- Serial: 32-bit integer
- FirmWareVer: 32-bit integer
- Reserved[64]: 64-byte string
- Speed: 16-bit integer
- IP: 32-bit integer
- Mask: 32-bit integer
- Gateway: 32-bit integer
- HostIP: 32-bit integer
- HostProfilesPort: 16-bit integer
- HTTPPort: 16-bit integer
- ServicePort: 16-bit integer
- EIPBroadcastRcvPort: 16-bit integer
- EIPListeningTCPPort: 16-bit integer
- Reserved[32]: 32-byte string
/**
 * Software tools for working with Laser Scanners RF627 Series
 */

#include <stdint.h>

enum {
    MaxPayloadSize,
    Reserved[32],
} ServiceProtocol;

struct {
    uint8_t ProfilesEnabled;
    uint8_t ProfilesFormat;
    uint8_t Reserved[32];
} Streams;

uint8_t Reserved[256];

} HelloAnswer_TypeDef;

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

```c
SrvcMsg_Type  Msg;
Msg.Header.Operation = MSG_COMMAND_CNFRM_FINAL;
Msg.Header.ModuleID = MID_USER_PARAMS;
Msg.Header.Cmd = CMD_U_SENSOR_SET;
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 f8 32 e4 b0 8a 91 00 00 45 00
0010 00 7d 42 05 00 00 00 11 74 f3 c0 01 02 00 08
0020 01 1e c3 5b c3 5b 00 69 b1 e7 1c 00 00 00 06
0030 00 00 00 05 00 00 00 00 00 00 00 00 00 00
0040 00 00 00 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
0060 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
```

The scanner will confirm receipt and execution of this command:

```
0000 f8 32 e4 b0 8a 91 00 00 a3 35 64 c6 e9 08 08 08 08
0010 00 2a 32 c9 00 00 00 11 04 61 c9 a8 01 1c 00 08
0020 01 02 c3 5b c3 5b 00 16 2e ca 24 00 00 00 00 00
0030 54 00 00 00 00 00 00 00 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 64 45 f8 32 e4 b0 8a 91 00 00 45 00
0010 00 2a 6c 2f 00 00 00 11 6b c0 a8 01 02 00 00
0020 01 1e c3 5b c3 5b 00 16 23 0e 1c 00 00 00 00 00
0030 46 45 02 00 00 00 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:
Setting and obtaining other parameters of the scanner is performed in the same way.

4. **BIN format description**

Using the web interface, you can save a file with accumulated profiles in **BIN** format (see section 16.2.3. of the User’s Manual for Laser Scanners RF627 Series).

A description of the **BIN** format:

<table>
<thead>
<tr>
<th>Points count</th>
<th>Point0 X (float)</th>
<th>Point0 Z (float)</th>
<th>Point1 X (float)</th>
<th>Point1 Z (float)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x85</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Profile0 (cont.)</th>
<th>Profile1</th>
</tr>
</thead>
</table>
| XXX | XXX | XXX | XXX | XXX | XXX | XXX | XXX | 0 | 1 | 2 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | ...
| Last point X (float) | Last point Z (float) | 0x85 | Points count | --- | --- | Point0 X (float) | Point0 Z (float) |

<table>
<thead>
<tr>
<th>Profile1 (cont.)</th>
<th>ProfileN</th>
</tr>
</thead>
</table>
| Point1 X (float) | Point1 Z (float) | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | XXX | XXX | XXX | XXX | XXX | XXX | XXX | XXX | ...
| --- | Last point X (float) | Last point Z (float) | --- | --- |

To view the accumulated profiles in **BIN** format, use the **RFProfileView** program ([https://riftek.com/downloads/RFProfileView.zip](https://riftek.com/downloads/RFProfileView.zip)).

5. **Technical support**

Requests for technical assistance should be addressed at support@riftek.com.

6. **Revisions**

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16.11.2018</td>
<td>1.0.0</td>
<td>Starting document.</td>
</tr>
</tbody>
</table>