RF62X-SDK Documentation

Release 1.3.7

Vladislav Kuzmin

Aug 31, 2020

PROGRAMMER'S MANUAL


RF62X SDK
A software development kit for
work with scanners series RF62X

CHAPTER

ONE

INTRODUCTION

RF627X-SDK - a Software Development Kit that allows specialists to create their own software for working with laser scanners RF62X (RF627Old, RF627Smart) series manufactured by RIFTEK LLC.

1.1 Overview

This guide was created to help developers and contains a detailed description of the RF62X-SDK library.

1.1.1 General description

RF62X-SDK - a Software Development Kit that allows specialists to create their own software for working with laser scanners RF62X (RF627Old, RF627Smart) series manufactured by RIFTEK LLC.

1.1.2 Library architecture

The RF62X-SDK consists of two parts:

- **RF62X-CORE** - the main library (“Core”) with a basic set of functions and types for working with laser scanners of the RF62X series. The library is written in the C programming language in accordance with the C99 standard (ISO / IEC 9899: 1999) and is cross-platform. To use this library, it is necessary to implement platform-dependent functions (working with memory, working with the network, input/output functions).
- **RF62X-WRAPPERS** - «wrapper»-libraries, in which platform-dependent «Core» functions for a specific platform are already implemented. The use of wrapper libraries simplifies the process of developing applications in the following programming languages: C++,C#, Python, LabVew, MatLab.

1.1.3 Ways of working

Developers who want to use **ready-made RF62X-SDK libraries** when creating their own applications for working with laser scanners of the RF62X series can **download** the latest libraries (download the RF62X-SDK libraries for *C++*, *C#*), as well as see **examples of their use** (see examples for *Examples for C++*, *Examples for C#*).

Developers who prefer **to compile RF62X-SDK libraries** from sources, the manual contains instructions for downloading sources (see *Download project*) and installing the necessary software (see *Installation and setup*).

1.1.4 Main functionality

- Search for RF62X-old series scanners.
- Search for RF62X-smart series scanners.
- Getting profiles.
- Getting/setting scanner parameters.
- Supported protocols for information exchange with scanners:
  - RF627-Protocol
  - RF62X-SmartProtocol
  - Ethernet/IP
  - ModbusTCP

1.1.5 What's new

- Added the ability to simultaneously work with multiple scanners on the network.

1.2 Getting Started

The RF62X-SDK provides the user with a simple interface when developing software for the RF62X Series Scanners.

Developers who want to use **ready-made RF62X-SDK libraries** when creating their own applications for working with laser scanners of the RF62X series can **download** the latest libraries (download the RF62X-SDK libraries for *C++*, *C#*), as well as see **examples of their use** (see examples for *Examples for C++*, *Examples for C#*) .

Developers who prefer **to compile RF62X-SDK libraries** from sources, the manual contains instructions for downloading sources (see *Download project*) and installing the necessary software (see *Installation and setup*).

1.2.1 Target Platforms and Compatibility

Programming languages

The main software library RF62X-CORE («Core») is written in C language of the C99 standard (ISO / IEC 9899: 1999) without the use of third-party software modules and functions dependent on the operating system or processor.

Target platforms

Compatibility is achieved with any operating systems of the Windows, Linux and FreeBSD family that support C language compilers of the C99 standard (ISO / IEC 9899: 1999). The library is compiled from source codes and can be used with any type of processor (x86, ARM, RISC-V, etc.).

Supported Compilers

- GCC 5.x or newer on Linux
- XCode 8.0 or newer on OS X
- Visual Studio 2017 or newer on Windows

Resources

This project uses git for source code management and GitLab for source code hosting.

- Source code: www.gitlab.com/riftek_llc/software/sdk/scanners/RF62X-SDK
- Documentation: RF62X-SDK.ru.pdf, RF62X-SDK.en.pdf
- Website: www.riftek.com

1.2.2 Installation and setup

Software installation

There are several options for building the RF62X-SDK library. All options are supported and should work equally correctly for:

- IDE Visual Studio 2019
- IDE Qt Creator 4.11.0
- CMake 3.16.1

Note: If you are familiar with CMake, then you can also independently create projects for Code-Blocks, Eclipse, KDevelop3, and Xcode.

If you have difficulty installing or configuring development environments, the following are more detailed instructions:

- IDE Visual Studio 2019 (additional information is available on the official website docs.microsoft.com)
- IDE Qt Creator (additional information is available on the official website qt.io)
- CMake (additional information is available on the official website cmake.org)

1.2.3 Download project

Git-client

For developers who want to download the library from source using the Git-client, follow these instructions:

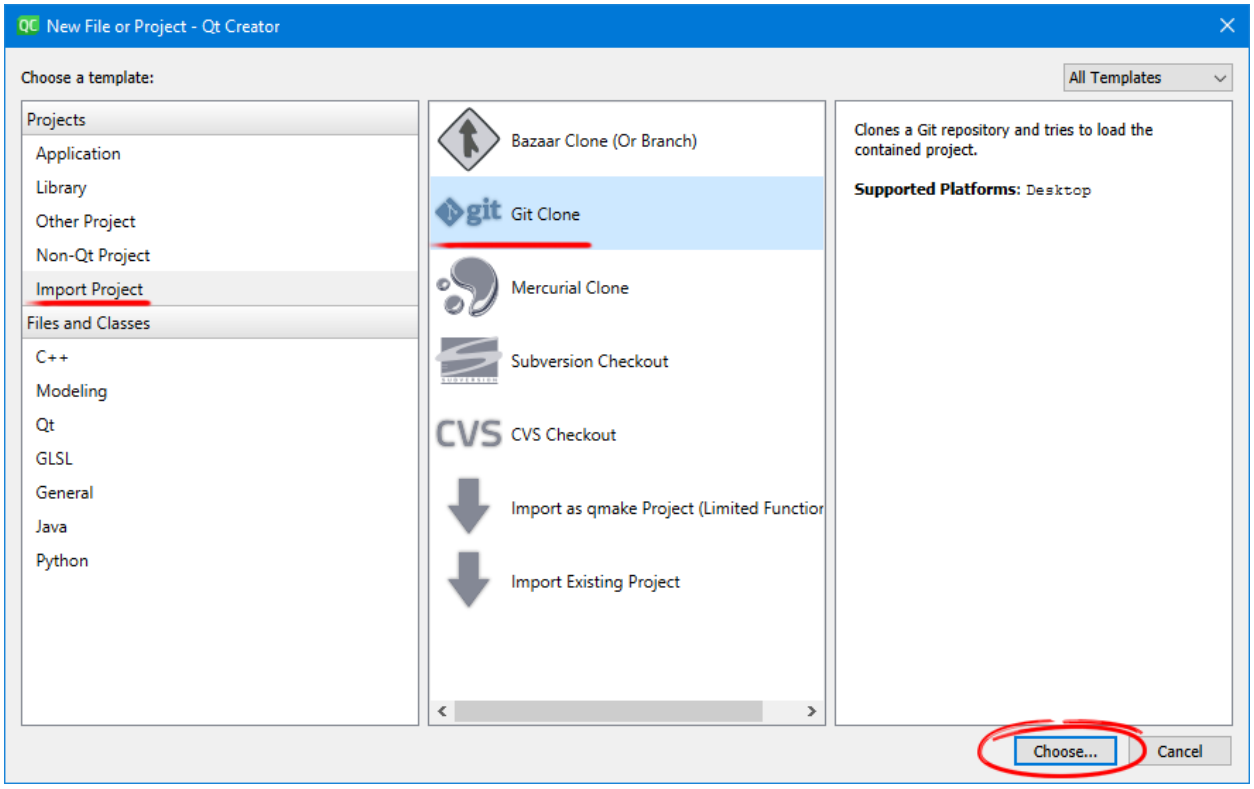
- **Install the git-client on your local computer (if not already installed)**
  - On Linux, use the terminal command: `sudo apt install git`
  - On MacOS, use the terminal command: `brew install git`
  - For other platforms see git installation documentation.
- **Open a command prompt/terminal on your computer**
  - On Linux, click on the launchpad and look for «terminal» - *terminal*
  - In OS X, press command-space and find «terminal» - *terminal*
  - On Windows, click the Start menu and find the «command line» - *cmd*.
- **Clone the repository using the following commands:**

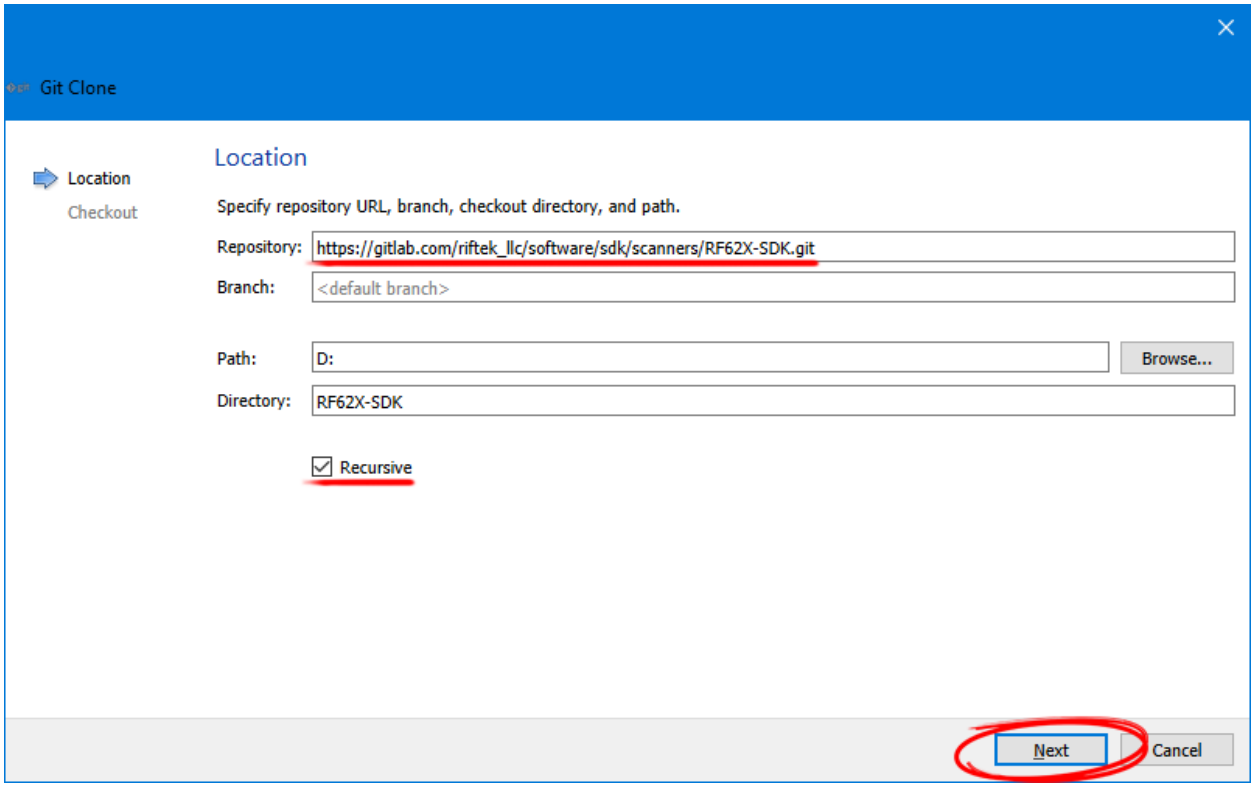
```
git clone https://gitlab.com/riftek_llc/software/sdk/scanners/RF62X-SDK.
↪git
cd RF62X-SDK
git submodule update --init --recursive
```

Git in Qt Creator

For developers who want to download and compile a library from source using the Git in the Qt Creator IDE, follow these instructions:

1. Click **File->New File or Project**
2. Select **Import Project->Git Clone** option as shown below.
3. Enter the SDK url `https://gitlab.com/riftek_llc/software/sdk/scanners/RF62X-SDK.git`, select the **«Recursive»** option, and then click **Next**.
4. After downloading, open the CMakeLists.txt file of the project you need through **File> Open File or Project**, select the compiler (MinGW, MSVC2017, Clang) and click **Configure Project**
5. Run the build project

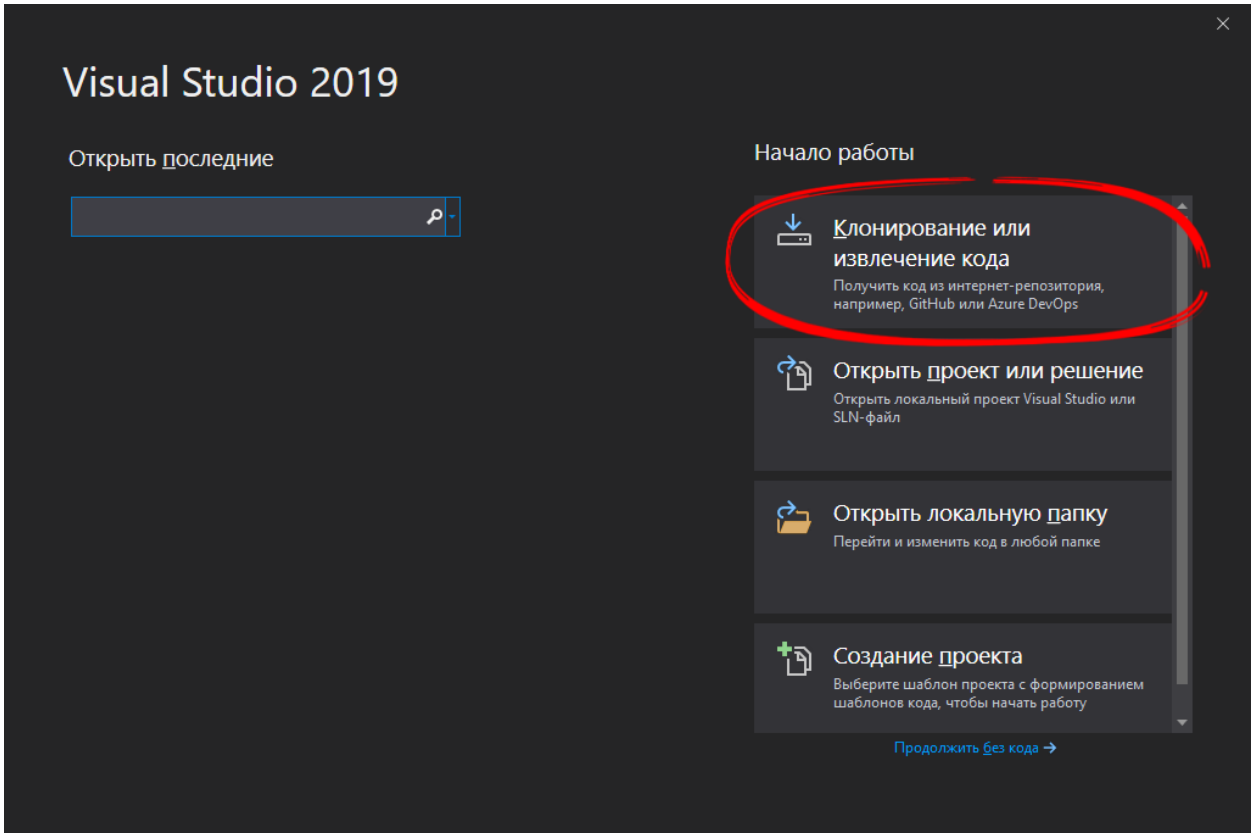

New File or Project - Qt Creator
Choose a template:
All Templates
Projects
Application
Library
Other Project
Non-Qt Project
Import Project
Files and Classes
C++
Modeling
Qt
GLSL
General
Java
Python
Bazaar Clone (Or Branch)
Git Clone
Mercurial Clone
Subversion Checkout
CVS Checkout
Import as qmake Project (Limited Functior
Import Existing Project
Clones a Git repository and tries to load the contained project.
Supported Platforms: Desktop
Choose...
Cancel


Git Clone
Location
Checkout
Location
Specify repository URL, branch, checkout directory, and path.
Repository: https://gitlab.com/riftek_llc/software/sdk/scanners/RF62X-SDK.git
Branch: <default branch>
Path: D:
Browse...
Directory: RF62X-SDK
Recursive
Next
Cancel

Git in Visual Studio

For developers who want to download and build a library from source using the Git in the Visual Studio IDE, you should follow these instructions:

1. Open Visual Studio 2019.
2. In the start window, select **Clone or Extract Code**.



3. Enter the SDK url `https://gitlab.com/riftek_llc/software/sdk/scanners/RF62X-SDK.git`, select or enter the storage location, and then click **Clone**.
4. After that, Visual Studio will load the project from the remote repository and open it.
5. Select one of the projects you need and run its assembly.

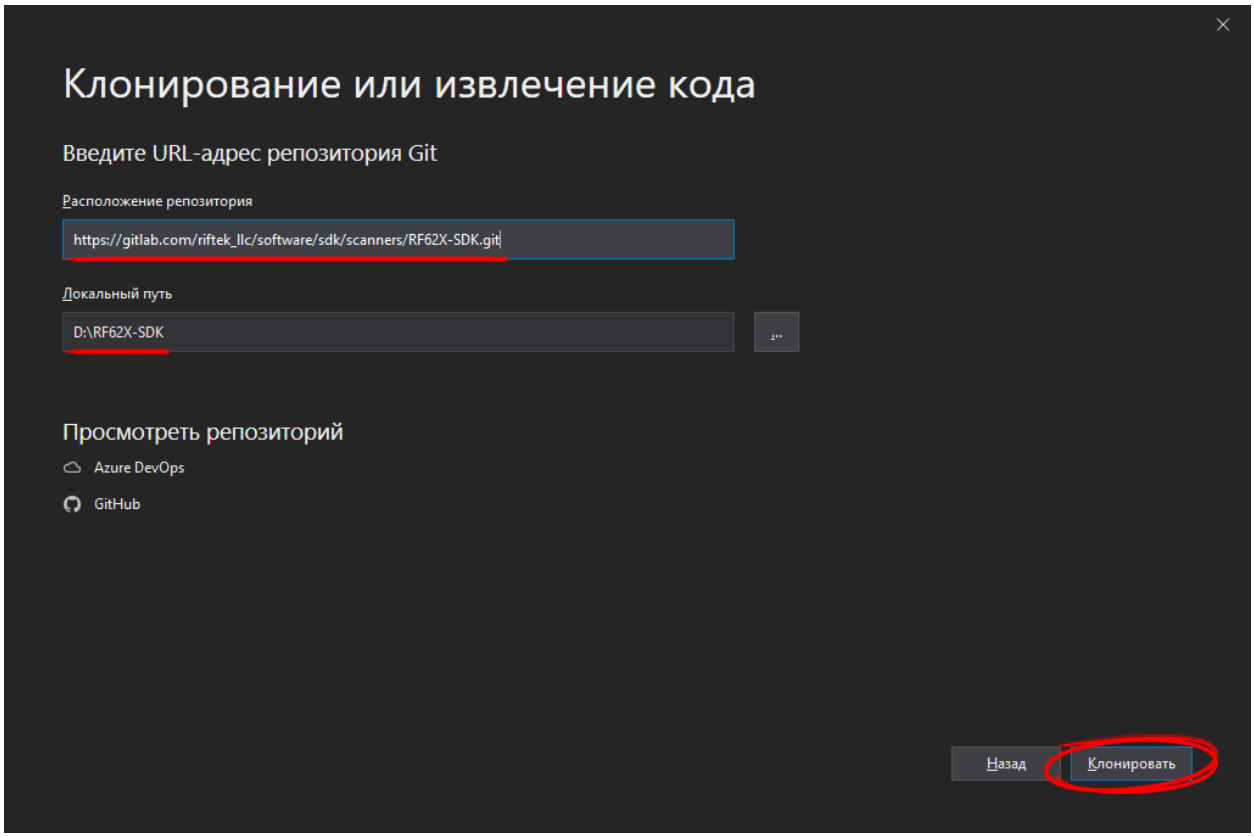
Note: To build libraries **in C++**, as well as compile applications with examples of their use, the C++ CMake tools для Windows must be installed in Visual Studio

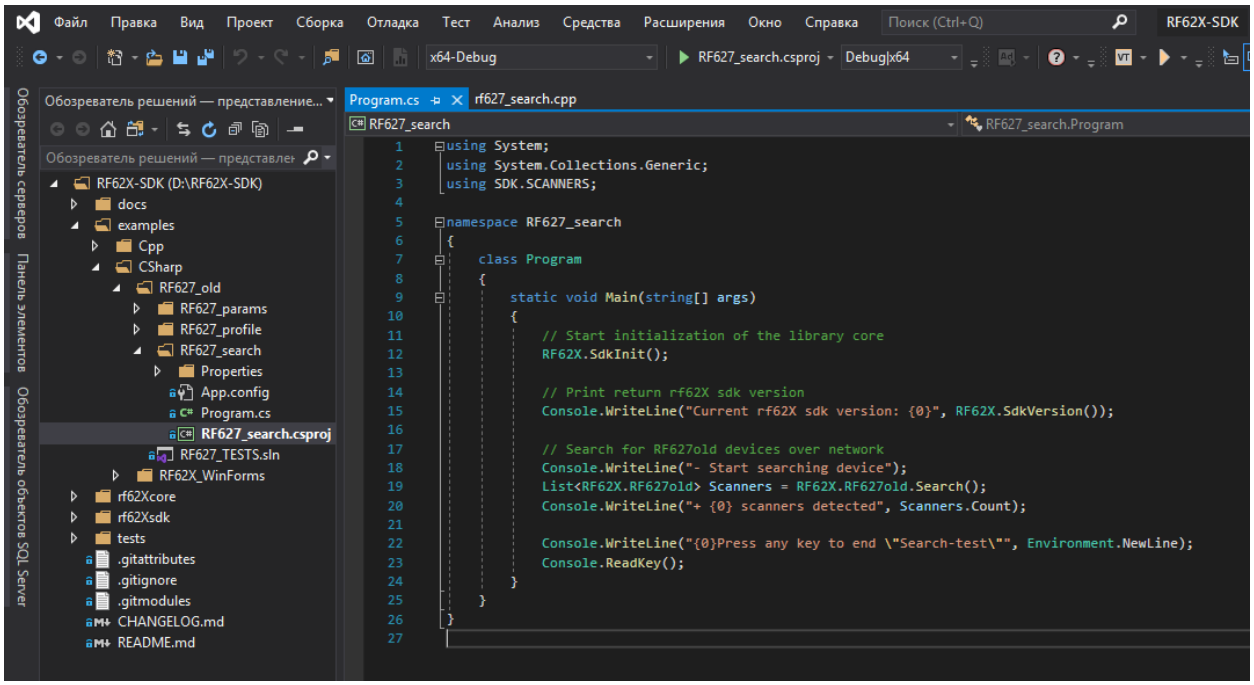
---

1.3 Compilation from source

As it mentioned earlier, the RF62X-SDK consists of two parts:

- **RF62X-CORE** - the main library («Core») with a basic set of functions and types for working with laser scanners of the RF62X series. The library is written in the C programming language in accordance with the C99 standard (ISO / IEC 9899: 1999) and is cross-platform. To use this library, it is necessary to implement platform-dependent functions (working with memory, working with the network, input/output functions).


Клонирование или извлечение кода
Введите URL-адрес репозитория Git
Расположение репозитория
https://gitlab.com/riftek_llc/software/sdk/scanners/RF62X-SDK.git
Локальный путь
D:\RF62X-SDK
Просмотреть репозиторий
Azure DevOps
GitHub
Назад
Клонировать


Файл Правка Вид Проект Сборка Отладка Тест Анализ Средства Расширения Окно Справка
Поиск (Ctrl+Q)
RF62X-SDK
x64-Debug
RF627_search.csproj
Debug|x64
Обозреватель решений — представление...
Program.cs
rf627_search.cpp
RF627_search
RF627_search.Program
RF62X-SDK (D:\RF62X-SDK)
docs
examples
Cpp
CSharp
RF627_old
RF627_params
RF627_profile
RF627_search
Properties
App.config
Program.cs
RF627_search.csproj
RF627_TESTS.sln
RF62X_WinForms
rf62Xcore
rf62Xsdk
tests
.gitattributes
.gitignore
.gitmodules
CHANGELOG.md
README.md
using System;
using System.Collections.Generic;
using SDK.SCANNERS;
namespace RF627_search
{
class Program
{
static void Main(string[] args)
{
// Start initialization of the library core
RF62X.SdkInit();
// Print return rf62X sdk version
Console.WriteLine("Current rf62X sdk version: {0}", RF62X.SdkVersion());
// Search for RF627old devices over network
Console.WriteLine("- Start searching device");
List<RF62X.RF627old> Scanners = RF62X.RF627old.Search();
Console.WriteLine("+ {0} scanners detected", Scanners.Count);
Console.WriteLine("{0}Press any key to end \"Search-test\"", Environment.NewLine);
Console.ReadKey();
}
}
}

- **RF62X-WRAPPERS** - Wrapper Libraries in which platform-dependent functions of the «Core» for a specific platform are already implemented. The use of wrapper libraries simplifies the process of developing applications in the following programming languages: C, C++, C#, PYTHON, LabVew, MatLab.

1.3.1 Compiling the «Core» in C

RF62X-CORE - the main library («Core») with a basic set of functions and types for working with laser scanners of the RF62X series. The library is written in the C programming language in accordance with the C99 standard (ISO / IEC 9899: 1999) and is cross-platform. To use this library, it is necessary to implement platform-dependent functions (working with memory, working with the network, input/output functions).

Table 1: Latest releases:

Compiler	64bit	Includes
MinGW 7.3.0	rf62Xcore.dll rf62Xcore.a	include.zip
MSVC2017	rf62Xcore.dll rf62Xcore.lib	include.zip
Clang 9.1.0	rf62Xcore.dll rf62Xcore.lib	include.zip

How to compile

RF62X-CORE can be compiled using the console or development environment (e.g. Visual Studio, Qt Creator)

First, you must download the project (if you haven't done this before)

Note: for more information on project loading steps see *Download project*

CMake

From the project folder, to build RF62X-CORE, enter the following command into the console (terminal):

```
cd rf62Xcore
mkdir build
cd build
cmake ..
cmake --build .
```

Qt Creator

To build an RF62X-CORE using the Qt Creator IDE:

- Download the CMakeLists.txt file from the **rf62Xcore** folder through **File>Open File or Project** (select the CMakeLists.txt file)
- Select the compiler (MinGW, MSVC2017, Clang) and click **Configure Project**
- Open **Build Settings** and check **install** for **Build Steps**
- Compile the project

Visual Studio

From the project folder, to build RF62X-CORE, enter the following command into the console (terminal):

```
cd rf62Xcore
mkdir build
cd build
cmake ..
```

- Open the resulting rf62Xcore.sln solution in Visual Studio
- Compile the project

How to use

If you want to use the RF62X-CORE library instead of the provided Wrapper Libraries, the developer needs to independently implement the platform-dependent part of the «Core».

Platform Dependent features overview

In the «Core» RF62X-CORE platform-dependent functions (working with memory, working with the network, input/output functions) are presented in the form of pointers to functions.

Pointers to platform-specific functions are declared in the files, `memory_platform.h`, `network_platform.h` and `iostream_platform.h`

memory_platform.h

typedef void *(***calloc_t**)(rfSize num, rfSize size)
Allocates an array in memory with elements initialized to 0.

Return

- On success: returns a pointer to the allocated space.
- On error: NULL

Parameters

- `num`: - number of elements to allocate.

- `size`: - size of each element.

typedef void *(***malloc_t**)(rfSize size)

malloc_t - ptr to function whish allocates memory block Allocates a block of size bytes of memory, returning a pointer to the beginning of the block.

Return On success, a pointer to the memory block allocated by the function. If the function failed to allocate the requested block of memory, a null pointer is returned.

Parameters

- `size`: - Size of the memory block, in bytes.

typedef void *(***realloc_t**)(void *ptr, rfSize newsize)

realloc_t - ptr to function whish reallocates memory block Changes the size of the memory block pointed to by ptr. The function may move the memory block to a new location (whose address is returned by the function).

Return A pointer to the reallocated memory block, which may be either the same as ptr or a new location.

Parameters

- `ptr`: - Pointer to a memory block previously allocated.
- `newsize`: - New size for the memory block, in bytes.

typedef void (***free_t**)(void *data)

Deallocates or frees a memory block.

Parameters

- `data`: - Previously allocated memory block to be freed.

typedef void *(***memset_t**)(void *memptr, rfInt val, rfSize num)

memset_t - ptr to function whish fills block of memory Sets the first num bytes of the block of memory pointed by ptr to the specified value (interpreted as an unsigned rfChar).

Return ptr is returned.

Parameters

- `memptr`: - Pointer to the block of memory to fill.
- `val`: - Value to be set.
- `num`: - Number of bytes to be set to the value. rfSize is an unsigned rfIntegral type.

typedef void *(***memcpy_t**)(void *destination, **const** void *source, rfSize num)

memcpy_t - ptr to function whish copies block of memory Copies the values of num bytes from the location pointed to by source directly to the memory block pointed to by destination.

Return destination is returned.

Parameters

- `destination`: - Pointer to the destination array where the content is to be copied, type-casted to a pointer of type void*.

- `source`: - Pointer to the source of data to be copied, type-casted to a pointer of type const void*.
- `num`: - Number of bytes to copy. rfSize is an unsigned rfIntegral type.

typedef rfInt (***memcmp_t**)(**const** void *ptr1, **const** void *ptr2, rfSize num)

memcmp_t - ptr to function whish compare two blocks of memory Compares the first num bytes of the block of memory pointed by ptr1 to the first num bytes pointed by ptr2, returning zero if they all match or a value different from zero representing which is greater if they do not.

Return 0 - if the contents of both memory blocks are equal, <0 - if the first byte that does not match in both memory blocks has a lower value in ptr1 than in ptr2. >0 - if the first byte that does not match in both memory blocks has a greater value in ptr1 than in ptr2.

Parameters

- `ptr1`: - Pointer to block of memory.
- `ptr2`: - Pointer to block of memory.
- `num`: - Number of bytes to compare.

network_platform.h

typedef rfUint32 (***hton_long_t**)(rfUint32 hostlong)

The modbusHtoN_long_t function converts a u_long from host to TCP/IP network byte order (which is big-endian).

Return : The modbusHtoN_long_t function returns the value in TCP/IP's network byte order.

Parameters

- `hostlong`: - A 32-bit number in host byte order.

typedef rfUint32 (***ntoh_long_t**)(rfUint32 netlong)

The modbusHtoN_long_t function converts a u_long from TCP/IP network order to host byte order (which is little-endian on rfIntel processors).

Return : The modbusNtoH_long_t function returns the value supplied in the netlong parameter with the byte order reversed.

Parameters

- `netlong`: - A 32-bit number in TCP/IP network byte order.

typedef rfUint16 (***hton_short_t**)(rfUint16 hostshort)

The modbusHtoN_short_t function converts a u_short from host to TCP/IP network byte order (which is big-endian).

Return : The modbusHtoN_short_t function returns the value in TCP/IP's network byte order.

Parameters

- `hostlong`: - A 16-bit number in host byte order.

typedef rfUint16 (***ntoh_short_t**)(rfUint16 netshort)

The modbusHtoN_short_t function converts a u_short from TCP/IP network byte order to host byte order.

Return : The modbusNtoH_short_t function returns the value in host byte order.

Parameters

- `netshort`: - A 16-bit number in TCP/IP network byte order.

typedef void *(***create_udp_socket_t**)()

Pointer to TCP socket creation function.

Return

- On success: If no error occurs, modbusCreateTcpSocket_t returns a descriptor referencing the new socket
- On error: NULL

Parameters

- `af`: - The address family specification.
- `type`: - The type specification for the new socket.
- `protocol`: - The protocol to be used.

typedef rfInt8 (***set_broadcast_socket_option_t**)(void *socket)

Pointer to the function that sets a socket option.

Return

- On success: If no error occurs, modbusSetSocketOption_t returns zero
- On error: -1

Parameters

- `socket`: - A descriptor that identifies a socket.
- `level`: - The level at which the option is defined.
- `optname`: - The socket option for which the value is to be set.
- `optval`: - A pointer to the buffer in which the value for the requested option is specified.
- `optlen`: - The size, in bytes, of the buffer pointed to by the optval parameter.

typedef rfInt8 (***set_reuseaddr_socket_option_t**)(void *socket)

Pointer to the function that sets a socket option.

Return

- On success: If no error occurs, modbusSetSocketOption_t returns zero
- On error: -1

Parameters

- `socket`: - A descriptor that identifies a socket.

- `level`: - The level at which the option is defined.
- `optname`: - The socket option for which the value is to be set.
- `optval`: - A pointer to the buffer in which the value for the requested option is specified.
- `optlen`: - The size, in bytes, of the buffer pointed to by the optval parameter.

typedef rfInt8 (***set_socket_option_t**)(void *socket, rfInt32 level, rfInt32 optname, **const** rfChar *optval, rfInt32 optlen)

Pointer to the function that sets a socket option.

Return

- On success: If no error occurs, modbusSetSocketOption_t returns zero
- On error: -1

Parameters

- `socket`: - A descriptor that identifies a socket.
- `level`: - The level at which the option is defined.
- `optname`: - The socket option for which the value is to be set.
- `optval`: - A pointer to the buffer in which the value for the requested option is specified.
- `optlen`: - The size, in bytes, of the buffer pointed to by the optval parameter.

typedef rfInt8 (***set_socket_recv_timeout_t**)(void *socket, rfInt32 msec)

Pointer to the function that sets a timeout for socket receive.

Return

- On success: If no error occurs, returns zero
- On error: -1

Parameters

- `socket`: - A descriptor that identifies a socket.
- `msec`: - The timeout in millisec.

typedef rfUint8 (***socket_connect_t**)(void *socket, rfUint32 dst_ip_addr, rfUint16 dst_port)

Pointer to the function that establishes a connection to a specified socket.

Return

- On success: If no error occurs, modbusSocketConnect_t returns zero
- On error: -1

Parameters

- `socket`: - A descriptor identifying an unconnected socket.
- `name`: - A pointer to the SockAddr structure to which the connection should be established.

- `namelen`: - The length, in bytes, of the SockAddr structure pointed to by the name parameter.

typedef rfInt (***socket_bind_t**)(void *socket, rfUint32 ip_addr, rfUint16 port)

Pointer to the function that associates a local address with a socket.

Return

- On success: If no error occurs, modbusSocketBind_t returns zero
- On error: -1

Parameters

- `socket`: - A descriptor identifying an unconnected socket.
- `name`: - A pointer to the SockAddr structure to which the connection should be established.
- `namelen`: - The length, in bytes, of the SockAddr structure pointed to by the name parameter.

typedef rfUint8 (***socket_listen_t**)(void *socket, rfInt32 backlog)

Pointer to the function that places a socket in a state in which it is listening for an incoming connection.

Return

- On success: If no error occurs, modbusSocketListen_t returns zero
- On error: -1

Parameters

- `socket`: - A descriptor identifying a bound, unconnected socket.
- `backlog`: - The maximum length of the queue of pending connections.

typedef void *(***socket_accept_t**)(void *socket, rfUint32 *srs_ip_addr, rfUint16 *srs_port)

Pointer to the function that permits an incoming connection attempt on a socket.

Return

- On success: If no error occurs, modbusSocketAccept_t returns value is a handle for the socket on which the actual connection is made
- On error : NULL

Parameters

- `socket`: - A descriptor that identifies a socket that has been placed in a listening state with the modbusSocketListen_t function. The connection is actually made with the socket that is returned by accept.
- `name`: - An optional pointer to a buffer that receives the address of the connecting entity, as known to the communications layer. The exact format of the addr parameter is determined by the address family that was established when the socket from the SockAddr structure was created.

- `addrlen`: - An optional pointer to an rfInteger that contains the length of structure pointed to by the addr parameter.

typedef rfUint8 (*__close_socket_t__)(void *socket)

Pointer to the function that closes an existing socket.

Return

- On success: If no error occurs, modbusCloseTcpSocket_t returns zero.
- On error: -1

Parameters

- `socket`: - A descriptor identifying the socket to close.

typedef rfInt (*__send_tcp_data_t__)(void *socket, **const** void *buf, rfSize len)

Pointer to the send function that sends data on a TCP connected socket.

Return

- On success: If no error occurs, send returns the total number of bytes sent, which can be less than the number requested to be sent in the len parameter
- On error: -1

Parameters

- `socket`: - A descriptor identifying a connected socket.
- `buf`: - A pointer to a buffer containing the data to be transmitted.
- `len`: - The length, in bytes, of the data in buffer pointed to by the buf parameter.

typedef rfInt (*__send_udp_data_t__)(void *socket, **const** void *data, rfSize len, rfUint32 dest_ip_addr, rfUint16 dest_port)

Pointer to the send function that sends data on a UDP socket.

Return

- On success: If no error occurs, send returns the total number of bytes sent, which can be less than the number requested to be sent in the len parameter
- On error: -1

Parameters

- `socket`: - A descriptor identifying a socket.
- `buf`: - A pointer to a buffer containing the message to be sent.
- `len`: - The size of the message in bytes.
- `dest_addr`: - Points to a sockaddr_in structure containing the destination address.
- `addrlen`: - Specifies the length of the sockaddr_in structure pointed to by the dest_addr argument.

typedef rfInt (*__recv_data_from_t__)(void *sockfd, void *buf, rfSize len, rfUint32 *srs_ip_addr, rfUint16 *srs_port)

Pointer to the function that receive message from socket and capture address of sender.

Return If successful - the number of bytes received. On failure, it returns a value of -1

Parameters

- `sockfd`: - Specifies a socket descriptor from which data should be received.
- `buf`: - Specifies the buffer in which to place the message.
- `len`: - Specifies the length of the buffer area.
- `src_addr`: - Specifies a socket address structure to record the address of the message sender.
- `addrlen`: - Specifies the length of the sender's address.

typedef rfInt (***recv_data_t**)(void *socket, void *buf, rfSize len)
Pointer to the function that receive message from socket and capture address of sender.

Return If successful - the number of bytes received. On failure, it returns a value of -1

Parameters

- `sockfd`: - Specifies a socket descriptor from which data should be received.
- `buf`: - Specifies the buffer in which to place the message.
- `len`: - Specifies the length of the buffer area.

iostream_platform.h

typedef rfInt (***trace_info_t**)(**const** rfChar *msg, ...)
Method for outputting debugging information.

typedef rfInt (***trace_warning_t**)(**const** rfChar *msg, ...)
Method for outputting alert information.

typedef rfInt (***trace_error_t**)(**const** rfChar *msg, ...)
Method for outputting error information.

The launch of the «Core»

After all platform-specific functions are implemented, the following structures `iostream_platform_dependent_methods_t`, `memory_platform_dependent_methods_t` and `network_platform_dependent_methods_t` must be initialized.

struct memory_platform_dependent_methods_t

Public Members

calloc_t **rf_calloc**

malloc_t **rf_malloc**

realloc_t **rf_realloc**

free_t **rf_free**

memset_t **rf_memset**

memcpy_t **rf_memcpy**

memcmp_t **rf_memcmp**

struct network_platform_dependent_methods_t

Public Members

hton_long_t **hton_long**

ntoh_long_t **ntoh_long**

hton_short_t **hton_short**

ntoh_short_t **ntoh_short**

create_udp_socket_t **create_udp_socket**

set_broadcast_socket_option_t **set_broadcast_socket_option**

set_reuseaddr_socket_option_t **set_reuseaddr_socket_option**

set_socket_option_t **set_socket_option**

set_socket_recv_timeout_t **set_socket_recv_timeout**

socket_connect_t **socket_connect**

socket_bind_t **socket_bind**

socket_listen_t **socket_listen**

socket_accept_t **socket_accept**

close_socket_t **close_socket**

send_tcp_data_t **send_tcp_data**

send_udp_data_t **send_udp_data**

recv_data_from_t **recv_data_from**

recv_data_t **recv_data**

struct iostream_platform_dependent_methods_t

Public Members

trace_info_t **trace_info**

trace_warning_t **trace_warning**

trace_error_t **trace_error**

struct network_platform_dependent_settings_t

Public Members

rfUint32 **host_ip_addr**

rfUint32 **host_mask**

Initialization of these structures is done by assigning pointers to implemented platform-dependent functions, and the addresses of the initialized instances of structures are passed to the init_platform_dependent_methods method to initialize the cross-platform part of the «Core».

void **init_platform_dependent_methods**(*memory_platform_dependent_methods_t *memory_methods*, *iostream_platform_dependent_methods_t *iostream_methods*, *network_platform_dependent_methods_t *network_methods*, *network_platform_dependent_settings_t *adapter_settings*)

init_platform_dependent_methods - Init platform dependent methods and settings

Parameters

- `memory_methods`: Structure with platform-specific methods for work with memory
- `iostream_methods`: Structure with platform-specific methods for work with iostream
- `network_methods`: Structure with platform-specific methods for work with network
- `adapter_settings`: Structure with platform-specific settings

1.3.2 Compiling a wrapper in C

This library allows you to simplify the development of C applications

To use it in C projects, the developer must include the necessary library h-files in his project and build a static or dynamic program library.

Table 2: Latest releases:

Compiler	32bit(Debug)	32bit(Release)	64bit(Debug)	64bit(Release)	Includes
MinGW 7.3.0	rf62Xsdk.dll	rf62Xsdk.dll	rf62Xsdk.dll	rf62Xsdk.dll	includes
MSVC 2017	rf62Xsdk.dll rf62Xsdk.lib	rf62Xsdk.dll rf62Xsdk.lib	rf62Xsdk.dll rf62Xsdk.lib	rf62Xsdk.dll rf62Xsdk.lib	includes

How to compile

The rf62Xsdk wrapper library can be compiled using the console or development environment (Visual Studio, Qt Creator)

First, you must download the project (if you haven't done so before)

Note: for more information on the steps of loading a project see *Download project*

CMake

Being in the folder with the project, to build the "wrapper" library (rf62Xsdk library) enter the following command into the console (terminal):

```
cd rf62Xwrappers/C/rf62Xsdk
mkdir build
cd build
cmake ..
cmake --build .
```

Qt Creator

To build a wrapper library (rf62Xsdk libraries) using the Qt Creator IDE:

- Load the CMakeLists.txt file from the **rf62Xwrappers/C/rf62Xsdk** folder via **File > Open File or Project** (select the CMakeLists.txt file)
- Select the compiler (MinGW, MSVC2017, Clang) and click **Configure Project**
- Open **Build Settings** and check **install** for **Build Steps**
- Compile the project

Visual Studio

Being in the folder with the project, to build RF62X CORE (rf62Xcore libraries) enter the following command into the console (terminal):

```
cd rf62Xwrappers/C/rf62Xsdk
mkdir build
cd build
cmake ..
```

- Open the resulting rf62Xsdk.sln solution in Visual Studio
- Compile the project

How to use

You can **build your project** by including a static or dynamic library and the required headers, or you can **open and compile** one of the below usage examples from the folder **examples/C/RF627_old/**.

Note: Besides the examples below, where each can be compiled and executed, you can also read the documentation for the C++ wrapper (see rf62x_wrappers_description_c), where each function contains a separate example code.

Find RF62X devices

Below is an example of searching for RF627Old series scanners on the network

```
#include <network.h>

#include <stdio.h>
#include <stdlib.h>

#include <rf62Xcore.h>
#include <rf62X_sdk.h>
#include <rf62X_types.h>

using namespace SDK::SCANNERS::RF62X;

int main()
{

   // Initialize sdk library
   core_init();

   // Print return rf627 sdk version
   printf("SDK version: %s\n", sdk_version());
   printf("========================================\n");

   // Create value for scanners vector's type
   vector_t* scanners = (vector_t*)calloc(1, sizeof (vector_t));
   //Initialization vector
   vector_init(&scanners);

   //Iterate over all available network adapters in the current operating
   //system to send "Hello" requests.
   for (int i=0; i<GetAdaptersCount(); i++)
   {
      // get another IP Addr and set this changes in network adapter settings.
      uint32_t host_ip_addr = ntohl(inet_addr(GetAdapterAddress(i)));
      uint32_t host_mask = ntohl(inet_addr("255.255.255.0"));
      // call the function to change adapter settings inside the library.
      set_platform_adapter_settings(host_mask, host_ip_addr);

      // Search for RF627-old devices over network by Service Protocol.
      search_scanners(scanners, kRF627_OLD, kSERVICE);
   }
```

(continues on next page)

(continued from previous page)

```
   // Print count of discovered RF627Old in network by Service Protocol
   printf("Discovered: %d rf627-old\n", (int)vector_count(scanners));

   for (int i = 0; i < (int)vector_count(scanners); i++)
   {
      hello_information info = get_info_about_scanner(vector_get(scanners,i),␣
↪kSERVICE);

      printf("\n\n\nID scanner's list: %d\n", i);
      printf("----------------------------------------\n");
      printf("Device information: \n");
      printf("* Name\t: %s\n", info.rf627old.hello_info_service_protocol->
↪device_name);
      printf("* Serial\t: %d\n", info.rf627old.hello_info_service_protocol->
↪serial_number);
      printf("* IP Addr\t: %d.%d.%d.%d\n",
             info.rf627old.hello_info_service_protocol->ip_address[0],
             info.rf627old.hello_info_service_protocol->ip_address[1],
             info.rf627old.hello_info_service_protocol->ip_address[2],
             info.rf627old.hello_info_service_protocol->ip_address[3]);
      printf("* MAC Addr\t: %d:%d:%d:%d:%d:%d\n",
             info.rf627old.hello_info_service_protocol->mac_address[0],
             info.rf627old.hello_info_service_protocol->mac_address[1],
             info.rf627old.hello_info_service_protocol->mac_address[2],
             info.rf627old.hello_info_service_protocol->mac_address[3],
             info.rf627old.hello_info_service_protocol->mac_address[4],
             info.rf627old.hello_info_service_protocol->mac_address[5]);

      printf("\nWorking ranges: \n");
      printf("* Zsmr, mm\t: %d\n", info.rf627old.hello_info_service_protocol->
↪z_begin);
      printf("* Zmr , mm\t: %d\n", info.rf627old.hello_info_service_protocol->
↪z_range);
      printf("* Xsmr, mm\t: %d\n", info.rf627old.hello_info_service_protocol->
↪x_begin);
      printf("* Xemr, mm\t: %d\n", info.rf627old.hello_info_service_protocol->
↪x_end);

      printf("\nVersions: \n");
      printf("* Firmware\t: %d\n", info.rf627old.hello_info_service_protocol->
↪firmware_version);
      printf("* Hardware\t: %d\n", info.rf627old.hello_info_service_protocol->
↪hardware_version);
      printf("----------------------------------------\n");
   }

   // Cleanup resources allocated with core_init()
   FreeAdapterAddresses();
   WinSockDeinit();
}
```

Below is the output of the application upon successful detection of the scanner on the network:

```
SDK version: 1.3.0
========================================
Discovered: 1 RF627Old
```

(continues on next page)

(continued from previous page)

```
ID scanner's list: 0
----------------------------------------
Device information:
* Name       : RF627
* Serial     : 190068
* IP Addr    : 192.168.1.32
* MAC Addr   : 00:0a:35:6e:07:f5

Working ranges:
* Zsmr, mm   : 70
* Zmr , mm   : 50
* Xsmr, mm   : 30
* Xemr, mm   : 42

Versions:
* Firmware   : 19.11.12
* Hardware   : 18.6.20
----------------------------------------
Press any key to continue . . .
```

You can open and compile this example with **Qt Creator**:

- Download the CMakeLists.txt file from the folder **examples/C/RF627_old/RF627_search** via **File > Open File or Project** (select the CMakeLists.txt file)
- Select the compiler (MinGW, MSVC2017, Clang) and click **Configure Project**
- Run the project

1.3.3 Compiling a wrapper in C++

This library makes it easy to develop C++ applications

To use it in C++ projects, the developer must include the necessary library h-files in his project and build a static or dynamic program library.

Table 3: Latest releases:

Compiler	64bit	Includes
MinGW 7.3.0	rf62Xsdk.dll rf62Xsdk.a	include.zip
MSVC2017	rf62Xsdk.dll rf62Xsdk.lib	include.zip
Clang 9.1.0	rf62Xsdk.dll rf62Xsdk.lib	include.zip

How to compile

The wrapper library rf62Xsdk can be compiled using the console or the development environment (Visual Studio, Qt Creator)

First, you should download the project (if you haven't done this before)

Note: for more information on project loading steps see *Download project*

CMake

From the folder with the project, to build the wrapper library (rf62Xsdk library), enter the following command into the console (terminal):

```
cd rf62Xwrappers/Cpp/rf62Xsdk
mkdir build
cd build
cmake ..
cmake --build .
```

Qt Creator

To build a wrapper library (rf62Xsdk library) using the IDE Qt Creator:

- Download the CMakeLists.txt file from the **rf62Xwrappers/Cpp/rf62Xsdk** folder through **File > Open File or Project** (select the CMakeLists.txt file)
- Select the compiler (MinGW, MSVC2017, Clang) and click **Configure Project**
- Open **Build Settings** and check **install** for **Build Steps**
- Compile the project

Visual Studio

From the project folder, to build the RF62X CORE (rf62Xcore library), enter the following command into the console (terminal):

```
cd rf62Xwrappers/Cpp/rf62Xsdk
mkdir build
cd build
cmake ..
```

- Open the resulting rf62Xsdk.sln solution in Visual Studio
- Compile the project

How to use

You can **create your project** by including a static or dynamic library and the necessary header files in it, or you can **open and compile** one of the following examples from the folder **examples/Cpp/RF627_old/**.

Note: In addition to the examples below, where everyone can be compiled and executed, you can also read the documentation for C++ wrappers (see rf62x_wrappers_description_cpp), where each function contains a separate code example.

Search for RF62X devices

Below is an example of searching the RF627Old series of scanners on the network

```
#include <rf62Xsdk.h>
#include <rf62Xtypes.h>
#include <string>
#include <iostream>

using namespace SDK::SCANNERS::RF62X;

int main()
{

   // Initialize sdk library
   sdk_init();

   // Print return rf627 sdk version
   std::cout << "SDK version: " << sdk_version()                 << std::endl;
   std::cout << "========================================"      << std::endl;

   // Create value for scanners vector's type
   std::vector<rf627old*> list;
   // Search for RF627old devices over network
   list = rf627old::search(PROTOCOLS::SERVICE);

   // Print count of discovered RF627Old in network by Service Protocol
   std::cout << "Discovered: " << list.size() << " RF627Old"   << std::endl;

   for (size_t i = 0; i < list.size(); i++)
   {
      rf627old::hello_info info = list[i]->get_info();

      std::cout << "\n\n\nID scanner's list: " << i             << std::endl;
      std::cout << "----------------------------------------" << std::endl;
      std::cout << "Device information: "                      << std::endl;
      std::cout << "* Name\t: "     << info.device_name()      << std::endl;
      std::cout << "* Serial\t: "   << info.serial_number()    << std::endl;
      std::cout << "* IP Addr\t: "  << info.ip_address()       << std::endl;
      std::cout << "* MAC Addr\t: " << info.mac_address()      << std::endl;

      std::cout << "\nWorking ranges: "                        << std::endl;
      std::cout << "* Zsmr, mm\t: " << info.z_smr()            << std::endl;
      std::cout << "* Zmr , mm\t: " << info.z_mr()             << std::endl;
      std::cout << "* Xsmr, mm\t: " << info.x_smr()            << std::endl;
      std::cout << "* Xemr, mm\t: " << info.x_emr()            << std::endl;

      std::cout << "\nVersions: "                              << std::endl;
      std::cout << "* Firmware\t: " << info.firmware_version() << std::endl;
      std::cout << "* Hardware\t: " << info.hardware_version() << std::endl;
      std::cout << "----------------------------------------" << std::endl;
   }

   system("pause");
}
```

The following is the output of the application upon successful detection of the scanner on the network:

```
SDK version: 1.3.0
========================================
Discovered: 1 RF627Old

ID scanner's list: 0
----------------------------------------
Device information:
* Name       : RF627
* Serial     : 190068
* IP Addr    : 192.168.1.32
* MAC Addr   : 00:0a:35:6e:07:f5

Working ranges:
* Zsmr, mm   : 70
* Zmr , mm   : 50
* Xsmr, mm   : 30
* Xemr, mm   : 42

Versions:
* Firmware   : 19.11.12
* Hardware   : 18.6.20
----------------------------------------
Press any key to continue . . .
```

You can open and compile this example with **Qt Creator**:

- Download the CMakeLists.txt file from the **examples/Cpp/RF627_old/RF627_search** folder through **File > Open File or Project** (select the CMakeLists.txt file)
- Select the compiler (MinGW, MSVC2017, Clang) and click **Configure Project**
- Run the project

Getting a scanner profile

The following is an example of retrieving profiles from RF627Old Series Scanners

```cpp
#include <rf62Xsdk.h>
#include <rf62Xtypes.h>
#include <string>
#include <iostream>

using namespace SDK::SCANNERS::RF62X;

int main()
{

    // Initialize sdk library
   sdk_init();

   // Print return rf627 sdk version
   std::cout << "SDK version: " << sdk_version()                << std::endl;
   std::cout << "========================================"      << std::endl;


    // Create value for scanners vector's type
   std::vector<rf627old*> list;
```

(continues on next page)

(continued from previous page)

```
    // Search for RF627old devices over network
    list = rf627old::search(PROTOCOLS::SERVICE);


    // Print count of discovered RF627Old in network by Service Protocol
    std::cout << "Discovered: " << list.size() << " RF627Old"   << std::endl;


    // Iterate over all discovered RF627Old in network, connect to each of
    // them and get a profile.
    for(size_t i = 0; i < scanners.size(); i++)
    {
        rf627old::hello_info info = list[i]->get_info();

        // Print information about the scanner to which the profile belongs.
        std::cout << "\n\n\nID scanner's list: " << i            << std::endl;
        std::cout << "----------------------------------------" << std::endl;
        std::cout << "Device information: "                     << std::endl;
        std::cout << "* Name\t: "     << info.device_name()      << std::endl;
        std::cout << "* Serial\t: "   << info.serial_number()    << std::endl;
        std::cout << "* IP Addr\t: "  << info.ip_address()       << std::endl;

        // Establish connection to the RF627 device by Service Protocol.
        list[i]->connect();

        // Get profile from scanner's data stream by Service Protocol.
        profile2D_t* profile = list[i]->get_profile2D();
        if (profile != nullptr)
        {
            std::cout << "Profile information: "                  << std::endl;
            switch (profile->header.data_type) {
            case (uint8_t)PROFILE_DATA_TYPE::PIXELS:
                std::cout << "* DataType\t: "<< "PIXELS"          << std::endl;
                std::cout << "* Count\t: " << profile->pixels.size()<< std::endl;
                break;
            case (uint8_t)PROFILE_DATA_TYPE::PROFILE:
                std::cout << "* DataType\t: "<< "PROFILE"         << std::endl;
                std::cout << "* Size\t: "  << profile->points.size()<< std::endl;
                break;
            case (uint8_t)PROFILE_DATA_TYPE::PIXELS_INTRP:
                std::cout << "* DataType\t: "<< "PIXELS_INTRP"    << std::endl;
                std::cout << "* Count\t: " << profile->pixels.size()<< std::endl;
                break;
            case (uint8_t)PROFILE_DATA_TYPE::PROFILE_INTRP:
                std::cout << "* DataType\t: "<< "PROFILE_INTRP"   << std::endl;
                std::cout << "* Size\t: "  << profile->points.size()<< std::endl;
                break;
            }
            std::cout << "Profile was successfully received!"    << std::endl;
            std::cout << "----------------------------------------"<< std::endl;
        }else
        {
            std::cout << "Profile was not received!"             << std::endl;
            std::cout << "----------------------------------------"<< std::endl;
        }

    }
```

(continues on next page)

(continued from previous page)

```
    system("pause");
}
```

The following is the result of the output of the application upon successful receipt of the profile from the scanner:

```
SDK version: 1.3.0
========================================
Discovered: 1 RF627Old


ID scanner's list: 0
----------------------------------------
Device information:
* Name       : RF627
* Serial     : 190068
* IP Addr    : 192.168.1.32
Profile information:
* DataType   : PROFILE
* Size       : 648
Profile was successfully received!
----------------------------------------
Press any key to continue . . .
```

You can open and compile this example with **Qt Creator**:

- Download the CMakeLists.txt file from the **examples/Cpp/RF627_old/RF627_profile** folder through **File > Open File or Project** (select the CMakeLists.txt file)
- Select the compiler (MinGW, MSVC2017, Clang) and click **Configure Project**
- Run the project

Getting and setting parameters

The following is an example of obtaining and changing the name of the scanner, setting the IP address, changing the state of the laser (turning it on or off):

```
#include <rf62Xsdk.h>
#include <rf62Xtypes.h>
#include <iostream>
#include <string>

using namespace SDK::SCANNERS::RF62X;

int main()
{

    // Initialize sdk library
    sdk_init();

    // Print return rf62X SDK version
    std::cout << "SDK version: " << sdk_version()                    <<␣
↪std::endl;
    std::cout << "========================================"          <<␣
↪std::endl;
```

(continues on next page)

(continued from previous page)

```
   // Create value for scanners vector's type
   std::vector<rf627old*> scanners;
   // Search for RF627old devices over network
   scanners = rf627old::search(PROTOCOLS::SERVICE);


   // Print count of discovered RF627Old in network by Service Protocol
   std::cout << "Discovered: " << scanners.size() << " RF627Old"  <<␣
↪std::endl;


   // Iterate over all discovered RF627Old in network, connect to each of
   // them and read/set parameters.
   for(size_t i = 0; i < scanners.size(); i++)
   {

      rf627old::hello_info info = scanners[i]->get_info();

      std::cout << "\n\n\nID scanner's list: " << i               <<␣
↪std::endl;
      std::cout << "----------------------------------------"    <<␣
↪std::endl;

      // Establish connection to the RF627 device by Service Protocol.
      scanners[i]->connect();

      // read params from RF627 device by Service Protocol.
      scanners[i]->read_params();

      // Get parameter of Device Name
      param_t* name = scanners[i]->get_param(PARAM_NAME_KEY::USER_GENERAL_
↪DEVICENAME);
      if (name->type == param_value_types[(int)PARAM_VALUE_TYPE::STRING_PARAM_
↪TYPE])
      {
         std::string str_name = name->get_value<value_str>();
         std::cout << "Current Device Name \t: " << str_name     << std::endl;

         // Add "_TEST" to the ending of the current name
         str_name += "_TEST";
         name->set_value<value_str>(str_name);
         std::cout << "New Device Name \t: " << str_name         << std::endl;
         std::cout << "----------------------------------------"<< std::endl;

         scanners[i]->set_param(name);
      }

      // Get parameter of Device IP Addr
      param_t* ip_addr = scanners[i]->get_param(PARAM_NAME_KEY::USER_NETWORK_
↪IP);
      if (ip_addr->type == param_value_types[(int)PARAM_VALUE_TYPE::UINT32_
↪ARRAY_PARAM_TYPE])
      {
         std::vector <uint32_t> ip = ip_addr->get_value<array_uint32>();
         std::cout << "Current Device IP\t: ";
```

(continues on next page)

(continued from previous page)

```
        for(auto i: ip) std::cout<<std::to_string(i)<<".";std::cout<
↪<std::endl;

        // Change last digit of IP address (e.g. 192.168.1.30 -> 192.168.1.
↪31)
        ip[3]++;
        ip_addr->set_value<array_uint32>(ip);
        std::cout << "New Device IP\t: ";
        for(auto i: ip) std::cout<<std::to_string(i)<<".";std::cout<
↪<std::endl;
        std::cout << "----------------------------------------"<< std::endl;

        scanners[i]->set_param(ip_addr);
     }

     // Get parameter of Laser Enabled
     param_t* laser_enabled = scanners[i]->get_param(PARAM_NAME_KEY::USER_
↪LASER_ENABLED);
     if (laser_enabled->type == param_value_types[(int)PARAM_VALUE_
↪TYPE::UINT_PARAM_TYPE])
     {
        bool isEnabled = laser_enabled->get_value<value_uint32>();
        std::cout<<"Current Laser State\t: "<<(isEnabled?"ON":"OFF")<
↪<std::endl;

        isEnabled = !isEnabled;
        // Change the current state to the opposite
        laser_enabled->set_value<value_uint32>(!isEnabled);
        std::cout<<"New Laser State\t: "<<(isEnabled?"ON":"OFF")<<std::endl;
        std::cout << "----------------------------------------"<< std::endl;

        scanners[i]->set_param(laser_enabled);
     }

     //  Write changes parameters to the device's memory
     scanners[i]->write_params();

  }

  system("pause");

}
```

The following is the output from the application upon successful installation of the new parameters:

```
SDK version: 1.3.0
========================================
Discovered: 1 RF627Old


ID scanner's list: 0
----------------------------------------
Current Device Name   : RF627
New Device Name       : RF627_TEST
----------------------------------------
Current Device IP     : 192.168.1.32.
New Device IP         : 192.168.1.33.
```

(continues on next page)

(continued from previous page)

```
----------------------------------------
Current Laser State  : ON
New Laser State      : OFF
----------------------------------------

Press any key to continue . . .
```

You can open and compile this example with **Qt Creator**:

- Download the CMakeLists.txt file from the **examples/Cpp/RF627_old/RF627_params** folder through **File > Open File or Project** (select the CMakeLists.txt file)
- Select the compiler (MinGW, MSVC2017, Clang) and click **Configure Project**
- Run the project

1.3.4 Compiling a wrapper in C#

This wrapper is a .NET library written in C# that can be used in applications in C#, Visual Basic .NET, C++/CLI, and JScript .NET.

To use it in .NET projects, the developer must compile or download the dynamic program library **rf62Xsdk.dll**, then add the library to the project references and collect or download **rf62Xcore.dll** by adding it to folder to the project executable file.

Table 4: Latest releases:

Platform	64bit	Dependencies (x64)
.NET Framework 4.5 (or above)	rf62Xsdk.dll	rf62Xcore.dll

How to compile

The wrapper library rf62Xsdk can be compiled using the Visual Studio development environment.

First, you should download the project (if you haven't done this before)

```
git clone https://gitlab.com/riftek_llc/software/sdk/scanners/RF62X-SDK.git
cd RF62X-SDK
git submodule update --init --recursive
```

Note: for more information on project loading steps see *Download project*

Visual Studio

- Open rf62Xsdk.sln solution along the path **rf62Xwrappers/CSharp/rf62Xsdk** in Visual Studio
- Compile the project

How to use

You can open usage examples with **Visual Studio**, for this:

- Open the solution **RF627_TESTS.sln** from the folder **rf62Xwrappers/CSharp/RF627_old**
- Select **x64 Debug** or **x64 Release** as the target platform
- Add **rf62Xsdk.dll** library to the project **references**
- Copy **rf62Xcore.dll** to the path to the project executable (**../bin/x64/Debug/** or **../bin/x64/Release/**)
- Compile the project

In addition to the examples of using the library from the solution **RF627_TESTS.sln**, where each example can be compiled and executed separately, you can also read the documentation for the wrapper library in C#, where each function contains a separate example code.

Search for RF62X devices

Below is an example of searching the RF627Old series of scanners on the network

```
using System;
using System.Collections.Generic;
using SDK.SCANNERS;

namespace RF627_search
{
    class Program
    {
        static void Main(string[] args)
        {

            // Start initialization of the library core
            RF62X.SdkInit();

            // Print return rf62X sdk version
            Console.WriteLine("SDK version: {0}", RF62X.SdkVersion());
            Console.WriteLine("========================================");

            // Search for RF627old devices over network
            Console.WriteLine("- Start searching device");
            List<RF62X.RF627old> Scanners = RF62X.RF627old.Search();
            Console.WriteLine("+ {0} scanners detected", Scanners.Count);

            for (int i = 0; i < Scanners.Count; i++)
            {
               RF62X.HelloInfo info = Scanners[i].GetInfo();
```

(continues on next page)

(continued from previous page)

```
                Console.WriteLine("\n\n\nID scanner's list: {0}", i);
                Console.WriteLine("----------------------------------------");
                Console.WriteLine("Device information: ");
                Console.WriteLine("* Name\t: {0}", info.device_name);
                Console.WriteLine("* Serial\t: {0}", info.serial_number);
                Console.WriteLine("* IP Addr\t: {0}", info.ip_address);
                Console.WriteLine("* MAC Addr\t: {0}", info.mac_address);

                Console.WriteLine("Working ranges: ");
                Console.WriteLine("* Zsmr, mm\t: {0}", info.z_smr);
                Console.WriteLine("* Zmr , mm\t: {0}", info.z_mr);
                Console.WriteLine("* Xsmr, mm\t: {0}", info.x_smr);
                Console.WriteLine("* Xemr, mm\t: {0}", info.x_emr);

                Console.WriteLine("\nVersions: ");
                Console.WriteLine("* Firmware\t: {0}", info.firmware_version);
                Console.WriteLine("* Hardware\t: {0}", info.hardware_version);
                Console.WriteLine("----------------------------------------");
            }

            Console.WriteLine("{0}Press any key to end \"Search-test\"",␣
↪Environment.NewLine);
            Console.ReadKey();

        }
    }
}
```

Getting a scanner profile

The following is an example of retrieving profiles from RF627Old Series Scanners

```
using System;
using System.Collections.Generic;
using SDK.SCANNERS;

namespace RF627_profile
{
   class Program
   {
      static void Main(string[] args)
      {

         // Start initialization of the library core
         RF62X.SdkInit();


         // Search for RF627old devices over network
         Console.WriteLine("- Start searching device");
         List<RF62X.RF627old> Scanners = RF62X.RF627old.Search();
         Console.WriteLine("+ {0} scanners detected", Scanners.Count);


         // foreach over an scanners list
```

(continues on next page)

(continued from previous page)

```
        for (int i = 0; i < Scanners.Count; i++)
        {
            RF62X.HelloInfo info = Scanners[i].GetInfo();

            Console.WriteLine("\n\n\nID scanner's list: {0}", i);
            Console.WriteLine("----------------------------------------");
            Console.WriteLine("Device information: ");
            Console.WriteLine("* Name\t: {0}", info.device_name);
            Console.WriteLine("* Serial\t: {0}", info.serial_number);
            Console.WriteLine("* IP Addr\t: {0}", info.ip_address);

            // Establish connection to the RF627 device by Service Protocol.
            Scanners[i].Connect();

            // Get profile from scanner's data stream by Service Protocol.
            RF62X.Profile profile = Scanners[i].GetProfile();
            if (profile.header != null)
            {
                Console.WriteLine("Profile information: ");
                switch (profile.header.data_type)
                {
                    case RF62X.PROFILE_TYPE.PIXELS_NORMAL:
                        Console.WriteLine("* DataType\t: PIXELS");
                        Console.WriteLine("* Count\t: {0}", profile.pixels.
↪Count);
                        break;
                    case RF62X.PROFILE_TYPE.PROFILE_NORMAL:
                        Console.WriteLine("* DataType\t: PROFILE");
                        Console.WriteLine("* Size\t: {0}", profile.points.Count);
                        break;
                    case RF62X.PROFILE_TYPE.PIXELS_INTERPOLATED:
                        Console.WriteLine("* DataType\t: PIXELS");
                        Console.WriteLine("* Count\t: {0}", profile.pixels.
↪Count);
                        break;
                    case RF62X.PROFILE_TYPE.PROFILE_INTERPOLATED:
                        Console.WriteLine("* DataType\t: PROFILE");
                        Console.WriteLine("* Size\t: {0}", profile.points.Count);
                        break;
                    default:
                        break;
                }
                Console.WriteLine("Profile was successfully received!");
                Console.WriteLine("----------------------------------------");
            }else
            {
                Console.WriteLine("Profile was not received!");
                Console.WriteLine("----------------------------------------");
            }
        }

        Console.WriteLine("{0}Press any key to end \"Search-test\"",␣
↪Environment.NewLine);
        Console.ReadKey();
    }
  }
}
```

Getting and setting parameters

The following is an example of getting and changing the name of the scanner, setting the IP address, changing the state of the laser (turning it on or off):

```
using System;
using System.Collections.Generic;
using SDK.SCANNERS;

namespace RF627_params
{
   class Program
   {
      static void Main(string[] args)
      {

         // Start initialization of the library core
         RF62X.SdkInit();

         // Search for RF627old devices over network
         Console.WriteLine("- Start searching device");
         List<RF62X.RF627old> Scanners = RF62X.RF627old.Search();
         Console.WriteLine("+ {0} scanners detected", Scanners.Count);

         // foreach over an scanners list
         for (int i = 0; i < Scanners.Count; i++)
         {
            // Establish connection to the RF627 device by Service Protocol.
            Scanners[i].Connect();

            // read params from RF627 device by Service Protocol.
            Scanners[i].ReadParams();

            // Get parameter of Device Name
            RF62X.Param<string> name = Scanners[i].GetParam(RF62X.Params.User.
↪General.deviceName);
            if (name != null)
            {
               string strName = name.GetValue();
               Console.WriteLine("\n\nCurrent Device Name \t: {0}", strName);

               // Add "_TEST" to the ending of the current name
               strName += "_TEST";
               name.SetValue(strName);
               Console.WriteLine("New Device Name \t: {0}", strName);
               Console.WriteLine("----------------------------------------");

               Scanners[i].SetParam(name);
            }

            // Get parameter of Device IP Addr
            RF62X.Param<List<uint>> ipAddr = Scanners[i].GetParam(RF62X.
↪Params.User.NetWork.ip);
            if (ipAddr != null)
            {
               List<uint> ip = ipAddr.GetValue();
               Console.WriteLine("Current Device IP Addr\t: {0}.{1}.{2}.{3}",␣
↪ip[0], ip[1], ip[2], ip[3]);
```

(continues on next page)

(continued from previous page)

```
                // Change last digit of IP address (e.g. 192.168.1.30 -> 192.
↪168.1.31)
                ip[3]++;
                ipAddr.SetValue(ip);
                Console.WriteLine("New Device IP Addr\t: {0}.{1}.{2}.{3}",␣
↪ip[0], ip[1], ip[2], ip[3]);
                Console.WriteLine("----------------------------------------");

                Scanners[i].SetParam(ipAddr);
            }

            // Get parameter of Laser Enabled
            RF62X.Param<uint> laserEnabled = Scanners[i].GetParam(RF62X.
↪Params.User.Laser.enabled);
            if (laserEnabled != null)
            {
                bool isLaserEnabled = Convert.ToBoolean(laserEnabled.
↪GetValue());
                Console.WriteLine("Current Laser State\t: {0}", isLaserEnabled␣
↪? "ON" : "OFF");

                // Change the current state to the opposite
                isLaserEnabled = !isLaserEnabled;
                laserEnabled.SetValue((uint)(Convert.
↪ToUInt32(isLaserEnabled)));
                Console.WriteLine("New Laser State\t\t: {0}", isLaserEnabled ?
↪"ON" : "OFF");
                Console.WriteLine("----------------------------------------");

                Scanners[i].SetParam(laserEnabled);
            }

            Scanners[i].WriteParams();
        }
        Console.WriteLine("{0}Press any key to end \"Parameters-test\"",␣
↪Environment.NewLine);
        Console.ReadKey();
    }
  }
}
```

1.4 Additional Information

1.4.1 PARAMETERS

Description of device parameters. All device parameters are readable. Those parameters that cannot be written without authorization as manufacturer have “awrite” access type. If no minimum value is specified for a parameter, the minimum value corresponds to the minimum value of the parameter type. If no maximum value is specified, it corresponds to the maximum value of the parameter type.

The main elements of the parameter description:

- **Access** - describes the availability of the parameter for reading and writing.
  - read - parameter is readable,
  - write - parameter is writable by user,
  - awrite - parameter is writable after authorization as a manufacturer

- **Type** - the data type of the parameter. - uint32_t - unsigned integer, 32 bits,
  - uint64_t - unsigned integer, 64 bits,
  - int32_t - signed integer, 32 bits,
  - int64_t - signed integer, 64 bits,
  - float_t - floating point, 32 bits,
  - double_t - floating point, 64 bits,
  - u32_arr_t - array of unsigned integer, 32 bits,
  - u64_arr_t - array of unsigned integer, 64 bits,
  - i32_arr_t - array of signed integer, 32 bits,
  - i64_arr_t - array of signed integer, 64 bits,
  - flt_array_t - array of floating point, 32 bits,
  - dbl_array_t - array of floating point, 64 bits,
  - string_t - string, ending with “0”, the maximum length of the string is specified in the parameter description

- **Min value** - minimum parameter value, writing a value less than this is not allowed. If no minimum
  value is specified, it is defined by the type of parameter.

- **Max value** - maximum parameter value, writing a value greater than this is
  not allowed. If no maximum value is specified, it is defined by the type
  of parameter.

- **Step** - step with which it is allowed to change the
  parameter value. Values that do not match the step will not be set. If
  no step is specified, any parameter change is allowed.

- **Enum** - enumeration of valid parameter values. Values that do not match the
  enumeration will not be set.

- **Default value** - default value of the parameter, set by the manufacturer
  or after switching on the device (depending on the parameter).

User

- **user_general_deviceState** - Current device state - combination
  of enum values. The device changes the value of this parameter when
  initializing the equipment, transferring important data over the
  network (e.g. firmware), updating the firmware and in other cases. In
  all modes except DEV_STATE_NORMAL, the device can pause the
  transfer of profiles and other data not related to the current
  operating mode.
  - *Access*: read
  - *Type*: uint32_t
  - *Min value*: DEV_STATE_NORMAL
  - *Max value*: DEV_STATE_HARDWARE_INIT
  - *Enum*:
    - DEV_STATE_NORMAL - the device operates in normal mode,
    - DEV_STATE_CALIB_FILE_RCV - the device receives a
      calibration file,
    - DEV_STATE_CALIB_FILE_SND - the device is transfer
      calibration file,
    - DEV_STATE_CALIB_FILE_SAVE - the device saves the
      calibration file to the internal flash drive,
    - DEV_STATE_FIRMWARE_RCV - device receives firmware,
    - DEV_STATE_FIRMWARE_SND - the device is transfer firmware,
    - DEV_STATE_FIRMWARE_SAVE - the device saves the firmware file
      to the internal flash drive,
    - DEV_STATE_ETH_INIT - the device initializes hardware and
      software for ethernet connection,
    - DEV_STATE_DUMP_DOWNLOAD - the device transfers dump data,
    - DEV_STATE_ETH_EXCESS - required connection speed exceeds
      current value for ethernet connection,
    - DEV_STATE_HARDWARE_INIT - device initializes hardware
  - *Default value*: DEV_STATE_NORMAL

- **user_general_deviceName** - User-defined scanner name. It is
  displayed on the web page of the scanner and can be used to quickly
  identify scanners.
  - *Access*: read/write
  - *Type*: string_t
  - *Max len*: 128
  - *Default value*: “2D laser scanner”

- **user_general_logSaveEnabled** - Allow automatic log saving after
  device boot and after critical events. When this option is enabled,
  it slightly (~ 100ms) increases the time until the device is ready
  for operation.

- *Access*: read/write
- *Type*: uint32_t
- *Min value*: FALSE
- *Max value*: TRUE
- *Default value*: FALSE

- **user_general_logSize** - The current size (number of records) of the device's internal log file.
- *Access*: read
- *Type*: uint32_t
- *Default value*: 0

- **user_sysMon_fpgaTemp** - The current temperature of the FPGA (internal computing module) of the device.
- *Access*: read
- *Type*: float_t
- *Min value*: -100
- *Max value*: +100
- *Default value*: 0
- *Units*: °C

- **user_sysMon_paramsChanged** - Device settings have been changed but not saved.
- *Access*: read
- *Type*: uint32_t
- *Min value*: FALSE
- *Max value*: TRUE
- *Default value*: FALSE

- **user_sysMon_tempSens00** - Current temperature inside the device case, measured by the sensor with address 00.
- *Access*: read
- *Type*: float_t
- *Min value*: -100
- *Max value*: +100
- *Default value*: 0
- *Units*: °C

- **user_sysMon_tempSens00Max** - Maximum temperature fixed by sensor with address 00.
- *Access*: read
- *Type*: float_t
- *Min value*: -100

- *Max value*: +100
- *Default value*: 0
- *Units*: °C

- **user_sysMon_tempSens00Min** - Minimum temperature fixed by sensor with address 00.
  - *Access*: read
  - *Type*: float_t
  - *Min value*: -100
  - *Max value*: +100
  - *Default value*: 0
  - *Units*: °C

- **user_sysMon_tempSens01** - Current temperature inside the device case, measured by the sensor with address 01.
  - *Access*: read
  - *Type*: float_t
  - *Min value*: -100
  - *Max value*: +100
  - *Default value*: 0
  - *Units*: °C

- **user_sysMon_tempSens01Max** - Maximum temperature fixed by sensor with address 01.
  - *Access*: read
  - *Type*: float_t
  - *Min value*: -100
  - *Max value*: +100
  - *Default value*: 0
  - *Units*: °C

- **user_sysMon_tempSens01Min** - Minimum temperature fixed by sensor with address 01.
  - *Access*: read
  - *Type*: float_t
  - *Min value*: -100
  - *Max value*: +100
  - *Default value*: 0
  - *Units*: °C

- **user_sysMon_tempSens10** - Current temperature inside the device case, measured by the sensor with address 10.
  - *Access*: read

- *Type*: float_t
- *Min value*: -100
- *Max value*: +100
- *Default value*: 0
- *Units*: °C

- **user_sysMon_tempSens10Max** - Maximum temperature fixed by sensor with address 10.
- *Access*: read
- *Type*: float_t
- *Min value*: -100
- *Max value*: +100
- *Default value*: 0
- *Units*: °C

- **user_sysMon_tempSens10Min** - Minimum temperature fixed by sensor with address 10.
- *Access*: read
- *Type*: float_t
- *Min value*: -100
- *Max value*: +100
- *Default value*: 0
- *Units*: °C

- **user_sysMon_tempSens11** - Current temperature inside the device case, measured by the sensor with address 11.
- *Access*: read
- *Type*: float_t
- *Min value*: -100
- *Max value*: +100
- *Default value*: 0
- *Units*: °C

- **user_sysMon_tempSens11Max** - Maximum temperature fixed by sensor with address 11.
- *Access*: read
- *Type*: float_t
- *Min value*: -100
- *Max value*: +100
- *Default value*: 0
- *Units*: °C

- **user_sysMon_tempSens11Min** - Minimum temperature fixed by sensor

with address 11.
- *Access*: read
- *Type*: float_t
- *Min value*: -100
- *Max value*: +100
- *Default value*: 0
- *Units*: °C

- **user_sensor_syncSource** - Measurement synchronization source.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: SYNC_INTERNAL
  - *Max value*: SYNC_SOFTWARE
  - *Enum*:
    - SYNC_INTERNAL - start of measurements from the device's internal generator,
    - SYNC_EXTERNAL - start of measurements from an external source,
    - SYNC_SOFTWARE - start of measurements by software request
  - *Default value*: SYNC_INTERNAL

- **user_sensor_framerate** - Frame rate of the CMOS-sensor, sets the measurement frequency. The value to be written should not exceed the value of the parameter **user_sensor_maxFramerate**.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: 1
  - *Max value*: 20000
  - *Default value*: 490
  - *Units*: Hz

- **user_sensor_maxFramerate** - Maximum frame rate (measurement frequency) for the current operation mode.
  - *Access*: read
  - *Type*: uint32_t
  - *Min value*: 1
  - *Max value*: 20000
  - *Default value*: 490
  - *Units*: Hz

- **user_sensor_exposureControl** - CMOS-sensor exposure control method.
  - *Access*: read/write
  - *Type*: uint32_t

- *Min value*: EXPOSE_AUTO
- *Max value*: EXPOSE_MULTI_3
- *Enum*:
    - EXPOSE_AUTO - automatic exposure control based on profile analysis,
    - EXPOSE_FIXED - exposure time is user-defined,
    - EXPOSE_MULTI_2 - mode with 2 exposures, used to obtain a profile on surfaces with different levels of reflection,
    - EXPOSE_MULTI_3 - mode with 3 exposures, used to obtain a profile on surfaces with different levels of reflection
- *Default value*: EXPOSE_FIXED

- **user_sensor_exposure1** - Frame exposure time in EXPOSE_AUTO and EXPOSE_FIXED modes.
    - *Access*: read/write
    - *Type*: uint32_t
    - *Min value*: 3000
    - *Max value*: 300000000
    - *Step*: 100
    - *Default value*: 300000
    - *Units*: ns

- **user_sensor_exposure2** - Frame #2 exposure time in EXPOSE_MULTI_2 mode.
    - *Access*: read/write
    - *Type*: uint32_t
    - *Min value*: 3000
    - *Max value*: 300000000
    - *Step*: 100
    - *Default value*: 300000
    - *Units*: ns

- **user_sensor_exposure3** - Frame #3 exposure time in EXPOSE_MULTI_2 mode.
    - *Access*: read/write
    - *Type*: uint32_t
    - *Min value*: 3000
    - *Max value*: 300000000
    - *Step*: 100
    - *Default value*: 300000
    - *Units*: ns

- **user_sensor_maxExposure** - Maximum frame exposure time in the

current device mode.
- *Access*: read/write
- *Type*: uint32_t
- *Min value*: 3000
- *Max value*: 300000000
- *Step*: 100
- *Default value*: 300000
- *Units*: ns

- **user_sensor_defectivePixels** - Array of coordinates [X1, Y1, X2, Y2, … X15, Y15] of the sensor's defective pixels.
- *Access*: read/write
- *Type*: u32_arr_t
- *Max value*: 4096
- *Max elements*: 32,
- *Default value*: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

- **user_sensor_doubleSpeedEnabled** - Turns on and off the sensor's double frame rate mode. Enabling this mode allows almost double the frequency of measurements (profiles per second) by reducing accuracy in the Z-axis.
- *Access*: read/write
- *Type*: uint32_t
- *Min value*: FALSE
- *Max value*: TRUE
- *Default value*: FALSE

- **user_sensor_edrType** - Enable CMOS-sensor operation in the extended dynamic range. Allows get a quality profile on light and dark surfaces.
- *Access*: read/write
- *Type*: uint32_t
- *Min value*: EDR_DISABLE
- *Max value*: EDR_COLUMN
- *Enum*:
  - EDR_DISABLE - extended dynamic range mode disabled,
  - EDR_COLUMN - different exposure mode for even and odd columns
- *Default value*: EDR_DISABLE

- **user_sensor_edrColumnDivider** - Exposure time divider for odd columns. This parameter controls the sensitivity to very bright areas of the profile.

- *Access*: read/write
- *Type*: uint32_t
- *Min value*: 2
- *Max value*: 32
- *Default value*: 2

- **user_roi_enabled** - Turns on and off the mode of obtaining measurements in the region of interest.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: FALSE
  - *Max value*: TRUE
  - *Default value*: FALSE

- **user_roi_active** - Indicates the status of the ROI in automatic positioning mode. In the automatic position control mode, if the profile is not detected, the activity switches to the FALSE state, when the profile is detected, the parameter switches to the TRUE state. In manual positioning mode, the parameter is always TRUE.
  - *Access*: read
  - *Type*: uint32_t
  - *Min value*: FALSE
  - *Max value*: TRUE
  - *Default value*: FALSE

- **user_roi_posMode** - ROI position control mode.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: ROI_POSITION_MANUAL
  - *Max value*: ROI_POSITION_AUTO
  - *Enum*:
    - ROI_POSITION_MANUAL - ROI position is set by the user,
    - ROI_POSITION_AUTO - ROI automatic position control with profile holding in the center
  - *Default value*: ROI_POSITION_MANUAL

- **user_roi_pos** - Current position of the upper edge of the ROI in the sensor lines.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Max value*: 1280
  - *Default value*: 100
  - *Units*: lines

- **user_roi_maxPos** - Maximum position of the upper limit of the
  ROI in the current operating mode of the device.
  - *Access*: read
  - *Type*: uint32_t
  - *Max value*: 1280
  - *Default value*: 1180
  - *Units*: lines

- **user_roi_size** - Sets the size of the area in the lines where
  the profile is searched and processed.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: 8
  - *Max value*: 488
  - *Step*: 8
  - *Default value*: 64
  - *Units*: lines

- **user_roi_reqProfSize** - Minimum required number of profile
  points for activating an ROI in ROI_POSITION_AUTO mode.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Max value*: 1280
  - *Step*: 64
  - *Default value*: 320
  - *Units*: points

- **user_roi_zsmr** - ROI start position in mm.
  - *Access*: read
  - *Type*: float_t
  - *Max value*: 10000
  - *Default value*: 0
  - *Units*: mm

- **user_roi_zemr** - ROI end position in mm.
  - *Access*: read
  - *Type*: float_t
  - *Max value*: 10000
  - *Default value*: 0
  - *Units*: mm

- **user_network_speed** - Current Ethernet connection speed. The
  connection speed is changed by writing to this parameter. In case of

auto-negotiation, writing is ignored.
- *Access*: read/write
- *Type*: uint32_t
- *Min value*: LINK_SPEED_10MBIT
- *Max value*: LINK_SPEED_1GBIT
- *Enum*:
    - LINK_SPEED_10MBIT - the connection speed is 10 Mbs, currently almost unused,
    - LINK_SPEED_100MBIT - the connection speed is 100 Mbs,
    - LINK_SPEED_1GBIT - the connection speed is 1000 Mbs
- *Default value*: LINK_SPEED_1GBIT
- *Units*: Mbps

- **user_network_requiredSpeed** - The required Ethernet connection speed in the current device operation mode. Depends on the number of profiles per second, the number of points in the profile, etc.
- *Access*: read
- *Type*: uint32_t
- *Min value*: 1
- *Max value*: 10000
- *Default value*: 1
- *Units*: Mbps

- **user_network_autoNeg** - Turns on and off the automatic negotiation of the Ethernet connection speed.
- *Access*: read/write
- *Type*: uint32_t
- *Min value*: FALSE
- *Max value*: TRUE
- *Default value*: TRUE

- **user_network_ip** - The network address of the device.
- *Access*: read/write
- *Type*: u32_arr_t
- *Max value*: 255
- *Max elements*: 4,
- *Default value*: [192, 168, 1, 30]

- **user_network_mask** - Subnet mask for the device.
- *Access*: read/write
- *Type*: u32_arr_t
- *Max value*: 255
- *Max elements*: 4,

- *Default value*: [255, 255, 255, 0]

- **user_network_gateway** - Gateway address.
  - *Access*: read/write
  - *Type*: u32_arr_t
  - *Max value*: 255
  - *Max elements*: 4,
  - *Default value*: [192, 168, 1, 1]

- **user_network_hostIP** - The network address of the device to which profiles and calculation results are sent using the UDP protocol.
  - *Access*: read/write
  - *Type*: u32_arr_t
  - *Max value*: 255
  - *Max elements*: 4,
  - *Default value*: [192, 168, 1, 2]

- **user_network_hostPort** - The port number on the device to which profiles and calculation results are sent over the UDP protocol.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Max value*: 65535
  - *Default value*: 50001

- **user_network_webPort** - Port number to access the Web page.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Max value*: 65535
  - *Default value*: 80

- **user_network_servicePort** - Port number for service protocol.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Max value*: 65535
  - *Default value*: 50011

- **user_streams_udpEnabled** - Enabling and disabling the profile stream, transmitted via the UDP protocol (sending to the network address, set by the user_network_hostIP parameter and the port, set by the user_network_hostPort parameter).
  - *Access*: read/write

- *Type*: uint32_t
- *Min value*: FALSE
- *Max value*: TRUE
- *Default value*: FALSE

- **user_streams_format** - The format of the transmitted profiles.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: DATA_FORMAT_RAW_PROFILE
  - *Max value*: DATA_FORMAT_PROFILE
  - *Enum*:
    - DATA_FORMAT_RAW_PROFILE - the position of the points in the profile is transferred without applying calibration data, in subpixel values. Used for debugging and setting up the device, allows to compare the image, generated by the CMOS-sensor and the calculated profile position,
    - DATA_FORMAT_PROFILE - the position of the points in the profile is transmitted in discretes, the main format for the operation of the device
  - *Default value*: DATA_FORMAT_PROFILE

- **user_streams_pointsCount** - The number of points in the profile that the device calculates and transmits.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: 648
  - *Max value*: 1296
  - *Step*: 648
  - *Default value*: 648
  - *Units*: points

- **user_streams_includeIntensity** - Enable or disable the transfer of brightness points in the profile. The brightness values are transferred after the profile data in the format of 1 byte per point, 0 - black … 255 - white.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: FALSE
  - *Max value*: TRUE
  - *Default value*: FALSE

- **user_streams_udpPacketsCounter** - Internal counter of transmitted UDP packets with profiles. It can be used to control the

loss of packets with profiles.
- *Access*: read/write
- *Type*: uint32_t
- *Default value*: 0

- **user_processing_threshold** - Threshold of profile points detection. Smaller values of the parameter allow detect the profile at a lower brightness of the signal, which may cause false detections on flare and reflections. Higher parameter values require higher signal brightness, but provide confident detection of the profile position.
- *Access*: read/write
- *Type*: uint32_t
- *Max value*: 100
- *Default value*: 2
- *Units*: %

- **user_processing_profPerSec** - The number of processed profiles per second.
- *Access*: read
- *Type*: uint32_t
- *Max value*: 20000
- *Default value*: 490
- *Units*: pps

- **user_processing_medianMode** - Enable and width of median profile filtering. The median filter allows remove random outliers and fill the gaps in the profile with a width of up to half the size of the filter.
- *Access*: read/write
- *Type*: uint32_t
- *Max value*: 15
- *Enum*:
    - 0 - the filter is disabled,
    - 3 - the filter is enabled, filter size is 3 points,
    - 5 - the filter is enabled, filter size is 5 points,
    - 7 - the filter is enabled, filter size is 7 points,
    - 9 - the filter is enabled, filter size is 9 points,
    - 11 - the filter is enabled, filter size is 11 points,
    - 13 - the filter is enabled, filter size is 13 points,
    - 14 - the filter is enabled, filter size is 15 points,
- *Default value*: 0

- **user_processing_bilateralMode** - Bilateral filter allows smooth the values of the points of the profile, while maintaining its sharp changes.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Max value*: 15
  - *Enum*:
    - 0 - the filter is disabled,
    - 3 - the filter is enabled, filter size is 3 points,
    - 5 - the filter is enabled, filter size is 5 points,
    - 7 - the filter is enabled, filter size is 7 points,
    - 9 - the filter is enabled, filter size is 9 points,
    - 11 - the filter is enabled, filter size is 11 points,
    - 13 - the filter is enabled, filter size is 13 points,
    - 14 - the filter is enabled, filter size is 15 points,
  - *Default value*: 0

- **user_processing_peakMode** - Profile peak detection mode for position calculation. Used to ignore reflections and highlights.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: PEAK_MODE_INTENSITY
  - *Max value*: PEAK_MODE_NUMBER_4
  - *Enum*:
    - PEAK_MODE_INTENSITY - the position of the profile points is calculated at maximum intensity,
    - PEAK_MODE_FIRST - the position of the profile points is calculated from the first overstepping of the detection threshold,
    - PEAK_MODE_LAST - the position of the profile points is calculated from the last overstepping of the detection threshold,
    - PEAK_MODE_NUMBER_2 - when calculating the position of profile points, the advantage is given to peak #2,
    - PEAK_MODE_NUMBER_3 - when calculating the position of profile points, the advantage is given to peak #3,
    - PEAK_MODE_NUMBER_4 - when calculating the position of profile points, the advantage is given to peak #4
  - *Default value*: PEAK_MODE_INTENSITY

- **user_processing_flip** - Profile reflection mode. Reflection applies only if **user_streams_format** is set to DATA_FORMAT_PROFILE.
  - *Access*: read/write

- *Type*: uint32_t
- *Min value*: FLIP_MODE_OFF
- *Max value*: FLIP_MODE_XZ
- *Enum*:
    - FLIP_MODE_OFF - no reflections,
    - FLIP_MODE_X - reflection along the X axis,
    - FLIP_MODE_Z - reflection along the Z axis,
    - FLIP_MODE_XZ - reflection along the X and Z axis
- *Default value*: FLIP_MODE_OFF

- **user_laser_enabled** - Switching the laser radiation on and off.
    - *Access*: read/write
    - *Type*: uint32_t
    - *Min value*: FALSE
    - *Max value*: TRUE
    - *Default value*: FALSE

- **user_laser_value** - Sets the brightness of the laser radiation.
    - *Access*: read/write
    - *Type*: uint32_t
    - *Max value*: 100
    - *Step*: 5
    - *Default value*: 0
    - *Units*: %

- **user_trigger_sync_source** - Selection of inputs and their combinations for synchronization of measurements.
    - *Access*: read/write
    - *Type*: uint32_t
    - *Min value*: TRIG_SOURCE_IN1
    - *Max value*: TRIG_SOURCE_IN1_AND_IN2
    - *Enum*:
        - TRIG_SOURCE_IN1 - trigger measurements on an event at input 1,
        - TRIG_SOURCE_IN2 - trigger measurements on an event at input 2,
        - TRIG_SOURCE_IN1_OR_IN2 - trigger measurements on an event at input 1 **or** input 2,
        - TRIG_SOURCE_IN1_AND_IN2 - trigger measurements on an event at input 1 **and** input 2
    - *Default value*: TRIG_SOURCE_IN1

- **user_trigger_sync_strictEnabled** - Enable or disable strict

synchronization mode. When this mode is enabled, synchronization events that occurred during a frame exposure will be ignored and the next measurement will only be triggered by the synchronization event, when the sensor has finished exposing the previous frame. In this case, if the synchronization event rate is slightly higher than the maximum frame rate of the sensor, the number of profiles per second will be lower than the maximum frame rate due to the stroboscopic effect. If the mode is off and there were synchronization events during the exposure, the next measurement will start as soon as the sensor finishes exposing the previous frame. In any situation, the encoder value in the profile will be recorded at the middle of the frame exposure.
- *Access*: read/write
- *Type*: uint32_t
- *Min value*: FALSE
- *Max value*: TRUE
- *Default value*: FALSE

- **user_trigger_sync_divider** - The synchronization event divider.
Does not affect the encoder counter.
- *Access*: read/write
- *Type*: uint32_t
- *Min value*: 1
- *Max value*: 8192
- *Default value*: 1

- **user_trigger_sync_delay** - The value of the delay in the start of measurement (start of frame exposure) relative to the synchronization event.
- *Access*: read/write
- *Type*: uint32_t
- *Min value*: 700
- *Max value*: 100000000
- *Step*: 100
- *Default value*: 700
- *Units*: ns

- **user_trigger_sync_value** - The value of the internal measurement start counter. Shows the number of measurements taken.
- *Access*: read/write
- *Type*: uint32_t
- *Default value*: 0

- **user_trigger_counter_type** - Type of encoder counter (internal pulse counter) at synchronization inputs.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: TRIG_COUNTER_UNIDIR
  - *Max value*: TRIG_COUNTER_BIDIR
  - *Enum*:
    - TRIG_COUNTER_UNIDIR - unidirectional counter, that does not take into account the phase of the signals at inputs 1 and 2,
    - TRIG_COUNTER_BIDIR - bidirectional counter, that takes into account the phase of the signals at inputs 1 and 2, and can both increase and decrease
  - *Default value*: TRIG_COUNTER_UNIDIR

- **user_trigger_counter_maxValue** - The maximum value of the encoder counter, upon reaching which it is reset to the 0.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: 1
  - *Max value*: 4294967295
  - *Default value*: 4294967295

- **user_trigger_counteraser_resetTimerEnabled** - Enabling and disabling the timer for automatically resetting the encoder counter to 0. If the timer is enabled, then if no synchronization events during the time, specified by the **user_trigger_counter_resetTimerValue** parameter, the encoder counter will be reset to 0.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: FALSE
  - *Max value*: TRUE
  - *Default value*: FALSE

- **user_trigger_counter_resetTimerValue** - Timeout value until the encoder counter value is automatically reset to 0.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: 100
  - *Max value*: 4294967295
  - *Step*: 1000
  - *Default value*: 4294967295
  - *Units*: ns

- **user_trigger_counter_value** - Encoder counter value. This is an internal event counter at inputs 1 and 2.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Default value*: 0

- **user_trigger_counter_dir** - The ratio of the phases of the signals at inputs 1 and 2. Determines the direction of movement if using a movement system.
  - *Access*: read
  - *Type*: uint32_t
  - *Default value*: 0

- **user_input1_enabled** - Turning the input 1 on and off. If the input is turned off, then all signals will be ignored.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: FALSE
  - *Max value*: TRUE
  - *Default value*: FALSE

- **user_input1_mode** - Input 1 operation mode. Defines which signal change is a synchronization event for a given input.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: IN1_MODE_RISE_OR_FALL
  - *Max value*: IN1_MODE_LVL0
  - *Enum*:
    - IN1_MODE_RISE_OR_FALL - the synchronization event is both the transition from low to high state (edge of the pulse) and from high to low state (fall of the pulse),
    - IN1_MODE_RISE - the synchronization event is only the transition from low to high state (edge of the pulse),
    - IN1_MODE_FALL - the synchronization event is only the transition from high to low (fall of the pulse),
    - IN1_MODE_LVL1 - the synchronization event is a high level at the input, measures starts from the internal generator,
    - IN1_MODE_LVL0 - the synchronization event is a low level at the input, measures starts from the internal generator
  - *Default value*: IN1_MODE_RISE_OR_FALL

- **user_input2_enabled** - Turning the input 2 on and off. If the input is turned off, then all signals will be ignored.

- *Access*: read/write
- *Type*: uint32_t
- *Min value*: FALSE
- *Default value*: FALSE

- **user_input2_mode** - Input 2 operation mode. Defines which signal change is a synchronization event for a given input.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: IN2_MODE_RISE_OR_FALL
  - *Max value*: IN2_MODE_LVL0
  - *Enum*:
    - IN2_MODE_RISE_OR_FALL - the synchronization event is both the transition from low to high state (edge of the pulse) and from high to low state (fall of the pulse),
    - IN2_MODE_RISE - the synchronization event is only the transition from low to high state (edge of the pulse),
    - IN2_MODE_FALL - the synchronization event is only the transition from high to low (fall of the pulse),
    - IN2_MODE_LVL1 - the synchronization event is a high level at the input, measures starts from the internal generator,
    - IN2_MODE_LVL0 - the synchronization event is a low level at the input, measures starts from the internal generator
  - *Default value*: IN2_MODE_RISE_OR_FALL

- **user_input3_enabled** - Turning the input 3 on and off. If the input is turned off, then all signals will be ignored.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: FALSE
  - *Max value*: TRUE
  - *Default value*: FALSE

- **user_input3_mode** - Input 3 operation mode. This input is mainly used to reset the encoder counter value.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: IN3_MODE_RISE
  - *Max value*: IN3_MODE_FALL
  - *Enum*:
    - IN3_MODE_RISE - reset the encoder counter value at the rising edge of the pulse at the input,
    - IN3_MODE_FALL - reset the encoder counter value at the falling edge of the pulse at the input

- *Default value*: IN3_MODE_RISE

- **user_input1_samples** - An array of signal values at input 1.
  The parameter is a time scan of the signals at input 1. Every 2 bits indicate the state of the signal at a certain point in time. Value 0b00 - low signal level, 0b01 - state changed (pulses), 0b10 - reserved, 0b11 - high level.
  - *Access*: read/write
  - *Type*: u32_arr_t
  - *Max elements*: 6,
  - *Default value*: [0, 0, 0, 0, 0, 0]

- **user_input2_samples** - An array of signal values at input 2.
  The parameter is a time scan of the signals at input 2. Every 2 bits indicate the state of the signal at a certain point in time. Value 0b00 - low signal level, 0b01 - state changed (pulses), 0b10 - reserved, 0b11 - high level.
  - *Access*: read/write
  - *Type*: u32_arr_t
  - *Max elements*: 6,
  - *Default value*: [0, 0, 0, 0, 0, 0]

- **user_input3_samples** - An array of signal values at input 3.
  The parameter is a time scan of the signals at input 3. Every 2 bits indicate the state of the signal at a certain point in time. Value 0b00 - low signal level, 0b01 - state changed (pulses), 0b10 - reserved, 0b11 - high level.
  - *Access*: read/write
  - *Type*: u32_arr_t
  - *Max elements*: 6,
  - *Default value*: [0, 0, 0, 0, 0, 0]

- **user_output1_enabled** - Turning output 1 on and off. When turned off, the output is low. In the on state, the signal is set by the parameters **user_output1_mode** and **user_output1_pulseWidth**.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: FALSE
  - *Max value*: TRUE
  - *Default value*: FALSE

- **user_output1_mode** - Output 1 mode. Sets which signal will be output.

- *Access*: read/write
- *Type*: uint32_t
- *Min value*: OUT_MODE_EXPOSE_START
- *Max value*: OUT_MODE_IN3_REPEATER
- *Enum*:
    - OUT_MODE_EXPOSE_START - impulse at the moment the frame starts to be exposed for the next measurement,
    - OUT_MODE_EXPOSE_TIME - pulse during the exposure of the frame for the next measurement,
    - OUT_MODE_IN1_REPEATER - input 1 repeater, regardless of whether the input is on or off,
    - OUT_MODE_IN2_REPEATER - input 2 repeater, regardless of whether the input is on or off,
    - OUT_MODE_IN3_REPEATER - input 3 repeater, regardless of whether the input is on or off
- *Default value*: OUT_MODE_EXPOSE_START

- **user_output1_pulseWidth** - Pulse width when **user_output1_mode** parameter has value OUT_MODE_EXPOSE_START.
    - *Access*: read
    - *Type*: uint32_t
    - *Min value*: 10
    - *Max value*: 1000000
    - *Step*: 10
    - *Default value*: 1000
    - *Units*: ns

- **user_output2_enabled** - Turning output 2 on and off. When turned off, the output is low. In the on state, the signal is set by the parameters **user_output2_mode** and **user_output2_pulseWidth**.
    - *Access*: read/write
    - *Type*: uint32_t
    - *Min value*: FALSE
    - *Max value*: TRUE
    - *Default value*: FALSE

- **user_output2_mode** - Output 2 mode. Sets which signal will be output.
    - *Access*: read/write
    - *Type*: uint32_t
    - *Min value*: OUT_MODE_EXPOSE_START
    - *Max value*: OUT_MODE_IN3_REPEATER
    - *Enum*:
        - OUT_MODE_EXPOSE_START - impulse at the moment the frame

starts to be exposed for the next measurement,
- OUT_MODE_EXPOSE_TIME - pulse during the exposure of the frame for the next measurement,
- OUT_MODE_IN1_REPEATER - input 1 repeater, regardless of whether the input is on or off,
- OUT_MODE_IN2_REPEATER - input 2 repeater, regardless of whether the input is on or off,
- OUT_MODE_IN3_REPEATER - input 3 repeater, regardless of whether the input is on or off

- *Default value*: OUT_MODE_EXPOSE_START

- **user_output2_pulseWidth** - Pulse width when **user_output2_mode** parameter has value OUT_MODE_EXPOSE_START.
  - *Access*: read
  - *Type*: uint32_t
  - *Min value*: 10
  - *Max value*: 1000000
  - *Step*: 10
  - *Default value*: 1000
  - *Units*: ns

- **user_dump_enabled** - Enabling profile recording in the internal memory of the device - forming a dump. The recording will be stopped when the maximum dump capacity is reached, either when **user_dump_capacity** is reached or when FALSE is written to this parameter. Before starting the dump recording, user_trigger_sync_value and user_trigger_counter_value counters will be reset to 0.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: FALSE
  - *Max value*: TRUE
  - *Default value*: FALSE

- **user_dump_capacity** - User-defined number of profiles to be dumped. Upon reaching this value, the recording will automatically stop and the value of the **user_dump_enabled** parameter will become FALSE.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: 1
  - *Max value*: 80000
  - *Default value*: 80000
  - *Units*: profiles

- **user_dump_size** - The current number of profiles in the dump.
  Before starting dump recording, this value is reset to 0. During dump recording, this value increases.
  - *Access*: read
  - *Type*: uint32_t
  - *Max value*: 80000
  - *Default value*: 0
  - *Units*: profiles

- **user_dump_timeStamp** - The time stamp of the dump. Setted by the device when the dump recording starts.
  - *Access*: read
  - *Type*: uint64_t
  - *Default value*: 0
  - *Units*: ticks

- **user_dump_view3d_motionType** - Type of movement system on which the device is installed. The value of the parameter is used to correctly draw the dump as a 3D model.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: MOTION_TYPE_LINEAR
  - *Max value*: MOTION_TYPE_RADIAL
  - *Enum*:
    - MOTION_TYPE_LINEAR - linear motion system,
    - MOTION_TYPE_RADIAL - radial motion system
  - *Default value*: MOTION_TYPE_LINEAR

- **user_dump_view3d_ySource** - Source of the Y-axis coordinates. The value of the parameter is used to correctly draw the dump as a 3D model.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: Y_AXIS_SYSTEM_TIME
  - *Max value*: Y_AXIS_MEASURES_COUNTER
  - *Enum*:
    - Y_AXIS_SYSTEM_TIME - internal device timer,
    - Y_AXIS_STEP_COUNTER - parameter **user_trigger_counter_value**,
    - Y_AXIS_MEASURES_COUNTER - measurements counter
  - *Default value*: Y_AXIS_SYSTEM_TIME

- **user_dump_view3d_yStep** - The value of a single step in the

Y-axis.
- *Access*: read/write
- *Type*: double_t
- *Max value*: 10000
- *Default value*: 0.0005
- *Units*: mm

- **user_dump_view3d_paintMode** - 3D model coloring mode. This parameter is used when drawing a 3D model in the WEB-interface.
- *Access*: read/write
- *Type*: uint32_t
- *Min value*: PAINT_MODE_HEIGHTMAP
- *Max value*: PAINT_MODE_INTENSITY
- *Enum*:
  - PAINT_MODE_HEIGHTMAP - coloring according to the height map,
  - PAINT_MODE_INTENSITY - intensity mapping, parameter **user_streams_includeIntensity** must be set to TRUE,
- *Default value*: PAINT_MODE_HEIGHTMAP

- **user_dump_view3d_decimation** - Profiles decimation when drawing a 3D model. This parameter is used when drawing a 3D model in the WEB-interface.
- *Access*: read/write
- *Type*: uint32_t
- *Min value*: DUMP_VIEW3D_DECIM_1
- *Max value*: DUMP_VIEW3D_DECIM_200
- *Enum*:
  - DUMP_VIEW3D_DECIM_1 - all dump profiles are displayed,
  - DUMP_VIEW3D_DECIM_2 - step to display dump profiles 2,
  - DUMP_VIEW3D_DECIM_5 - step to display dump profiles 5,
  - DUMP_VIEW3D_DECIM_10 - step to display dump profiles 10,
  - DUMP_VIEW3D_DECIM_20 - step to display dump profiles 20,
  - DUMP_VIEW3D_DECIM_50 - step to display dump profiles 50,
  - DUMP_VIEW3D_DECIM_100 - step to display dump profiles 100,
  - DUMP_VIEW3D_DECIM_200 - step to display dump profiles 200,
- *Default value*: DUMP_VIEW3D_DECIM_1

- **user_eip_tcpPort** - The port number that the device listens for incoming TCP connections via EthernetIP.
- *Access*: read/write
- *Type*: uint32_t
- *Max value*: 65535
- *Default value*: 44818

- **user_eip_udpPort** - The port number that the device listens for
  UDP packets with EthernetIP data.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Max value*: 65535
  - *Default value*: 2222

- **user_compatibility_rf625Enabled** - Enable or disable
  compatibility mode with the obsolete RF625 scanner. When enabling
  compatibility mode, UDP profile stream will be suspended.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Min value*: FALSE
  - *Max value*: TRUE
  - *Default value*: FALSE

- **user_compatibility_rf625TCPPort** - Port number for incoming TCP
  connections via RF625 protocol.
  - *Access*: read/write
  - *Type*: uint32_t
  - *Max value*: 65535
  - *Default value*: 620

Factory

- **fact_general_firmwareVer** - Device firmware version [Major,
  Minor, Patch].
  - *Access*: read
  - *Type*: u32_arr_t
  - *Max elements*: 3,
  - *Default value*: [1, 0, 0]

- **fact_general_hardwareVer** - Device hardware version.
  - *Access*: read
  - *Type*: uint32_t
  - *Default value*: 403051520

- **fact_general_deviceType** - Device type identifier.
  - *Access*: read/awrite
  - *Type*: uint32_t
  - *Max value*: 65535
  - *Default value*: 627

- **fact_general_serial** - Device serial number.
    - *Access*: read/awrite
    - *Type*: uint32_t
    - *Default value*: 0

- **fact_general_pcbSerial** - Device PCB serial number.
    - *Access*: read/awrite
    - *Type*: uint32_t
    - *Default value*: 0

- **fact_general_lifeTime** - Total device runtime in UNIX format.
    - *Access*: read/awrite
    - *Type*: uint32_t
    - *Max value*: 1577846300
    - *Default value*: 0
    - *Units*: s

- **fact_general_workTime** - Device uptime in UNIX format.
    - *Access*: read/awrite
    - *Type*: uint32_t
    - *Max value*: 1577846300
    - *Default value*: 0
    - *Units*: s

- **fact_general_startsCount** - Total number of device starts.
    - *Access*: read/awrite
    - *Type*: uint32_t
    - *Max value*: 8760
    - *Default value*: 0
    - *Units*: times

- **fact_general_customerID** - Device customer identifier. The identifier of the company that purchased / ordered the device.
    - *Access*: read/awrite
    - *Type*: uint32_t
    - *Default value*: 0
    - *Units*: id

- **fact_general_fpgaFreq** - FPGA project clock frequency for this device.
    - *Access*: read/awrite
    - *Type*: uint32_t

- *Min value*: 10000000
- *Max value*: 500000000
- *Default value*: 10000000
- *Units*: Hz

- **fact_general_smr** - Start of measuring range in Z axis in mm.
  - *Access*: read/awrite
  - *Type*: uint32_t
  - *Max value*: 10000
  - *Default value*: 80
  - *Units*: mm

- **fact_general_mr** - Size of the measuring range in Z axis in mm.
  - *Access*: read/awrite
  - *Type*: uint32_t
  - *Max value*: 10000
  - *Default value*: 130
  - *Units*: mm

- **fact_general_xsmr** - The size along the X axis of the measuring range at the beginning of the range.
  - *Access*: read/awrite
  - *Type*: uint32_t
  - *Max value*: 10000
  - *Default value*: 40
  - *Units*: mm

- **fact_general_xemr** - The size along the X axis of the measuring range at the end of the range.
  - *Access*: read/awrite
  - *Type*: uint32_t
  - *Max value*: 10000
  - *Default value*: 40
  - *Units*: mm

- **fact_general_pixDivider** - Divider to obtain the subpixel position of profile points in the uncalibrated data transfer mode (parameter **user_streams_format** is set to DATA_FORMAT_RAW_PROFILE).
  - *Access*: read
  - *Type*: uint32_t
  - *Max value*: 65535
  - *Default value*: 32

- **fact_general_profDivider** - Divider to obtain the subpixel
  position of profile points in the calibrated data transfer mode
  (parameter **user_streams_format** is set to
  DATA_FORMAT_PROFILE).
  - *Access*: read
  - *Type*: uint32_t
  - *Max value*: 65535
  - *Default value*: 16384

- **fact_general_oemDevName** - Device name assigned by the OEM
  customer.
  - *Access*: read/awrite
  - *Type*: string_t
  - *Max len*: 128
  - *Default value*: “Laser scanner”

- **fact_general_authStatus** - Authorization status for changing the
  factory settings of the device.
  - *Access*: read
  - *Type*: uint32_t
  - *Min value*: AUTH_STATUS_USER
  - *Max value*: AUTH_STATUS_FACTORY
  - *Enum*:
    - AUTH_STATUS_USER - authorized as a user, factory settings
      cannot be changed,
    - AUTH_STATUS_FACTORY - authorized as a manufacturer, factory
      settings can be changed
  - *Default value*: AUTH_STATUS_USER

- **fact_sensor_name** - Name of the sensor used in the device.
  - *Access*: read/awrite
  - *Type*: string_t
  - *Max len*: 64
  - *Default value*: “TYPE 1”

- **fact_sensor_width** - Number of pixels in the CMOS sensor.
  - *Access*: read/awrite
  - *Type*: uint32_t
  - *Min value*: 648
  - *Max value*: 648
  - *Default value*: 648
  - *Units*: pixels

- **fact_sensor_height** - Number of lines in the CMOS sensor.
    - *Access*: read/awrite
    - *Type*: uint32_t
    - *Min value*: 488
    - *Max value*: 488
    - *Default value*: 488
    - *Units*: lines

- **fact_sensor_pixFreq** - Pixel frequency for installed CMOS sensor.
    - *Access*: read/awrite
    - *Type*: uint32_t
    - *Min value*: 1000000
    - *Max value*: 500000000
    - *Default value*: 40000000
    - *Units*: Hz

- **fact_sensor_frmConstPart** - Constant part of the frame cycle.
    - *Access*: read/awrite
    - *Type*: uint32_t
    - *Min value*: 200
    - *Max value*: 200000
    - *Default value*: 3500
    - *Units*: ticks

- **fact_sensor_frmPerLinePart** - Frame cycle part for each line.
    - *Access*: read/awrite
    - *Type*: uint32_t
    - *Min value*: 10
    - *Max value*: 100000
    - *Default value*: 160
    - *Units*: ticks

- **fact_sensor_minExposure** - Minimum allowable exposure value.
    - *Access*: read/awrite
    - *Type*: uint32_t
    - *Max value*: 100000000
    - *Step*: 10
    - *Default value*: 3000
    - *Units*: ns

- **fact_sensor_maxExposure** - Maximum allowable exposure value.
    - *Access*: read/awrite

- *Type*: uint32_t
- *Max value*: 300000000
- *Step*: 10
- *Default value*: 300000000
- *Units*: ns

- **fact_sensor_imgFlip** - Image reflection mode. Applies directly to the image transmitted, by the CMOS sensor.
  - *Access*: read/awrite
  - *Type*: uint32_t
  - *Min value*: FLIP_MODE_OFF
  - *Max value*: FLIP_MODE_XZ
  - *Enum*:
    - FLIP_MODE_OFF - no reflections,
    - FLIP_MODE_X - reflection along the X axis,
    - FLIP_MODE_Z - reflection along the Z axis,
    - FLIP_MODE_XZ - reflection along the X and Z axis
  - *Default value*: FLIP_MODE_OFF

- **fact_sensor_analogGain** - CMOS sensor analog gain value.
  - *Access*: read/awrite
  - *Type*: uint32_t
  - *Max value*: 7
  - *Default value*: 5

- **fact_sensor_digitalGain** - CMOS sensor digital gain value.
  - *Access*: read/awrite
  - *Type*: uint32_t
  - *Max value*: 55
  - *Default value*: 48

- **fact_sensor_blackOdd** - Black level for odd lines.
  - *Access*: read/awrite
  - *Type*: uint32_t
  - *Max value*: 65535
  - *Default value*: 2300

- **fact_sensor_blackEven** - Black level for even lines.
  - *Access*: read/awrite
  - *Type*: uint32_t
  - *Max value*: 65535
  - *Default value*: 2400

- **fact_network_initRegs** - CMOS sensor registers values [regAddr, regValue …].
  - *Access*: read/awrite
  - *Type*: u32_arr_t
  - *Max value*: 255
  - *Max elements*: 64,
  - *Default value*: [41, 1, 83, 155, 58, 20, 59, 0, 60, 11, 69, 9, 80, 4, 97, 0, 98, 12, 101, 98, 102, 34, 103, 64, 106, 90, 107, 110, 108, 91, 109, 82, 110, 80, 117, 91]

- **fact_network_macAddr** - Physical address of the device.
  - *Access*: read/awrite
  - *Type*: u32_arr_t
  - *Max value*: 255
  - *Max elements*: 6,
  - *Default value*: [0x00, 0x0A, 0x35, 0x01, 0x02, 0x03]

- **fact_network_forceAutoNegTime** - The time after which the auto-negotiation of the Ethernet connection will be forced if the connection is not established.
  - *Access*: read/awrite
  - *Type*: uint32_t
  - *Max value*: 255
  - *Default value*: 5
  - *Units*: s

- **fact_network_webSockServicePort** - Port number for the service data transmission WEB-socket. Used by the Web-page.
  - *Access*: read/awrite
  - *Type*: uint32_t
  - *Min value*: 16384
  - *Max value*: 65535
  - *Default value*: 50002

- **fact_network_webSockDataPort** - Port number for the large data transmission WEB-socket. Used by the Web-page.
  - *Access*: read/awrite
  - *Type*: uint32_t
  - *Min value*: 16384
  - *Max value*: 65535
  - *Default value*: 50003

- **fact_network_webSockMathPort** - Port number for the math data

transmission WEB-socket. Used by the Web-page.
- *Access*: read/awrite
- *Type*: uint32_t
- *Min value*: 16384
- *Max value*: 65535
- *Default value*: 50004

- **fact_laser_waveLength** - The wavelength of the laser, installed in the device.
- *Access*: read/awrite
- *Type*: uint32_t
- *Max value*: 10000
- *Default value*: 650
- *Units*: nm

- **fact_laser_minValue** - Minimum DAC value. At this value, the laser stops emitting light.
- *Access*: read/awrite
- *Type*: uint32_t
- *Max value*: 4095
- *Default value*: 0

- **fact_laser_maxValue** - Maximum DAC value. At this value, the laser starts to emit light with maximum power.
- *Access*: read/awrite
- *Type*: uint32_t
- *Max value*: 4095
- *Default value*: 4095

- **fact_eip_identity_vendorID** - Identification number for the manufacturer of an EtherNet/IP device.
- *Access*: read
- *Type*: uint32_t
- *Default value*: 1588

- **fact_eip_identity_deviceType** - The list of device types is managed by ODVA and CI. It is used to identify the device profile that a particular product is using.
- *Access*: read
- *Type*: uint32_t
- *Max value*: 65535
- *Default value*: 0x2B

- **fact_eip_identity_productCode** - Product identifier according
  to developer documentation.
  - *Access*: read
  - *Type*: uint32_t
  - *Default value*: 627

- **fact_eip_identity_rev** - The Revision attribute, which consists
  of major and minor revisions, identifies the revision of the item the
  Identity Object is representing.
  - *Access*: read
  - *Type*: u32_arr_t
  - *Max value*: 255
  - *Max elements*: 2,
  - *Default value*: [1, 0]

- **fact_eip_identity_status** - Represents the current status of
  the entire device. Its value changes as the state of the device
  changes.
  - *Access*: read
  - *Type*: uint32_t
  - *Enum*:
    - Owned - the device (or an object within the device) has an
      owner,
    - Configured - the application of the device has been configured
      to do something different than the “out–of–box” default,
    - Minor Recoverable Fault - the device detected a problem with
      itself, which is thought to be recoverable. The problem does
      not cause the device to go into one of the faulted states,
    - Minor Unrecoverable Fault- the device detected a problem with
      itself, which is thought to be unrecoverable. The problem does
      not cause the device to go into one of the faulted states,
    - Major Recoverable Fault - the device detected a problem with
      itself, which caused the device to go into the “Major
      Recoverable Fault” state,
    - Major Unrecoverable Fault - the device detected a problem with
      itself, which caused the device to go into the “Major
      Unrecoverable Fault” state
  - *Default value*: 0

- **fact_eip_tcpIntrf_status** - is a bitmap that shall indicate the
  status of the TCP/IP network interface.
  - *Access*: read
  - *Type*: uint32_t
  - *Enum*:

- No configured - The Interface Configuration attribute has not
  been configured,
- Configured - The Interface Configuration attribute contains
  configuration obtained from BOOTP, DHCP or nonvolatile storage,
- Hardware Configured - The IP address member of the Interface
  Configuration attribute contains configuration, obtained from
  hardware settings,
- Mcast Pending - Indicates a pending configuration change in the
  TTL Value and/or Mcast Config attributes,
- Interface Configuration Pending - Indicates a pending
  configuration change in the Interface Configuration attribute,
- Address Conflict Detection Status - Indicates when an IP
  address conflict has been detected by ACD

- *Default value*: 0

- **fact_eip_tcpIntrf_capability** - is a bitmap that indicates the
device's support for optional network configuration capability.
- *Access*: read
- *Type*: uint32_t
- *Enum*:
  - BOOTP Client - the device is capable of obtaining its network
    configuration via BOOTP,
  - DNS Client - the device is capable of resolving host names by
    querying a DNS server,
  - DHCP Client - the device is capable of obtaining its network
    configuration via DHCP,
  - Configuration Settable - the Interface Configuration attribute
    is settable,
  - Hardware Configurable - the IP Address member of the Interface
    Configuration attribute can be obtained from hardware settings
    (e.g., pushwheel, thumbwheel, etc.),
  - Interface Configuration Change Requires Reset - the device
    requires a restart in order for a change to the Interface
    Configuration attribute to take effect,
  - Address Conflict Detection Capable - the device is capable of
    ACD
- *Default value*: 0x14

- **fact_eip_tcpIntrf_control** - is a bitmap used to control
network configuration options.
- *Access*: read
- *Type*: uint32_t
- *Enum*:
  - Static ip mode - The device shall use statically-assigned IP

configuration values,
- BOOTP mode - The device shall obtain its interface
configuration values via BOOTP,
- DHCP mode - The device shall obtain its interface configuration
values via DHCP
- DNS Enable - the device shall resolve host names by querying a
DNS server
- *Default value*: 0

- **fact_eip_tcpIntrf_phyLink** - identifies the object associated
with the underlying physical communications interface (e.g., an 802.3
interface).
- *Access*: read
- *Type*: u32_arr_t
- *Max value*: 255
- *Max elements*: 6
- *Default value*: [0x20, 0xF6, 0x24, 0x01]

- **fact_eip_tcpIntrf_inactTimeout** - is used to enable TCP socket
cleanup (closing) when the defined number of seconds have elapsed
with no Encapsulation activity.
- *Access*: read/awrite
- *Type*: uint32_t
- *Max value*: 255
- *Default value*: 120

- **fact_smart_enabled** - Turn on and off the capabilities of a
smart device.
- *Access*: read/awrite
- *Type*: uint32_t
- *Min value*: FALSE
- *Max value*: TRUE
- *Default value*: FALSE

1.4.2 WEB API v1

Using the easy-to-use WEB API, the user can get information about the device, read or write the value of the parameter. Also, through the WEB API, the device can execute some commands. A complete list of commands supported through this access is given in the description of the commands. The WEB API examples use the factory IP address of the device and presented as they should be typed in the address bar of the browser. If it has been changed by the user, the IP address of the device should be used.

Quick device info

- **/hello** - Getting general information about the device in JSON format.
  - *GET*:
    - 192.168.1.30/hello

- **/api/v1/config/commands** - Getting the list of commands, supported by the device. The formalized description will contain the command name, WEB API access capability, command identifier and access mode.
  - *GET*:
    - 192.168.1.30/api/v1/config/commands

- **/api/v1/config/returnCodes** - Getting a text description of the codes of operation results and errors, returned by the device.
  - *GET*:
    - 192.168.1.30/api/v1/config/returnCodes

Device parameters

- **/api/v1/config/params** - Getting general information about all device parameters in JSON format. The formalized description of the parameter will contain its name, type, access mode, index in the parameter array, offset for binary data, parameter data size, current value, default value, minimum and maximum values, parameter value step, for arrays - the maximum number of elements.
  - *GET*:
    - 192.168.1.30/api/v1/config/params

- **/api/v1/config/params/values** - Reading and writing values of the device parameters. For reading it is possible to request specific parameters by name or index. To write a parameter, it is necessary to form a “PUT” request with the parameters “parameter_name:value”.
  - *GET*:
    - 192.168.1.30/api/v1/config/params/values
    - 192.168.1.30/api/v1/config/params/values?name=fact_general_hardwareVer&index=120
  - *PUT*:
    - 192.168.1.30/api/v1/config/params/values?user_sensor_framerate=100&user_sensor_exposure1

- **/api/v1/sensor** - Reading and writing CMOS-sensor registers.
  - *GET*:

- 192.168.1.30/api/v1/sensor?reg=0x5B&val=0x003F
- 192.168.1.30/api/v1/sensor?index=0&reg=0x5B&val=0x003F

- *PUT*:
  - 192.168.1.30/api/v1/sensor?reg=0x5B&val=0x003F
  - 192.168.1.30/api/v1/sensor?index=0&reg=0x5B&val=0x003F

Save, restore and reboot

- **/api/v1/config/params/save** - Saving the current values of the device parameters in non-volatile memory in user area. Saved values will be used when the device is switched on again.
  - *GET*:
    - 192.168.1.30/api/v1/config/params/save

- **/api/v1/config/params/restore/save** - Saving the current values of the device parameters in the recovery area. These parameters will be applied when parameters from the user area are damaged.
  - *GET*:
    - 192.168.1.30/api/v1/config/params/restore/save

- **/api/v1/config/params/restore/load** - Loading device parameter values from the recovery area. The loaded values will be written to the user area, the device will be automatically rebooted.
  - *GET*:
    - 192.168.1.30/api/v1/config/params/restore/load

- **/api/v1/reboot** - Reboot the device. The parameters will be loaded from the user area (if they are not damaged).
  - *GET*:
    - 192.168.1.30/api/v1/reboot

Log

- **/api/v1/log** - Getting a log of the device with full description of records.
  - *GET*:
    - 192.168.1.30/api/v1/log

- **/api/v1/log/content** - Getting the device log in an abbreviated form - is easier to read.
  - *GET*:
    - 192.168.1.30/api/v1/log/content

Authorization

- **/api/v1/authorization** - Authorization on the device as a manufacturer - allows editing factory parameters of the device. Using the “GET” request, get a token for which generate a key and send to the device in the “PUT” request.
  - *GET*:
    - 192.168.1.30/api/v1/authorization
  - *PUT*:
    - 192.168.1.30/api/v1/authorization?key=230d84e16c0dae529098f1f1bb…..

1.4.3 COMMANDS

The commands transmitted to the device are intended for searching devices in the network, reading and setting parameters, downloading service data, firmware upgrade, receiving frames generated by CMOS-sensor and other functions. The commands and their answers are given in the service protocol (in the current revision, RF627 protocol). The service protocol uses UDP packets sent to the device’s network address (parameter **user_network_ip**) and the service port (parameter **user_network_servicePort**).

General device commands

- **HELLO_JSON_REQUEST** - Search for devices on the network. In answer to the command, JSON will be sent with a description of the main parameters of the device.
  - *URI*: /hello
  - *CID*: 0x0010
  - *Access*: unlocked
  - *Command payload*: no
  - *Answer payload*: JSON

- **PARAMS_DESCRIPTION_REQUEST** - Getting general information about all device parameters in JSON format. The formalized description of the parameter will contain its name, type, access mode, index in the parameter array, offset for binary data, parameter data size, current value, default value, minimum and maximum values, parameter value step, for arrays - the maximum number of elements.
  - *URI*: /api/v1/config/params
  - *CID*: 0x0110
  - *Access*: unlocked
  - *Command payload*: no
  - *Answer payload*: JSON

- **COMMANDS_DESCRIPTION_REQUEST** - Getting the list of commands, supported by the device. The formalized description will contain the

command name, WEB API access capability, command identifier and access mode.
- *URI*: /api/v1/config/commands
- *CID*: 0x0210
- *Access*: unlocked
- *Command payload*: no
- *Answer payload*: JSON

- **PARAMS_VALUES_JSON_REQUEST** - Reading values of the device parameters. For reading it is possible to request specific parameters by name or index.
- *URI*: /api/v1/config/params/values
- *CID*: 0x0310
- *Access*: unlocked
- *Command payload*: JSON [name:XXXX, name:XXXX, index:XXXX…]
- *Answer payload*: JSON [name:value, name:value, name:value…]

- **PARAMS_VALUES_JSON_WRITE** - Writing values of the device parameters, it is necessary to send the parameters in form of pair “parameter_name:value”.
- *CID*: 0x1010
- *Access*: unlocked
- *Command payload*: JSON [name:value, name:value, index:value…]
- *Answer payload*: JSON [name:OK, name:OK, name:OK…]

- **PARAMS_VALUES_BIN_REQUEST** - Reading parameter values in binary form. Each parameter will be stacked according to its index and size.
- *CID*: 0x0410
- *Access*: unlocked
- *Command payload*: no
- *Answer payload*: BIN

- **RETURN_CODES_JSON_REQUEST** - Getting a text description of the codes of operation results and errors, returned by the device.
- *URI*: /api/v1/config/returnCodes
- *CID*: 0x2010
- *Access*: unlocked
- *Command payload*: no
- *Answer payload*: JSON

- **PARAMS_SAVE** - Saving the current values of the device parameters in non-volatile memory in user area. Saved values will be used when the device is switched on again.

- *URI*: /api/v1/config/params/save
- *CID*: 0x0510
- *Access*: unlocked
- *Command payload*: no
- *Answer payload*: JSON [result:OK]

- **PARAMS_RESTORE_SAVE** - Saving the current values of the device parameters in the recovery area. These parameters will be applied when parameters from the user area are damaged.
- *URI*: /api/v1/config/params/restore/save
- *CID*: 0x0610
- *Access*: unlocked
- *Command payload*: no
- *Answer payload*: JSON [result:OK]

- **PARAMS_RESTORE_LOAD** - Loading device parameter values from the recovery area. The loaded values will be written to the user area, the device will be automatically rebooted.
- *URI*: /api/v1/config/params/restore/load
- *CID*: 0x0710
- *Access*: unlocked
- *Command payload*: no
- *Answer payload*: JSON [result:OK]

- **AUTHORIZATION_REQUEST** - Authorization on the device as a manufacturer - allows editing factory parameters of the device.
- *URI*: /api/v1/authorization
- *CID*: 0x2110
- *Access*: unlocked
- *Command payload*: no/key
- *Answer payload*: JSON

Calibration file

- **CALIB_FILE_DATA_WRITE** - Writing a fragment of a calibration file into a device.
- *CID*: 0x1052
- *Access*: unlocked
- *Command payload*: BIN (uint32_t: offset; uint8_t: data[])
- *Answer payload*: no

- **CALIB_FILE_CRC16_REQUEST** - Getting the checksum of the calibration file, uploaded to the device.

- *CID*: 0x1252
- *Access*: unlocked
- *Command payload*: no
- *Answer payload*: BIN (uint16_t: CRC)

- **CALIB_FILE_SAVE** - Saving the calibration file in a non-volatile memory of the device.
  - *CID*: 0x2052
  - *Access*: unlocked
  - *Command payload*: no
  - *Answer payload*: no

Profiles request

- **PROFILE_CAPTURE** - Command to start measurement. It is used only in the software measurement start mode (parameter **user_sensor_syncSource** = SYNC_SOFTWARE). When the command is received, the device starts the cycle of measurement, after that, the profile is calculated and a standard package with the profile is sent.
  - *CID*: 0x0459
  - *Access*: unlocked
  - *Command payload*: BIN (uint32_t: count (max: 16777215))
  - *Answer payload*: no

- **PROFILE_REQUEST** - The command to read the last calculated profile. The profile will be transferred in the payload of the service protocol message.
  - *CID*: 0x0559
  - *Access*: unlocked
  - *Command payload*: no
  - *Answer payload*: BIN

Dump request

- **DUMP_CONTENT_REQUEST** - Request the contents of the profile dump.
  - *CID*: 0x2259
  - *Access*: unlocked
  - *Command payload*: BIN (uint32_t: index; uint32_t: count)
  - *Answer payload*: BIN

Frame request

- **FRAME_REQUEST** - Request one frame of the image, exposed by the CMOS sensor.
  - *CID*: 0x1083
  - *Access*: unlocked
  - *Command payload*: BIN (uint32_t: index; uint32_t: count)
  - *Answer payload*: BIN (uint32_t: offset; uint8_t: data[])

Log request

- **LOG_PART_REQUEST** - Request a part of the device log file with a full description of the entries.
  - *URI*: /api/v1/log
  - *CID*: 0x0357
  - *Access*: unlocked
  - *Command payload*: JSON {index: XXX, count: XXX}
  - *Answer payload*: JSON

- **LOG_CONTENT_REQUEST** - Request the device log in an abbreviated form - is easier to read.
  - *URI*: /api/v1/log/content
  - *CID*: 0x0457
  - *Access*: unlocked
  - *Command payload*: no
  - *Answer payload*: JSON

Internal non-volatile memory

- **FLASH_ERASE** - Cleaning of the internal non-volatile memory of the device (execution of the command may lead to inoperability of the device). The command arguments are the start address of the erase area and the size of the erase area. The address must be aligned to 65536 bytes and the size is a multiple of 65536 bytes.
  - *CID*: 0x005A
  - *Access*: locked
  - *Command payload*: BIN (uint32_t: addr; uint32_t: size}
  - *Answer payload*: no

- **FLASH_FIRMWARE_READ** - Reading the firmware of the device.
  - *CID*: 0x1A5A
  - *Access*: unlocked
  - *Command payload*: no
  - *Answer payload*: BIN (uint32_t: offset; uint8_t: data[])

- **FLASH_FIRMWARE_WRITE** - Write device firmware.
  - *CID*: 0x205A
  - *Access*: unlocked
  - *Command payload*: BIN (uint32_t: offset; uint8_t: data[])
  - *Answer payload*: no

- **FLASH_FIRMWARE_CRC16_REQUEST** - Request the checksum of the firmware that has been uploaded to the device. The request must be made before writing the firmware to the internal non-volatile memory of the device.
  - *CID*: 0x215A
  - *Access*: unlocked
  - *Command payload*: no
  - *Answer payload*: BIN (uint16_t: CRC)

- **FLASH_FIRMWARE_SAVE** - Saving the loaded firmware to the internal non-volatile memory of the device. A checksum (command **FLASH_FIRMWARE_CRC16_REQUEST**) must be requested before saving.
  - *CID*: 0x225A
  - *Access*: unlocked
  - *Command payload*: no
  - *Answer payload*: no

Device to Web-page notification

- **FLASH_FIRMWARE_SAVE** - With this command, the device notifies the Web-page of various internal events: status changes, warnings, errors.
  - *CID*: 0x1063
  - *Access*: locked
  - *Command payload*: JSON {time: XXX, type: (NTF_INFO/NTF_WARN/NTF_ERR), message: TEXT}
  - *Answer payload*: no

Periphery commands

- **PERIPHERY_TRANSFER** - Transfer of data to and from connected peripheral devices.
  - *CID*: 0x1080
  - *Access*: unlocked
  - *Command payload*: BIN
  - *Answer payload*: BIN

CHAPTER

TWO

API DESCRIPTIONS

2.1 Core API in C

RF62X-CORE - the main library («Core») with a basic set of functions and types for working with laser scanners of the RF62X series. The library is written in the C programming language in accordance with the C99 standard (ISO / IEC 9899: 1999) and is cross-platform. To use this library, it is necessary to implement platform-dependent functions (working with memory, working with the network, input/output functions).

To download the library, see :ref: *the latest releases of the «Core» in C <rf62x_core_last_release>*.
To compile the library, see :ref: *compile and run the «Core» in C <compilation_rf62x_core>*.

2.1.1 «Core» initialization

If you want to use the RF62X-CORE library instead of the provided «wrapper»-libraries, the developer needs to independently implement the platform-dependent part of the «Core» (see :ref: *compile and run the «Core» <compilation_rf62x_core>*).

The file `rf62X_core.h` is a header file with a description of the functions for launching the «Core». This file contains definitions of the main functions used to initialize it:

init_platform_dependent_methods()

Initialization function of the platform-dependent part of the «Core»

void **`init_platform_dependent_methods`**(*memory_platform_dependent_methods_t *memory_methods*, *iostream_platform_dependent_methods_t *iostream_methods*, *network_platform_dependent_methods_t *network_methods*, *network_platform_dependent_settings_t *adapter_settings*)

init_platform_dependent_methods - Init platform dependent methods and settings

Parameters

- `memory_methods`: Structure with platform-specific methods for work with memory

- `iostream_methods`: Structure with platform-specific methods for work with iostream
- `network_methods`: Structure with platform-specific methods for work with network
- `adapter_settings`: Structure with platform-specific settings

core_version()

Function to get the current «Core» version

rfChar ***core_version**()
core_version - Return rf627 sdk version.

Return ptr to rfChar

2.1.2 Software Interface Overview

The file `rf62X_sdk.h` is the main file of the «Core» Application Programming Interface (API) and defines its functionality. `rf62X_sdk.h` contains the following set of basic functions for development:

set_platform_adapter_settings()

Function for transferring the current adapter settings to the «Core». This function is used if there have been any changes to the settings in the network adapter used by the «Core».

void **set_platform_adapter_settings**(rfUint32 *subnet_mask*, rfUint32 *host_ip_addr*)
change_platform_adapter_settings - change adapter's settings

Parameters

- `[in] subnet_mask`: Subnet mask on your local machine. A subnet mask is a number that defines a range of IP addresses that can be used in a network.
- `[in] host_ip_addr`: IP address of your network adapter(card)

Usage example

```
// Create value for scanners vector's type
vector_t* scanners = (vector_t*)calloc(1, sizeof (vector_t));

//Initialization vector
vector_init(&scanners);


// Iterate over all available network adapters in the current operating
// system to send "Hello" requests.
for (int i=0; i<GetAdaptersCount(); i++)
{
   // get another IP Addr and set this changes in network adapter settings.
   uint32_t host_ip_addr = ntohl(inet_addr(GetAdapterAddress(i)));
```

(continues on next page)

(continued from previous page)

```
    uint32_t host_mask = ntohl(inet_addr("255.255.255.0"));
    // call the function to change adapter settings inside the library.
    set_platform_adapter_settings(host_mask, host_ip_addr);

    // Search for RF627Old devices over network by Service Protocol.
    search_scanners(scanners, kRF627_OLD, kSERVICE);
}
```

search_scanners()

Function for searching RF62X devices by network

rfUint8 **search_scanners**(vector_t **list*, scanner_types_t *model*, protocol_types_t *protocol*)

search - Search for RF62X devices over network

Return 0 on success

Parameters

- `list`: - ptr to list of rf627 objects. If not null list will be filled with found devices
- `model`: - scanner's type (RF627-old, RF627-smart)
- `protocol`: - protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example

```
// Create value for scanners vector's type
vector_t* scanners = (vector_t*)calloc(1, sizeof (vector_t));
//Initialization vector
vector_init(&scanners);

// set IP Addr and NetMask for setting in network adapter settings.
uint32_t host_ip_addr = ntohl(inet_addr("192.168.1.2"));
uint32_t host_mask = ntohl(inet_addr("255.255.255.0"));
// call the function to change adapter settings inside the library.
set_platform_adapter_settings(host_mask, host_ip_addr);

// Search for RF627Old devices over network by Service Protocol.
search_scanners(scanners, kRF627_OLD, kSERVICE);
```

get_info_about_scanner()

Function to get scanner information from Hello-packet

hello_information **get_info_about_scanner**(scanner_base_t **device*, protocol_types_t *protocol*)

get_hello_info_of_scanners - Get information about scanner from hello packet

Return 0 on success

Parameters

- `device`: - prt to scanner

- `protocol`: - protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example

```
// Create value for scanners vector's type
vector_t* scanners = (vector_t*)calloc(1, sizeof (vector_t));
//Initialization vector
vector_init(&scanners);

// set IP Addr and NetMask for setting in network adapter settings.
uint32_t host_ip_addr = ntohl(inet_addr("192.168.1.2"));
uint32_t host_mask = ntohl(inet_addr("255.255.255.0"));
// call the function to change adapter settings inside the library.
set_platform_adapter_settings(host_mask, host_ip_addr);

// Search for RF627Old devices over network by Service Protocol.
search_scanners(scanners, kRF627_OLD, kSERVICE);

// Iterate over all discovered RF627Old in network and get info.
for(size_t i = 0; i < vector_count(scanners); i++)
   hello_information info = get_info_about_scanner(
                                (scanner_base_t*)vector_get(scanners,i),␣
↪kSERVICE);
```

connect_to_scanner()

Function to connect to the RF62X Series Scanner

rfUint8 **connect_to_scanner**(scanner_base_t **device*, protocol_types_t *protocol*)
connect - Establish connection to the RF62X device

Return 0 on success

Parameters

- `device`: - prt to scanner
- `protocol`: - protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example

```
// Create value for scanners vector's type
vector_t* scanners = (vector_t*)calloc(1, sizeof (vector_t));
//Initialization vector
vector_init(&scanners);

// set IP Addr and NetMask for setting in network adapter settings.
uint32_t host_ip_addr = ntohl(inet_addr("192.168.1.2"));
uint32_t host_mask = ntohl(inet_addr("255.255.255.0"));
// call the function to change adapter settings inside the library.
set_platform_adapter_settings(host_mask, host_ip_addr);

// Search for RF627Old devices over network by Service Protocol.
search_scanners(scanners, kRF627_OLD, kSERVICE);

// Iterate over all discovered RF627Old in network and Establish connection.
```

(continues on next page)

(continued from previous page)

```
for(size_t i = 0; i < vector_count(scanners); i++)
   connect_to_scanner((scanner_base_t*)vector_get(scanners,i), kSERVICE);
```

disconnect_from_scanner()

Function to close a previously established connection with the RF62X Series Scanner

rfUint8 **disconnect_from_scanner**(scanner_base_t **device*, protocol_types_t *protocol*)

disconnect_from_scanner - Close connection to the device

Return 0 on success

Parameters

- `device`: - prt to scanner
- `protocol`: - protocol's type (Service, ENIP, Modbus-TCP)

Usage example

```
// Create value for scanners vector's type
vector_t* scanners = (vector_t*)calloc(1, sizeof (vector_t));
//Initialization vector
vector_init(&scanners);

// set IP Addr and NetMask for setting in network adapter settings.
uint32_t host_ip_addr = ntohl(inet_addr("192.168.1.2"));
uint32_t host_mask = ntohl(inet_addr("255.255.255.0"));
// call the function to change adapter settings inside the library.
set_platform_adapter_settings(host_mask, host_ip_addr);

// Search for RF627Old devices over network by Service Protocol.
search_scanners(scanners, kRF627_OLD, kSERVICE);

// Iterate over all discovered RF627Old in network and Establish connection.
for(size_t i = 0; i < vector_count(scanners); i++)
   connect_to_scanner((scanner_base_t*)vector_get(scanners,i), kSERVICE);

// Iterate over all discovered RF627Old in network for Disabling connection.
for(size_t i = 0; i < vector_count(scanners); i++)
   disconnect_from_scanner((scanner_base_t*)vector_get(scanners,i), kSERVICE);
```

get_profile2D_from_scanner()

Function for receiving profile from RF62X series scanners

rf627_profile2D_t ***get_profile2D_from_scanner**(scanner_base_t **device*, rfBool *zero_points*, protocol_types_t *protocol*)

get_profile - Get measurement from scanner's data stream

Return ptr to rf627_profile_t structure

Parameters

- `device`: - ptr to scanner
- `zero_points`: - include zero points in return profile2D
- `protocol`: - protocol's type (Service, ENIP, Modbus-TCP)

Usage example

```
// Create value for scanners vector's type
vector_t* scanners = (vector_t*)calloc(1, sizeof (vector_t));
//Initialization vector
vector_init(&scanners);

// set IP Addr and NetMask for setting in network adapter settings.
uint32_t host_ip_addr = ntohl(inet_addr("192.168.1.2"));
uint32_t host_mask = ntohl(inet_addr("255.255.255.0"));
// call the function to change adapter settings inside the library.
set_platform_adapter_settings(host_mask, host_ip_addr);

// Search for RF627Old devices over network by Service Protocol.
search_scanners(scanners, kRF627_OLD, kSERVICE);

// Iterate over all discovered RF627Old in network and Establish connection.
for(size_t i = 0; i < vector_count(scanners); i++)
{
   scanner_base_t* scanner = vector_get(scanners,i);
   connect_to_scanner(scanner, kSERVICE);

   // Flag for included zero points in return profile2D
   bool zero_points = true;
   // Get profile from scanner's data stream by Service Protocol.
   rf627_profile2D_t* profile = get_profile2D_from_scanner(scanner, zero_
↪points, kSERVICE);

   {
      // some actions with profile
   }

   disconnect_from_scanner(scanner, kSERVICE);

   // Freeing memory after using profile structure
   free(profile->rf627_profile2D->intensity);
   free(profile->rf627_profile2D->pixels_format.pixels);
   free(profile->rf627_profile2D);
   free(profile);
}
```

read_params_from_scanner()

Function for receiving current scanner settings. When this function is called, the «Core» reads out all relevant parameters from the scanner, saving them in the form of a «list of parameters» for further work.

rfUint8 **read_params_from_scanner**(scanner_base_t **device*, protocol_types_t *protocol*)

read_params_from_scanner - Read parameters from device to rfInternal structure. This structure is accessible via get_params() function

Return 0 on success

Parameters

- `device`: - ptr to scanner
- `protocol`: - protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example

```
// Create value for scanners vector's type
vector_t* scanners = (vector_t*)calloc(1, sizeof (vector_t));
//Initialization vector
vector_init(&scanners);

// set IP Addr and NetMask for setting in network adapter settings.
uint32_t host_ip_addr = ntohl(inet_addr("192.168.1.2"));
uint32_t host_mask = ntohl(inet_addr("255.255.255.0"));
// call the function to change adapter settings inside the library.
set_platform_adapter_settings(host_mask, host_ip_addr);

// Search for RF627Old devices over network by Service Protocol.
search_scanners(scanners, kRF627_OLD, kSERVICE);

// Iterate over all discovered RF627Old in network and Establish connection.
for(size_t i = 0; i < vector_count(scanners); i++)
{
   scanner_base_t* scanner = vector_get(scanners,i);
   connect_to_scanner(scanner, kSERVICE);

   // Read parameters from device to the internal structure of the core
   read_params_from_scanner(scanner, kSERVICE);

   {
      // some actions with params
   }

   disconnect_from_scanner(scanner, kSERVICE);
}
```

get_parameter()

The function of obtaining a specific parameter by its name (key). When this function is called, the «Core» searches for the desired parameter from the last read when the function was called :ref: *read_params_from_scanner*. In case the requested parameter is absent in a specific scanner, the function will return null.

parameter_t ***get_parameter**(scanner_base_t **device*, **const** rfChar **param_name*)

get_parameter - Search parameters by his name

Return param on success, else - null

Parameters

- `device`: - ptr to scanner
- `param_name`: - name of parameter

Usage example

```
// Create value for scanners vector's type
vector_t* scanners = (vector_t*)calloc(1, sizeof (vector_t));
//Initialization vector
vector_init(&scanners);

// set IP Addr and NetMask for setting in network adapter settings.
uint32_t host_ip_addr = ntohl(inet_addr("192.168.1.2"));
uint32_t host_mask = ntohl(inet_addr("255.255.255.0"));
// call the function to change adapter settings inside the library.
set_platform_adapter_settings(host_mask, host_ip_addr);

// Search for RF627Old devices over network by Service Protocol.
search_scanners(scanners, kRF627_OLD, kSERVICE);

// Iterate over all discovered RF627Old in network and Establish connection.
for(size_t i = 0; i < vector_count(scanners); i++)
{
   scanner_base_t* scanner = vector_get(scanners,i);
   connect_to_scanner(scanner, kSERVICE);

   // Read parameters from device to the internal structure of the core
   read_params_from_scanner(scanner, kSERVICE);

   // Get parameter of Device Name
   parameter_t* name = get_parameter(scanner, "user_general_deviceName");
   if ((name != NULL) && (strcmp(name->type, "string_t")==0)
   {
      char* str_name = name->val_str->value;
      printf("Current Device Name: %s\n", str_name);
   }

   disconnect_from_scanner(scanner, kSERVICE);
}
```

For more convenient work with parameters, you can use the corresponding «keys» (parameter name key, parameter type and access to the parameter). To do this, in the file `rt62X_types.h` are the following `enum`

enum paramValueType_t

Values:

PVT_UNKN = 0

PVT_UINT

PVT_UINT64

PVT_INT

PVT_INT64

PVT_FLOAT

PVT_DOUBLE

PVT_ARRAY_UINT32

PVT_ARRAY_UINT64

PVT_ARRAY_INT32

PVT_ARRAY_INT64

PVT_ARRAY_FLT

PVT_ARRAY_DBL

PVT_STRING

PVT_UNKN = 0

PVT_UINT

PVT_UINT64

PVT_INT

PVT_INT64

PVT_FLOAT

PVT_DOUBLE

PVT_ARRAY_UINT32

PVT_ARRAY_UINT64

PVT_ARRAY_INT32

PVT_ARRAY_INT64

PVT_ARRAY_FLT

PVT_ARRAY_DBL

PVT_STRING

PVT_UNKN = 0

PVT_UINT

PVT_UINT64

PVT_INT

PVT_INT64

PVT_FLOAT

PVT_DOUBLE

PVT_ARRAY_UINT32

PVT_ARRAY_UINT64

PVT_ARRAY_INT32

PVT_ARRAY_INT64

PVT_ARRAY_FLT

PVT_ARRAY_DBL

PVT_STRING

enum paramAccessType_t

Values:

PAT_UNKN = 0

PAT_READ_ONLY

PAT_WRITE

PAT_LOCKED

PAT_UNKN = 0

PAT_READ_ONLY

PAT_WRITE

PAT_LOCKED

PAT_UNKN = 0

PAT_READ_ONLY

PAT_WRITE

PAT_LOCKED

enum parameter_name_keys_t

Values:

FACT_GENERAL_PROTOCOLREV = 0

FACT_GENERAL_DEVICETYPE

FACT_GENERAL_SERIAL

FACT_GENERAL_PCBSERIAL

FACT_GENERAL_LIFETIME

FACT_GENERAL_WORKTIME

FACT_GENERAL_STARTSCOUNT

FACT_GENERAL_FIRMWAREREV

FACT_GENERAL_HARDWAREREV

FACT_GENERAL_FSBLREV

FACT_GENERAL_CUSTOMERID

FACT_GENERAL_FPGAFREQ

FACT_GENERAL_SMR

FACT_GENERAL_MR

FACT_GENERAL_XSMR

FACT_GENERAL_XEMR

FACT_GENERAL_PIXDIVIDER

FACT_GENERAL_PROFDIVIDER

FACT_GENERAL_OEMDEVNAME

FACT_GENERAL_AUTHSTATUS

FACT_SENSOR_NAME

FACT_SENSOR_WIDTH

FACT_SENSOR_HEIGHT

FACT_SENSOR_PIXFREQ

FACT_SENSOR_FRMCONSTPART

FACT_SENSOR_FRMPERLINEPART

FACT_SENSOR_FPSOREXP

FACT_SENSOR_MINEXPOSURE

FACT_SENSOR_MAXEXPOSURE

FACT_SENSOR_IMGFLIP

FACT_NETWORK_MACADDR

FACT_NETWORK_FORCEAUTONEGTIME

FACT_NETWORK_WEBSOCKSERVICEPORT

FACT_NETWORK_WEBSOCKDATAPORT

FACT_NETWORK_WEBSOCKMATHPORT

FACT_LASER_WAVELENGTH

FACT_LASER_KOEFF1

FACT_LASER_KOEFF2

FACT_LASER_MINVALUE

FACT_LASER_MAXVALUE

FACT_PROFILES_MAXDUMPSIZE

FACT_EIP_IDENTITY_VENDORID

FACT_EIP_IDENTITY_DEVICETYPE

FACT_EIP_IDENTITY_PRODUCTCODE

FACT_EIP_IDENTITY_REV

FACT_EIP_TCPINTRF_CAPABILITY

FACT_EIP_TCPINTRF_PHY_PATHSIZE

FACT_EIP_TCPINTRF_PHY_CLASSID

FACT_EIP_TCPINTRF_PHY_INSTNUMBER

FACT_EIP_TCPINTRF_PHY_ATTRNUMBER

FACT_EIP_INTRFTYPE

FACT_EIP_INTRFCAPABILITY_BITS

FACT_EIP_INTRFCAPABILITY_SPEEDDUPCOUNT

FACT_EIP_INTRFCAPABILITY_SPEEDDUPLEX_SPEED

FACT_EIP_INTRFCAPABILITY_SPEEDDUPLEX_DUPLEX

FACT_SENSOR_ANALOGGAIN

FACT_SENSOR_DIGITALGAIN

FACT_SENSOR_BLACKODD

FACT_SENSOR_BLACKEVEN

FACT_SENSOR_HDRPIECEWISEDIV1

FACT_SENSOR_HDRPIECEWISEDIV2

FACT_SENSOR_INITREGS

USER_GENERAL_DEVICESTATE

USER_GENERAL_DEVICENAME

USER_GENERAL_SAVELOG

USER_SYSMON_FPGATEMP

USER_SYSMON_PARAMSCHANGED

USER_SYSMON_TEMPSENS00

USER_SYSMON_TEMPSENS00MAX

USER_SYSMON_TEMPSENS00MIN

USER_SYSMON_TEMPSENS01

USER_SYSMON_TEMPSENS01MAX

USER_SYSMON_TEMPSENS01MIN

USER_SYSMON_TEMPSENS10

USER_SYSMON_TEMPSENS10MAX

USER_SYSMON_TEMPSENS10MIN

USER_SYSMON_TEMPSENS11

USER_SYSMON_TEMPSENS11MAX

USER_SYSMON_TEMPSENS11MIN

USER_SENSOR_SYNCSOURCE

USER_SENSOR_FRAMERATE

USER_SENSOR_MAXFRAMERATE

USER_SENSOR_EXPOSURECONTROL

USER_SENSOR_EXPOSURE1

USER_SENSOR_EXPOSURE2

USER_SENSOR_EXPOSURE3

USER_SENSOR_EXPOSURE4

USER_SENSOR_MAXEXPOSURE

USER_ROI_ENABLED

USER_ROI_ACTIVE

USER_ROI_POSMODE

USER_ROI_POS

USER_ROI_MAXPOS

USER_ROI_REQPROFSIZE

USER_NETWORK_SPEED

USER_NETWORK_REQUIREDSPEED

USER_NETWORK_AUTONEG

USER_NETWORK_IP

USER_NETWORK_MASK

USER_NETWORK_GATEWAY

USER_NETWORK_HOSTIP

USER_NETWORK_HOSTPORT

USER_NETWORK_WEBPORT

USER_NETWORK_SERVICEPORT

USER_STREAMS_UDPENABLED

USER_STREAMS_FORMAT

USER_STREAMS_INCLUDEINTENSITY

USER_PROCESSING_THRESHOLD

USER_PROCESSING_PROFPERSEC

USER_PROCESSING_MEDIANMODE

USER_PROCESSING_BILATERALMODE

USER_PROCESSING_PEAKMODE

USER_PROCESSING_FLIP

USER_LASER_ENABLED

USER_LASER_VALUE

USER_TRIGGER_SYNC_SOURCE

USER_TRIGGER_SYNC_STRICTENABLED

USER_TRIGGER_SYNC_DIVIDER

USER_TRIGGER_SYNC_DELAY

USER_TRIGGER_COUNTER_TYPE

USER_TRIGGER_COUNTER_MAXVALUEENABLED

USER_TRIGGER_COUNTER_MAXVALUE

USER_TRIGGER_COUNTER_RESETTIMERENABLED

USER_TRIGGER_COUNTER_RESETTIMERVALUE

USER_TRIGGER_COUNTER_VALUE

USER_INPUT1_ENABLED

USER_INPUT1_MODE

USER_INPUT2_ENABLED

USER_INPUT2_MODE

USER_INPUT3_ENABLED

USER_INPUT3_MODE

USER_INPUT1_SAMPLES

USER_INPUT2_SAMPLES

USER_INPUT3_SAMPLES

USER_OUTPUT1_ENABLED

USER_OUTPUT1_MODE

USER_OUTPUT1_PULSEWIDTH

USER_OUTPUT2_ENABLED

USER_OUTPUT2_MODE

USER_OUTPUT2_PULSEWIDTH

USER_DUMP_ENABLED

USER_DUMP_CAPACITY

USER_DUMP_SIZE

USER_DUMP_TIMESTAMP

USER_DUMP_VIEW3D_MOTIONTYPE

USER_DUMP_VIEW3D_YSOURCE

USER_DUMP_VIEW3D_YSTEP

USER_DUMP_VIEW3D_PAINTMODE

USER_DUMP_VIEW3D_DECIMATION

USER_EIP_TCPPORT

USER_EIP_UDPPORT

USER_EIP_TCP_TTL

USER_EIP_TCP_TIMEOUT

USER_EIP_TCP_MULTICAST_ALLOC

USER_EIP_TCP_MULTICAST_NUM

USER_EIP_TCP_MULTICAST_ADDR

USER_COMPATIBILITY_RF625ENABLED

USER_COMPATIBILITY_RF625TCPPORT

USER_SENSOR_DOUBLESPEEDENABLED

USER_SENSOR_EDRTYPE

USER_SENSOR_EDRCOLUMNDIVIDER

USER_STREAMS_POINTSCOUNT

USER_ROI_SIZE

FACT_GENERAL_PROTOCOLREV = 0

FACT_GENERAL_DEVICETYPE

FACT_GENERAL_SERIAL

FACT_GENERAL_PCBSERIAL

FACT_GENERAL_LIFETIME

FACT_GENERAL_WORKTIME

FACT_GENERAL_STARTSCOUNT

FACT_GENERAL_FIRMWAREREV

FACT_GENERAL_HARDWAREREV

FACT_GENERAL_FSBLREV

FACT_GENERAL_CUSTOMERID

FACT_GENERAL_FPGAFREQ

FACT_GENERAL_SMR

FACT_GENERAL_MR

FACT_GENERAL_XSMR

FACT_GENERAL_XEMR

FACT_GENERAL_PIXDIVIDER

FACT_GENERAL_PROFDIVIDER

FACT_GENERAL_OEMDEVNAME

FACT_GENERAL_AUTHSTATUS

FACT_SENSOR_NAME

FACT_SENSOR_WIDTH

FACT_SENSOR_HEIGHT

FACT_SENSOR_PIXFREQ

FACT_SENSOR_FRMCONSTPART

FACT_SENSOR_FRMPERLINEPART

FACT_SENSOR_FPSOREXP

FACT_SENSOR_MINEXPOSURE

FACT_SENSOR_MAXEXPOSURE

FACT_SENSOR_IMGFLIP

FACT_NETWORK_MACADDR

FACT_NETWORK_FORCEAUTONEGTIME

FACT_NETWORK_WEBSOCKSERVICEPORT

FACT_NETWORK_WEBSOCKDATAPORT

FACT_NETWORK_WEBSOCKMATHPORT

FACT_LASER_WAVELENGTH

FACT_LASER_KOEFF1

FACT_LASER_KOEFF2

FACT_LASER_MINVALUE

FACT_LASER_MAXVALUE

FACT_PROFILES_MAXDUMPSIZE

FACT_EIP_IDENTITY_VENDORID

FACT_EIP_IDENTITY_DEVICETYPE

FACT_EIP_IDENTITY_PRODUCTCODE

FACT_EIP_IDENTITY_REV

FACT_EIP_TCPINTRF_CAPABILITY

FACT_EIP_TCPINTRF_PHY_PATHSIZE

FACT_EIP_TCPINTRF_PHY_CLASSID

FACT_EIP_TCPINTRF_PHY_INSTNUMBER

FACT_EIP_TCPINTRF_PHY_ATTRNUMBER

FACT_EIP_INTRFTYPE

FACT_EIP_INTRFCAPABILITY_BITS

FACT_EIP_INTRFCAPABILITY_SPEEDDUPCOUNT

FACT_EIP_INTRFCAPABILITY_SPEEDDUPLEX_SPEED

FACT_EIP_INTRFCAPABILITY_SPEEDDUPLEX_DUPLEX

FACT_SENSOR_ANALOGGAIN

FACT_SENSOR_DIGITALGAIN

FACT_SENSOR_BLACKODD

FACT_SENSOR_BLACKEVEN

FACT_SENSOR_HDRPIECEWISEDIV1

FACT_SENSOR_HDRPIECEWISEDIV2

FACT_SENSOR_INITREGS

USER_GENERAL_DEVICESTATE

USER_GENERAL_DEVICENAME

USER_GENERAL_SAVELOG

USER_SYSMON_FPGATEMP

USER_SYSMON_PARAMSCHANGED

USER_SYSMON_TEMPSENS00

USER_SYSMON_TEMPSENS00MAX

USER_SYSMON_TEMPSENS00MIN

USER_SYSMON_TEMPSENS01

USER_SYSMON_TEMPSENS01MAX

USER_SYSMON_TEMPSENS01MIN

USER_SYSMON_TEMPSENS10

USER_SYSMON_TEMPSENS10MAX

USER_SYSMON_TEMPSENS10MIN

USER_SYSMON_TEMPSENS11

USER_SYSMON_TEMPSENS11MAX

USER_SYSMON_TEMPSENS11MIN

USER_SENSOR_SYNCSOURCE

USER_SENSOR_FRAMERATE

USER_SENSOR_MAXFRAMERATE

USER_SENSOR_EXPOSURECONTROL

USER_SENSOR_EXPOSURE1

USER_SENSOR_EXPOSURE2

USER_SENSOR_EXPOSURE3

USER_SENSOR_EXPOSURE4

USER_SENSOR_MAXEXPOSURE

USER_ROI_ENABLED

USER_ROI_ACTIVE

USER_ROI_POSMODE

USER_ROI_POS

USER_ROI_MAXPOS

USER_ROI_REQPROFSIZE

USER_NETWORK_SPEED

USER_NETWORK_REQUIREDSPEED

USER_NETWORK_AUTONEG

USER_NETWORK_IP

USER_NETWORK_MASK

USER_NETWORK_GATEWAY

USER_NETWORK_HOSTIP

USER_NETWORK_HOSTPORT

USER_NETWORK_WEBPORT

USER_NETWORK_SERVICEPORT

USER_STREAMS_UDPENABLED

USER_STREAMS_FORMAT

USER_STREAMS_INCLUDEINTENSITY

USER_PROCESSING_THRESHOLD

USER_PROCESSING_PROFPERSEC

USER_PROCESSING_MEDIANMODE

USER_PROCESSING_BILATERALMODE

USER_PROCESSING_PEAKMODE

USER_PROCESSING_FLIP

USER_LASER_ENABLED

USER_LASER_VALUE

USER_TRIGGER_SYNC_SOURCE

USER_TRIGGER_SYNC_STRICTENABLED

USER_TRIGGER_SYNC_DIVIDER

USER_TRIGGER_SYNC_DELAY

USER_TRIGGER_COUNTER_TYPE

USER_TRIGGER_COUNTER_MAXVALUEENABLED

USER_TRIGGER_COUNTER_MAXVALUE

USER_TRIGGER_COUNTER_RESETTIMERENABLED

USER_TRIGGER_COUNTER_RESETTIMERVALUE

USER_TRIGGER_COUNTER_VALUE

USER_INPUT1_ENABLED

USER_INPUT1_MODE

USER_INPUT2_ENABLED

USER_INPUT2_MODE

USER_INPUT3_ENABLED

USER_INPUT3_MODE

USER_INPUT1_SAMPLES

USER_INPUT2_SAMPLES

USER_INPUT3_SAMPLES

USER_OUTPUT1_ENABLED

USER_OUTPUT1_MODE

USER_OUTPUT1_PULSEWIDTH

USER_OUTPUT2_ENABLED

USER_OUTPUT2_MODE

USER_OUTPUT2_PULSEWIDTH

USER_DUMP_ENABLED

USER_DUMP_CAPACITY

USER_DUMP_SIZE

USER_DUMP_TIMESTAMP

USER_DUMP_VIEW3D_MOTIONTYPE

USER_DUMP_VIEW3D_YSOURCE

USER_DUMP_VIEW3D_YSTEP

USER_DUMP_VIEW3D_PAINTMODE

USER_DUMP_VIEW3D_DECIMATION

USER_EIP_TCPPORT

USER_EIP_UDPPORT

USER_EIP_TCP_TTL

USER_EIP_TCP_TIMEOUT

USER_EIP_TCP_MULTICAST_ALLOC

USER_EIP_TCP_MULTICAST_NUM

USER_EIP_TCP_MULTICAST_ADDR

USER_COMPATIBILITY_RF625ENABLED

USER_COMPATIBILITY_RF625TCPPORT

USER_SENSOR_DOUBLESPEEDENABLED

USER_SENSOR_EDRTYPE

USER_SENSOR_EDRCOLUMNDIVIDER

USER_STREAMS_POINTSCOUNT

USER_ROI_SIZE

FACT_GENERAL_PROTOCOLREV = 0

FACT_GENERAL_DEVICETYPE

FACT_GENERAL_SERIAL

FACT_GENERAL_PCBSERIAL

FACT_GENERAL_LIFETIME

FACT_GENERAL_WORKTIME

FACT_GENERAL_STARTSCOUNT

FACT_GENERAL_FIRMWAREREV

FACT_GENERAL_HARDWAREREV

FACT_GENERAL_FSBLREV

FACT_GENERAL_CUSTOMERID

FACT_GENERAL_FPGAFREQ

FACT_GENERAL_SMR

FACT_GENERAL_MR

FACT_GENERAL_XSMR

FACT_GENERAL_XEMR

FACT_GENERAL_PIXDIVIDER

FACT_GENERAL_PROFDIVIDER

FACT_GENERAL_OEMDEVNAME

FACT_GENERAL_AUTHSTATUS

FACT_SENSOR_NAME

FACT_SENSOR_WIDTH

FACT_SENSOR_HEIGHT

FACT_SENSOR_PIXFREQ

FACT_SENSOR_FRMCONSTPART

FACT_SENSOR_FRMPERLINEPART

FACT_SENSOR_FPSOREXP

FACT_SENSOR_MINEXPOSURE

FACT_SENSOR_MAXEXPOSURE

FACT_SENSOR_IMGFLIP

FACT_NETWORK_MACADDR

FACT_NETWORK_FORCEAUTONEGTIME

FACT_NETWORK_WEBSOCKSERVICEPORT

FACT_NETWORK_WEBSOCKDATAPORT

FACT_NETWORK_WEBSOCKMATHPORT

FACT_LASER_WAVELENGTH

FACT_LASER_KOEFF1

FACT_LASER_KOEFF2

FACT_LASER_MINVALUE

FACT_LASER_MAXVALUE

FACT_PROFILES_MAXDUMPSIZE

FACT_EIP_IDENTITY_VENDORID

FACT_EIP_IDENTITY_DEVICETYPE

FACT_EIP_IDENTITY_PRODUCTCODE

FACT_EIP_IDENTITY_REV

FACT_EIP_TCPINTRF_CAPABILITY

FACT_EIP_TCPINTRF_PHY_PATHSIZE

FACT_EIP_TCPINTRF_PHY_CLASSID

FACT_EIP_TCPINTRF_PHY_INSTNUMBER

FACT_EIP_TCPINTRF_PHY_ATTRNUMBER

FACT_EIP_INTRFTYPE

FACT_EIP_INTRFCAPABILITY_BITS

FACT_EIP_INTRFCAPABILITY_SPEEDDUPCOUNT

FACT_EIP_INTRFCAPABILITY_SPEEDDUPLEX_SPEED

FACT_EIP_INTRFCAPABILITY_SPEEDDUPLEX_DUPLEX

FACT_SENSOR_ANALOGGAIN

FACT_SENSOR_DIGITALGAIN

FACT_SENSOR_BLACKODD

FACT_SENSOR_BLACKEVEN

FACT_SENSOR_HDRPIECEWISEDIV1

FACT_SENSOR_HDRPIECEWISEDIV2

FACT_SENSOR_INITREGS

USER_GENERAL_DEVICESTATE

USER_GENERAL_DEVICENAME

USER_GENERAL_SAVELOG

USER_SYSMON_FPGATEMP

USER_SYSMON_PARAMSCHANGED

USER_SYSMON_TEMPSENS00

USER_SYSMON_TEMPSENS00MAX

USER_SYSMON_TEMPSENS00MIN

USER_SYSMON_TEMPSENS01

USER_SYSMON_TEMPSENS01MAX

USER_SYSMON_TEMPSENS01MIN

USER_SYSMON_TEMPSENS10

USER_SYSMON_TEMPSENS10MAX

USER_SYSMON_TEMPSENS10MIN

USER_SYSMON_TEMPSENS11

USER_SYSMON_TEMPSENS11MAX

USER_SYSMON_TEMPSENS11MIN

USER_SENSOR_SYNCSOURCE

USER_SENSOR_FRAMERATE

USER_SENSOR_MAXFRAMERATE

USER_SENSOR_EXPOSURECONTROL

USER_SENSOR_EXPOSURE1

USER_SENSOR_EXPOSURE2

USER_SENSOR_EXPOSURE3

USER_SENSOR_EXPOSURE4

USER_SENSOR_MAXEXPOSURE

USER_ROI_ENABLED

USER_ROI_ACTIVE

USER_ROI_POSMODE

USER_ROI_POS

USER_ROI_MAXPOS

USER_ROI_REQPROFSIZE

USER_NETWORK_SPEED

USER_NETWORK_REQUIREDSPEED

USER_NETWORK_AUTONEG

USER_NETWORK_IP

USER_NETWORK_MASK

USER_NETWORK_GATEWAY

USER_NETWORK_HOSTIP

USER_NETWORK_HOSTPORT

USER_NETWORK_WEBPORT

USER_NETWORK_SERVICEPORT

USER_STREAMS_UDPENABLED

USER_STREAMS_FORMAT

USER_STREAMS_INCLUDEINTENSITY

USER_PROCESSING_THRESHOLD

USER_PROCESSING_PROFPERSEC

USER_PROCESSING_MEDIANMODE

USER_PROCESSING_BILATERALMODE

USER_PROCESSING_PEAKMODE

USER_PROCESSING_FLIP

USER_LASER_ENABLED

USER_LASER_VALUE

USER_TRIGGER_SYNC_SOURCE

USER_TRIGGER_SYNC_STRICTENABLED

USER_TRIGGER_SYNC_DIVIDER

USER_TRIGGER_SYNC_DELAY

USER_TRIGGER_COUNTER_TYPE

USER_TRIGGER_COUNTER_MAXVALUEENABLED

USER_TRIGGER_COUNTER_MAXVALUE

USER_TRIGGER_COUNTER_RESETTIMERENABLED

USER_TRIGGER_COUNTER_RESETTIMERVALUE

USER_TRIGGER_COUNTER_VALUE

USER_INPUT1_ENABLED

USER_INPUT1_MODE

USER_INPUT2_ENABLED

USER_INPUT2_MODE

USER_INPUT3_ENABLED

USER_INPUT3_MODE

USER_INPUT1_SAMPLES

USER_INPUT2_SAMPLES

USER_INPUT3_SAMPLES

USER_OUTPUT1_ENABLED

USER_OUTPUT1_MODE

USER_OUTPUT1_PULSEWIDTH

USER_OUTPUT2_ENABLED

USER_OUTPUT2_MODE

USER_OUTPUT2_PULSEWIDTH

USER_DUMP_ENABLED

USER_DUMP_CAPACITY

USER_DUMP_SIZE

USER_DUMP_TIMESTAMP

USER_DUMP_VIEW3D_MOTIONTYPE

USER_DUMP_VIEW3D_YSOURCE

USER_DUMP_VIEW3D_YSTEP

USER_DUMP_VIEW3D_PAINTMODE

USER_DUMP_VIEW3D_DECIMATION

USER_EIP_TCPPORT

USER_EIP_UDPPORT

USER_EIP_TCP_TTL

USER_EIP_TCP_TIMEOUT

USER_EIP_TCP_MULTICAST_ALLOC

USER_EIP_TCP_MULTICAST_NUM

USER_EIP_TCP_MULTICAST_ADDR

USER_COMPATIBILITY_RF625ENABLED

USER_COMPATIBILITY_RF625TCPPORT

USER_SENSOR_DOUBLESPEEDENABLED

USER_SENSOR_EDRTYPE

USER_SENSOR_EDRCOLUMNDIVIDER

USER_STREAMS_POINTSCOUNT

USER_ROI_SIZE

Key usage example:

```
{
...Search devices
...Establish connections
...Read parameters
}

// Get parameter of Device Name
parameter_t* name = get_parameter(scanner, parameter_names_array[USER_GENERAL_
↪DEVICENAME]);
if ((name != NULL) && (strcmp(name->type, parameter_value_types[PVT_
↪STRING])==0)
{
   char* str_name = name->val_str->value;
   printf("Current Device Name: %s\n", str_name);
}
```

For a more detailed description of each parameter and its properties, see *PARAMETERS*

set_parameter()

The function of setting a specific parameter. When this function is called, the transmitted parameter is set in the local parameter list in the «Core». To send changes to the scanner, you must call the function `write_params_to_scanner`.

rfUint8 **`set_parameter`**(scanner_base_t **device*, parameter_t **param*)

set_parameter - Set parameter

Return 0 if success

Parameters

- `device`: - ptr to scanner
- `param`: - setting parameter

Usage example

```
// Create value for scanners vector's type
vector_t* scanners = (vector_t*)calloc(1, sizeof (vector_t));
//Initialization vector
vector_init(&scanners);

// set IP Addr and NetMask for setting in network adapter settings.
uint32_t host_ip_addr = ntohl(inet_addr("192.168.1.2"));
uint32_t host_mask = ntohl(inet_addr("255.255.255.0"));
// call the function to change adapter settings inside the library.
set_platform_adapter_settings(host_mask, host_ip_addr);

// Search for RF627Old devices over network by Service Protocol.
search_scanners(scanners, kRF627_OLD, kSERVICE);

// Iterate over all discovered RF627Old in network and Establish connection.
for(size_t i = 0; i < vector_count(scanners); i++)
{
   scanner_base_t* scanner = vector_get(scanners,i);
   connect_to_scanner(scanner, kSERVICE);

   // Read parameters from device to the internal structure of the core
   read_params_from_scanner(scanner, kSERVICE);

   // Get parameter of Device Name
   parameter_t* name = get_parameter(scanner, "user_general_deviceName");
   if ((name != NULL) && (strcmp(name->type, "string_t")==0)
   {
      char* str_name = name->val_str->value;
      printf("Current Device Name: %s\n", str_name);

      char* new_name = "NEW NAME";
      memcpy(name->val_str->value, new_name, strlen(new_name)+1);
      set_parameter(scanner, name);
   }

   //  Write changes parameters to the device's memory
   write_params_to_scanner(scanner, kSERVICE);

   disconnect_from_scanner(scanner, kSERVICE);
}
```

write_params_to_scanner()

The function of writing local parameters from the «Core» to the scanner. When this function is called, a list of local parameters is sent from the «Core» to the scanner.

rfUint8 **write_params_to_scanner**(scanner_base_t **device*, protocol_types_t *protocol*)

write_params_to_scanner - Write current parameters to device's memory

Return 0 on success

Parameters

- `device`: - ptr to scanner
- `protocol`: - protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example

```
{
...Search devices
...Establish connections
...Read parameters
}

// Get parameter of Laser Enabled
parameter_t* laser_enabled = get_parameter(scanner, "user_laser_enabled");
if ((name != NULL) && (strcmp(name->type, "uint32_t")==0)
{
   uint32_t is_enabled = laser_enabled->val_uint32->value;
   printf("Current Laser State: %s\n", is_enabled == 0 ? "OFF" : "ON");

   uint32_t new_state;
   if (is_enabled == 1)
      new_state = 0;
   else
      new_state = 1;

   laser_enabled->val_uint32->value = new_state;

   set_parameter(scanner, laser_enabled);
}

//  Write changes parameters to the device's memory
write_params_to_scanner(scanner, kSERVICE);
```

send_command()

Function for sending commands to the scanner

rfUint8 **send_command**(scanner_base_t **device*, command_t **command*)

set_parameter - Search parameters by his name

Return param on success, else - null

Parameters

- `device`: - ptr to scanner
- `param_name`: - name of parameter

For a more detailed description of commands and their properties, see *General device commands*

2.2 Wrapper API in C

This library allows you to simplify the development of C applications

To use it in C projects, the developer needs to include h-files of the library in his project, and also add a “wrapper” to the project as a static or dynamic program library.

To download the library see *latest C wrapper releases*.
To compile the library see compile and run a C wrapper.

2.2.1 SDK initialization

The file `rf62Xcore.h` is required to call the SDK initialization function: `core_init()`

The file `rf62X_sdk.h` is the main program interface (API) file for developing programs in C language and defines the functionality of the «wrapper» library for rf62Xcore.

The file `rf62X_types.h` contains the basic structures and types used in the SDK.

core_init()

SDK initialization function. Must be called once before further calls to any library functions:

rfBool **`core_init`**`()`

sdk_version()

Function to get the current SDK version:

char ***`sdk_version`**`(`void`)`
sdk_version - Return info about SDK version

Return SDK version

2.2.2 Interface for working with rf627old

The files `rf62X_sdk.h`, `rf62X_types.h` and `rf62Xcore.h` provide all the necessary interface for working with the RF627Old series scanners

search_scanners()

Function to search for RF627 devices available on the network

rfUint8 **`search_scanners`**`(`vector_t **list*, scanner_types_t *model*, protocol_types_t *protocol*`)`
search - Search for RF62X devices over network

Return 0 on success

Parameters

- `list`: - ptr to list of rf627 objects. If not null list will be filled with found devices
- `model`: - scanner's type (RF627-old, RF627-smart)
- `protocol`: - protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example:

```
// Initialize sdk library
score_init();

// Print return rf627 sdk version
printf("SDK version: %s\n", sdk_version());
```

(continues on next page)

(continued from previous page)

```
printf("========================================\n");

// Create value for scanners vector's type
vector_t* scanners = (vector_t*)calloc(1, sizeof (vector_t));
// Initialization vector
vector_init(&scanners);

uint32_t host_ip_addr = ntohl(inet_addr("192.168.1.1"));
uint32_t host_mask = ntohl(inet_addr("255.255.255.0"));

// call the function to change adapter settings inside the library.
set_platform_adapter_settings(host_mask, host_ip_addr);

// Search for RF627-old devices over network by Service Protocol.
search_scanners(scanners, kRF627_OLD, kSERVICE);
```

get_info_about_scanner()

Function for getting information about the scanner from the hello package

hello_information **get_info_about_scanner**(scanner_base_t **device*, protocol_types_t *protocol*)

get_hello_info_of_scanners - Get information about scanner from hello packet

Return 0 on success

Parameters

- `device`: - prt to scanner
- `protocol`: - protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example:

```
// Initialize sdk library
score_init();

// Print return rf627 sdk version
printf("SDK version: %s\n", sdk_version());
printf("========================================\n");

// Create value for scanners vector's type
vector_t* scanners = (vector_t*)calloc(1, sizeof (vector_t));
// Initialization vector
vector_init(&scanners);

uint32_t host_ip_addr = ntohl(inet_addr("192.168.1.1"));
uint32_t host_mask = ntohl(inet_addr("255.255.255.0"));

// call the function to change adapter settings inside the library.
set_platform_adapter_settings(host_mask, host_ip_addr);

// Search for RF627-old devices over network by Service Protocol.
search_scanners(scanners, kRF627_OLD, kSERVICE);

// Print count of discovered RF627Old in network by Service Protocol
```

(continues on next page)

(continued from previous page)

```
printf("Discovered: %d rf627-old\n", (int)vector_count(scanners));

for (int i = 0; i < (int)vector_count(scanners); i++)
{
   hello_information info = get_info_about_scanner(vector_get(scanners,i),␣
↪kSERVICE);

   printf("\n\n\nID scanner's list: %d\n", i);
   printf("----------------------------------------\n");
   printf("Device information: \n");
   printf("* Name\t: %s\n", info.rf627old.hello_info_service_protocol->device_
↪name);
   printf("* Serial\t: %d\n", info.rf627old.hello_info_service_protocol->
↪serial_number);
   printf("* IP Addr\t: %d.%d.%d.%d\n",
              info.rf627old.hello_info_service_protocol->ip_address[0],
              info.rf627old.hello_info_service_protocol->ip_address[1],
              info.rf627old.hello_info_service_protocol->ip_address[2],
              info.rf627old.hello_info_service_protocol->ip_address[3]);
   printf("* MAC Addr\t: %d:%d:%d:%d:%d:%d\n",
              info.rf627old.hello_info_service_protocol->mac_address[0],
              info.rf627old.hello_info_service_protocol->mac_address[1],
              info.rf627old.hello_info_service_protocol->mac_address[2],
              info.rf627old.hello_info_service_protocol->mac_address[3],
              info.rf627old.hello_info_service_protocol->mac_address[4],
              info.rf627old.hello_info_service_protocol->mac_address[5]);

   printf("\nWorking ranges: \n");
   printf("* Zsmr, mm\t: %d\n", info.rf627old.hello_info_service_protocol->z_
↪begin);
   printf("* Zmr , mm\t: %d\n", info.rf627old.hello_info_service_protocol->z_
↪range);
   printf("* Xsmr, mm\t: %d\n", info.rf627old.hello_info_service_protocol->x_
↪begin);
   printf("* Xemr, mm\t: %d\n", info.rf627old.hello_info_service_protocol->x_
↪end);

   printf("\nVersions: \n");
   printf("* Firmware\t: %d\n", info.rf627old.hello_info_service_protocol->
↪firmware_version);
   printf("* Hardware\t: %d\n", info.rf627old.hello_info_service_protocol->
↪hardware_version);
   printf("----------------------------------------\n");
}
```

connect_to_scanner()

Function for establishing connection with RF627 series scanner

rfUint8 **connect_to_scanner**(scanner_base_t **device*, protocol_types_t *protocol*)

connect - Establish connection to the RF62X device

Return 0 on success

Parameters

- `device`: - prt to scanner

- `protocol`: - protocol's type (Service Protocol, ENIP, Modbus-TCP)

disconnect_from_scanner()

Function to close a previously established connection to the RF627 series scanner

rfUint8 **`disconnect_from_scanner`**(scanner_base_t **device*, protocol_types_t *protocol*)

disconnect_from_scanner - Close connection to the device

Return 0 on success

Parameters

- `device`: - prt to scanner
- `protocol`: - protocol's type (Service, ENIP, Modbus-TCP)

get_profile2D_from_scanner()

Function to get a profile from RF627 series scanners

rf627_profile2D_t ***`get_profile2D_from_scanner`**(scanner_base_t **device*, rfBool *zero_points*, protocol_types_t *protocol*)

get_profile - Get measurement from scanner's data stream

Return ptr to rf627_profile_t structure

Parameters

- `device`: - ptr to scanner
- `zero_points`: - include zero points in return profile2D
- `protocol`: - protocol's type (Service, ENIP, Modbus-TCP)

read_params_from_scanner()

The function of obtaining the current parameters of the scanner. When this function is called, the SDK reads all the actual parameters from the scanner, saving them as a «parameter list» for further work.

rfUint8 **`read_params_from_scanner`**(scanner_base_t **device*, protocol_types_t *protocol*)

read_params_from_scanner - Read parameters from device to rfInternal structure. This structure is accessible via get_params() function

Return 0 on success

Parameters

- `device`: - ptr to scanner
- `protocol`: - protocol's type (Service Protocol, ENIP, Modbus-TCP)

get_parameter()

Function for getting a specific parameter by its name (key). When this function is called, the SDK searches for the required parameter from the last read values when calling the function read_params_from_scanner. If the requested parameter is absent in a specific scanner, the function will return null.

parameter_t *__get_parameter__(scanner_base_t *_device_, __const__ rfChar *_param_name_)

get_parameter - Search parameters by his name

Return param on success, else - null

Parameters

- `device`: - ptr to scanner
- `param_name`: - name of parameter

set_parameter()

Function for setting a specific parameter. When this function is called, the passed parameter is set in the local parameter list in the SDK. To send changes to the scanner, call the `write_params` function.

rfUint8 __set_parameter__(scanner_base_t *_device_, parameter_t *_param_)

set_parameter - Set parameter

Return 0 if success

Parameters

- `device`: - ptr to scanner
- `param`: - setting parameter

write_params_to_scanner()

Function of writing local parameters from SDK to scanner. When this function is called, the list of local parameters is sent from the SDK to the scanner.

rfUint8 __write_params_to_scanner__(scanner_base_t *_device_, protocol_types_t _protocol_)

write_params_to_scanner - Write current parameters to device's memory

Return 0 on success

Parameters

- `device`: - ptr to scanner
- `protocol`: - protocol's type (Service Protocol, ENIP, Modbus-TCP)

2.3 Wrapper API in C++

This library makes it easy to develop C++ applications

To use it in C++ projects, the developer needs to include the library h-files in his project, as well as add a wrapper to the project as a static or dynamic program library.

To download the library, see *the latest C++ wrapper releases*.
To compile the library, see *compile and run the wrapper in C++*.

2.3.1 SDK initialization

The file `rf62Xsdk.h` is the main file for the programming interface (API) for developing C++ programs and defines the functionality of the wrapper library for rf62Xcore. `rf62Xsdk.h` contains the following set of classes and functions for initializing the SDK:

sdk_init()

SDK initialization function. Must be called once before further calls to any library functions:

bool `SDK::SCANNERS::RF62X::`**`sdk_init`**`()`
sdk_init - Initialize sdk library Must be called once before further calls to any library functions

Return true if success.

sdk_cleanup()

Function for cleaning resources allocated using the `sdk_init` function:

void `SDK::SCANNERS::RF62X::`**`sdk_cleanup`**`()`
sdk_cleanup - Cleanup resources allocated with sdk_init() function

sdk_version()

Function to get the current version of the SDK:

std::string `SDK::SCANNERS::RF62X::`**`sdk_version`**`()`
sdk_version - Return info about SDK version

Return SDK version

2.3.2 Class rf627old

This class is defined in the file `rf62Xsdk.h` and provides an interface for working with RF627Old series scanners

class rf627old
rf627old - This class is the main interface for working with RF627-old series scanners.

search()

Function to search for RF627 devices available on the network

std::vector<*rf627old* *> SDK::SCANNERS::RF62X::*rf627old*::**search**(PROTOCOLS *protocol*)
search - Search for *RF627old* devices over network

Return vector of rf627old devices

Parameters

- `protocol`: - protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example:

```
// Initialize sdk library
sdk_init();

// Print return rf627 sdk version
std::cout << "SDK version: " << sdk_version()                << std::endl;
std::cout << "========================================="     << std::endl;

// Create value for scanners vector's type
std::vector<rf627old*> list;
// Search for RF627old devices over network
list = rf627old::search(PROTOCOLS::SERVICE);

// Print count of discovered RF627Old in network by Service Protocol
std::cout << "Discovered: " << list.size() << " RF627Old"   << std::endl;
```

get_info()

Function for receiving information about the scanner from the Hello packet

rf627old::hello_info SDK::SCANNERS::RF62X::*rf627old*::**get_info**(PROTOCOLS *protocol* = PROTOCOLS::CURRENT)
get_info - Get information about scanner from hello packet

Return hello_info on success

Parameters

- `protocol`: - protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example:

```
// Initialize sdk library
sdk_init();

// Print return rf627 sdk version
std::cout << "SDK version: " << sdk_version()                 << std::endl;
std::cout << "========================================"      << std::endl;

// Create value for scanners vector's type
std::vector<rf627old*> list;
// Search for RF627old devices over network
list = rf627old::search(PROTOCOLS::SERVICE);

// Print count of discovered RF627Old in network by Service Protocol
std::cout << "Discovered: " << list.size() << " RF627Old"   << std::endl;

for (size_t i = 0; i < list.size(); i++)
{
   rf627old::hello_info info = list[i]->get_info();

   std::cout << "\n\n\nID scanner's list: " << i             << std::endl;
   std::cout << "----------------------------------------" << std::endl;
   std::cout << "Device information: "                      << std::endl;
   std::cout << "* Name\t: "     << info.device_name()      << std::endl;
   std::cout << "* Serial\t: "   << info.serial_number()    << std::endl;
   std::cout << "* IP Addr\t: "  << info.ip_address()       << std::endl;
   std::cout << "* MAC Addr\t: " << info.mac_address()      << std::endl;

   std::cout << "\nWorking ranges: "                         << std::endl;
   std::cout << "* Zsmr, mm\t: " << info.z_smr()            << std::endl;
   std::cout << "* Zmr , mm\t: " << info.z_mr()             << std::endl;
   std::cout << "* Xsmr, mm\t: " << info.x_smr()            << std::endl;
   std::cout << "* Xemr, mm\t: " << info.x_emr()            << std::endl;

   std::cout << "\nVersions: "                               << std::endl;
   std::cout << "* Firmware\t: " << info.firmware_version() << std::endl;
   std::cout << "* Hardware\t: " << info.hardware_version() << std::endl;
   std::cout << "----------------------------------------" << std::endl;
}

// Cleanup resources allocated with sdk_init()
sdk_cleanup();
```

connect()

Function for establishing connection with the RF627 Series Scanner

bool `SDK::SCANNERS::RF62X::rf627old::connect`(PROTOCOLS *protocol* = PROTOCOLS::CURRENT)

connect - Establish connection to the *RF627old* device

Return true on success

Parameters

- `protocol`: - protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example:

```
// Initialize sdk library
sdk_init();

// Create value for scanners vector's type
std::vector<rf627old*> list;
// Search for RF627old devices over network
list = rf627old::search(PROTOCOLS::SERVICE);

// Print count of discovered RF627Old in network by Service Protocol
std::cout << "Discovered: " << list.size() << " RF627Old"   << std::endl;

for (size_t i = 0; i < list.size(); i++)
{
   if (list[i]->connect())
      std::cout << "Connected to scanner №" << i << " successfully" <<␣
↪std::endl;
}
```

disconnect()

Function to close a previously established connection with the RF627 Series Scanner

bool SDK::SCANNERS::RF62X::*rf627old*::**disconnect**(PROTOCOLS *protocol* = PROTOCOLS::CURRENT)

disconnect_from_scanner - Close connection to the device

Return true on success

Parameters

- `protocol`: - protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example:

```
// Initialize sdk library
sdk_init();

// Create value for scanners vector's type
std::vector<rf627old*> list;
// Search for RF627old devices over network
list = rf627old::search(PROTOCOLS::SERVICE);

// Print count of discovered RF627Old in network by Service Protocol
std::cout << "Discovered: " << list.size() << " RF627Old"   << std::endl;

for (size_t i = 0; i < list.size(); i++)
   list[i]->connect();

{
...some actions with scanners
}

for (size_t i = 0; i < list.size(); i++)
   list[i]->disconnect();
```

get_profile2D()

Function for receiving a profile from scanners of the RF627 series

profile2D_t *SDK::SCANNERS::RF62X::*rf627old*::**get_profile2D**(bool *zero_points* = true, PROTOCOLS *protocol* = PROTOCOLS::CURRENT)

get_profile2D - Get 2D measurement from scanner's data stream

Return ptr to profile2D_t structure if success, else - null

Parameters

- `zero_points`: - include zero points in return profile2D
- `protocol`: - protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example:

```
// Initialize sdk library
sdk_init();

// Create value for scanners vector's type
std::vector<rf627old*> list;
// Search for RF627old devices over network
list = rf627old::search(PROTOCOLS::SERVICE);

// Print count of discovered RF627Old in network by Service Protocol
std::cout << "Discovered: " << list.size() << " RF627Old"   << std::endl;

// Iterate over all discovered RF627Old in network, connect to each of
// them and get a profile.
for(size_t i = 0; i < list.size(); i++)
{
   // Establish connection to the RF627 device by Service Protocol.
   list[i]->connect();

   // Get profile from scanner's data stream by Service Protocol.
   profile2D_t* profile = list[i]->get_profile2D();
   if (profile != nullptr)
   {
      std::cout << "Profile information: "                    << std::endl;
      switch (profile->header.data_type) {
      case (uint8_t)PROFILE_DATA_TYPE::PIXELS:
         std::cout << "* DataType\t: "<< "PIXELS"            << std::endl;
         std::cout << "* Count\t: " << profile->pixels.size()<< std::endl;
         break;
      case (uint8_t)PROFILE_DATA_TYPE::PIXELS_INTRP:
         std::cout << "* DataType\t: "<< "PIXELS_INTRP"      << std::endl;
         std::cout << "* Count\t: " << profile->pixels.size()<< std::endl;
         break;
      case (uint8_t)PROFILE_DATA_TYPE::PROFILE:
         std::cout << "* DataType\t: "<< "PROFILE"           << std::endl;
         std::cout << "* Size\t: "  << profile->points.size()<< std::endl;
         break;
```

(continues on next page)

(continued from previous page)

```
        case (uint8_t)PROFILE_DATA_TYPE::PROFILE_INTRP:
            std::cout << "* DataType\t: "<< "PROFILE_INTRP"     << std::endl;
            std::cout << "* Size\t: "  << profile->points.size()<< std::endl;
            break;
        }
        delete profile;
        std::cout << "Profile was successfully received!"       << std::endl;
        std::cout << "----------------------------------------"<< std::endl;
    }else
    {
        std::cout << "Profile was not received!"                << std::endl;
        std::cout << "----------------------------------------"<< std::endl;
    }

    // Disconnect from scanner.
    list[i]->disconnect();
}

// Cleanup resources allocated with sdk_init()
sdk_cleanup();
```

read_params()

Function for receiving current scanner settings. When this function is called, the SDK reads out all relevant parameters from the scanner, saving them in the form of a «parameter list» for further work.

bool SDK::SCANNERS::RF62X::*rf627old*::**read_params**(PROTOCOLS *protocol* = PROTOCOLS::CURRENT)

read_params - Read parameters from device to internal structure. This structure is accessible via get_params() function

Return true on success

Parameters

- `protocol`: - protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example:

```
// Initialize sdk library
sdk_init();

// Create value for scanners vector's type
std::vector<rf627old*> scanners;
// Search for RF627old devices over network
scanners = rf627old::search(PROTOCOLS::SERVICE);

// Print count of discovered RF627Old in network by Service Protocol
std::cout << "Discovered: " << scanners.size() << " RF627Old"  << std::endl;

// Iterate over all discovered RF627Old in network, connect to each of
// them and read/set parameters.
for(size_t i = 0; i < scanners.size(); i++)
```

(continues on next page)

(continued from previous page)

```
{
   // Establish connection to the RF627 device by Service Protocol.
   scanners[i]->connect();

   // read params from RF627 device by Service Protocol.
   scanners[i]->read_params();

   {
   ...some actions with params
   }

   // Disconnect from scanner.
   scanners[i]->disconnect();
}
```

get_param()

The function of obtaining a specific parameter by its name (key). When this function is called, the SDK searches for the desired parameter from the last read when the function was called read_params. In case the requested parameter is absent in a specific scanner, the function will return null.

param_t *SDK::SCANNERS::RF62X::*rf627old*::**get_param**(std::string *param_name*)

get_param - Search parameters by his name

Return param on success, else - null

Parameters

- param_name: - name of parameter

Usage example:

```
{
...Initialize sdk library
...Search for RF627old
}

// Establish connection to the RF627 device by Service Protocol.
scanners[i]->connect();

// read params from RF627 device by Service Protocol.
scanners[i]->read_params();

// Get parameter of Device Name
param_t* name = scanners[i]->get_param("user_general_deviceName");
if (name->type == "string_t")
{
   std::string str_name = name->get_value<value_str>();
   std::cout << "Current Device Name \t: " << str_name     << std::endl;
}

// Get parameter of Device IP Addr
param_t* ip_addr = scanners[i]->get_param("user_network_ip");
```

(continues on next page)

(continued from previous page)

```
if (ip_addr->type == "u32_arr_t")
{
   std::vector <uint32_t> ip = ip_addr->get_value<array_uint32>();
   std::cout << "Current Device IP\t: ";
   for(auto i: ip) std::cout<<std::to_string(i)<<".";std::cout<<std::endl;
}

// Get parameter of Laser Enabled
param_t* laser_enabled = scanners[i]->get_param("user_laser_enabled");
if (laser_enabled->type == "uint32_t")
{
   bool isEnabled = laser_enabled->get_value<value_uint32>();
   std::cout<<"Current Laser State\t: "<<(isEnabled?"ON":"OFF")<<std::endl;
}
```

For more convenient work with parameters, you can use the corresponding «keys» (parameter name key, parameter type and access to the parameter).

param_t *SDK::SCANNERS::RF62X::*rf627old*::**get_param**(*PARAM_NAME_KEY param_key*)

get_param - Search parameters by his name's key

Return param on success, else - null

Parameters

- `param_name`: - name's key of parameter

To do this, the following `enum` are located in the file `rt62Xtypes.h`:

enum SDK::SCANNERS::RF62X::**PARAM_VALUE_TYPE**

Values:

UNKN_PARAM_TYPE = 0

UINT_PARAM_TYPE = 1

UINT64_PARAM_TYPE = 2

INT_PARAM_TYPE = 3

INT64_PARAM_TYPE = 4

FLOAT_PARAM_TYPE = 5

DOUBLE_PARAM_TYPE = 6

UINT32_ARRAY_PARAM_TYPE = 7

UINT64_ARRAY_PARAM_TYPE = 8

INT32_ARRAY_PARAM_TYPE = 9

INT64_ARRAY_PARAM_TYPE = 10

FLT_ARRAY_PARAM_TYPE = 11

DBL_ARRAY_PARAM_TYPE = 12

STRING_PARAM_TYPE = 13

UNKN_PARAM_TYPE = 0

UINT_PARAM_TYPE = 1

UINT64_PARAM_TYPE = 2

INT_PARAM_TYPE = 3

INT64_PARAM_TYPE = 4

FLOAT_PARAM_TYPE = 5

DOUBLE_PARAM_TYPE = 6

UINT32_ARRAY_PARAM_TYPE = 7

UINT64_ARRAY_PARAM_TYPE = 8

INT32_ARRAY_PARAM_TYPE = 9

INT64_ARRAY_PARAM_TYPE = 10

FLT_ARRAY_PARAM_TYPE = 11

DBL_ARRAY_PARAM_TYPE = 12

STRING_PARAM_TYPE = 13

enum SDK::SCANNERS::RF62X::**PARAM_ACCESS_TYPE**

Values:

PAT_UNKN = 0

PAT_READ_ONLY = 1

PAT_WRITE = 2

PAT_RESTRICTED = 3

PAT_UNKN = 0

PAT_READ_ONLY = 1

PAT_WRITE = 2

PAT_RESTRICTED = 3

enum SDK::SCANNERS::RF62X::**PARAM_NAME_KEY**

Values:

FACT_GENERAL_PROTOCOLREV = 0

FACT_GENERAL_DEVICETYPE

FACT_GENERAL_SERIAL

FACT_GENERAL_PCBSERIAL

FACT_GENERAL_LIFETIME

FACT_GENERAL_WORKTIME

FACT_GENERAL_STARTSCOUNT

FACT_GENERAL_FIRMWAREREV

FACT_GENERAL_HARDWAREREV

FACT_GENERAL_FSBLREV

FACT_GENERAL_CUSTOMERID

FACT_GENERAL_FPGAFREQ

FACT_GENERAL_SMR

FACT_GENERAL_MR

FACT_GENERAL_XSMR

FACT_GENERAL_XEMR

FACT_GENERAL_PIXDIVIDER

FACT_GENERAL_PROFDIVIDER

FACT_GENERAL_OEMDEVNAME

FACT_GENERAL_AUTHSTATUS

FACT_SENSOR_NAME

FACT_SENSOR_WIDTH

FACT_SENSOR_HEIGHT

FACT_SENSOR_PIXFREQ

FACT_SENSOR_FRMCONSTPART

FACT_SENSOR_FRMPERLINEPART

FACT_SENSOR_FPSOREXP

FACT_SENSOR_MINEXPOSURE

FACT_SENSOR_MAXEXPOSURE

FACT_SENSOR_IMGFLIP

FACT_NETWORK_MACADDR

FACT_NETWORK_FORCEAUTONEGTIME

FACT_NETWORK_WEBSOCKSERVICEPORT

FACT_NETWORK_WEBSOCKDATAPORT

FACT_NETWORK_WEBSOCKMATHPORT

FACT_LASER_WAVELENGTH

FACT_LASER_KOEFF1

FACT_LASER_KOEFF2

FACT_LASER_MINVALUE

FACT_LASER_MAXVALUE

FACT_PROFILES_MAXDUMPSIZE

FACT_EIP_IDENTITY_VENDORID

FACT_EIP_IDENTITY_DEVICETYPE

FACT_EIP_IDENTITY_PRODUCTCODE

FACT_EIP_IDENTITY_REV

FACT_EIP_TCPINTRF_CAPABILITY

FACT_EIP_TCPINTRF_PHY_PATHSIZE

FACT_EIP_TCPINTRF_PHY_CLASSID

FACT_EIP_TCPINTRF_PHY_INSTNUMBER

FACT_EIP_TCPINTRF_PHY_ATTRNUMBER

FACT_EIP_INTRFTYPE

FACT_EIP_INTRFCAPABILITY_BITS

FACT_EIP_INTRFCAPABILITY_SPEEDDUPCOUNT

FACT_EIP_INTRFCAPABILITY_SPEEDDUPLEX_SPEED

FACT_EIP_INTRFCAPABILITY_SPEEDDUPLEX_DUPLEX

FACT_SENSOR_ANALOGGAIN

FACT_SENSOR_DIGITALGAIN

FACT_SENSOR_BLACKODD

FACT_SENSOR_BLACKEVEN

FACT_SENSOR_HDRPIECEWISEDIV1

FACT_SENSOR_HDRPIECEWISEDIV2

FACT_SENSOR_INITREGS

USER_GENERAL_DEVICESTATE

USER_GENERAL_DEVICENAME

USER_GENERAL_SAVELOG

USER_SYSMON_FPGATEMP

USER_SYSMON_PARAMSCHANGED

USER_SYSMON_TEMPSENS00

USER_SYSMON_TEMPSENS00MAX

USER_SYSMON_TEMPSENS00MIN

USER_SYSMON_TEMPSENS01

USER_SYSMON_TEMPSENS01MAX

USER_SYSMON_TEMPSENS01MIN

USER_SYSMON_TEMPSENS10

USER_SYSMON_TEMPSENS10MAX

USER_SYSMON_TEMPSENS10MIN

USER_SYSMON_TEMPSENS11

USER_SYSMON_TEMPSENS11MAX

USER_SYSMON_TEMPSENS11MIN

USER_SENSOR_SYNCSOURCE

USER_SENSOR_FRAMERATE

USER_SENSOR_MAXFRAMERATE

USER_SENSOR_EXPOSURECONTROL

USER_SENSOR_EXPOSURE1

USER_SENSOR_EXPOSURE2

USER_SENSOR_EXPOSURE3

USER_SENSOR_EXPOSURE4

USER_SENSOR_MAXEXPOSURE

USER_ROI_ENABLED

USER_ROI_ACTIVE

USER_ROI_POSMODE

USER_ROI_POS

USER_ROI_MAXPOS

USER_ROI_REQPROFSIZE

USER_NETWORK_SPEED

USER_NETWORK_REQUIREDSPEED

USER_NETWORK_AUTONEG

USER_NETWORK_IP

USER_NETWORK_MASK

USER_NETWORK_GATEWAY

USER_NETWORK_HOSTIP

USER_NETWORK_HOSTPORT

USER_NETWORK_WEBPORT

USER_NETWORK_SERVICEPORT

USER_STREAMS_UDPENABLED

USER_STREAMS_FORMAT

USER_STREAMS_INCLUDEINTENSITY

USER_PROCESSING_THRESHOLD

USER_PROCESSING_PROFPERSEC

USER_PROCESSING_MEDIANMODE

USER_PROCESSING_BILATERALMODE

USER_PROCESSING_PEAKMODE

USER_PROCESSING_FLIP

USER_LASER_ENABLED

USER_LASER_VALUE

USER_TRIGGER_SYNC_SOURCE

USER_TRIGGER_SYNC_STRICTENABLED

USER_TRIGGER_SYNC_DIVIDER

USER_TRIGGER_SYNC_DELAY

USER_TRIGGER_COUNTER_TYPE

USER_TRIGGER_COUNTER_MAXVALUEENABLED

USER_TRIGGER_COUNTER_MAXVALUE

USER_TRIGGER_COUNTER_RESETTIMERENABLED

USER_TRIGGER_COUNTER_RESETTIMERVALUE

USER_TRIGGER_COUNTER_VALUE

USER_INPUT1_ENABLED

USER_INPUT1_MODE

USER_INPUT2_ENABLED

USER_INPUT2_MODE

USER_INPUT3_ENABLED

USER_INPUT3_MODE

USER_INPUT1_SAMPLES

USER_INPUT2_SAMPLES

USER_INPUT3_SAMPLES

USER_OUTPUT1_ENABLED

USER_OUTPUT1_MODE

USER_OUTPUT1_PULSEWIDTH

USER_OUTPUT2_ENABLED

USER_OUTPUT2_MODE

USER_OUTPUT2_PULSEWIDTH

USER_DUMP_ENABLED

USER_DUMP_CAPACITY

USER_DUMP_SIZE

USER_DUMP_TIMESTAMP

USER_DUMP_VIEW3D_MOTIONTYPE

USER_DUMP_VIEW3D_YSOURCE

USER_DUMP_VIEW3D_YSTEP

USER_DUMP_VIEW3D_PAINTMODE

USER_DUMP_VIEW3D_DECIMATION

USER_EIP_TCPPORT

USER_EIP_UDPPORT

USER_EIP_TCP_TTL

USER_EIP_TCP_TIMEOUT

USER_EIP_TCP_MULTICAST_ALLOC

USER_EIP_TCP_MULTICAST_NUM

USER_EIP_TCP_MULTICAST_ADDR

USER_COMPATIBILITY_RF625ENABLED

USER_COMPATIBILITY_RF625TCPPORT

USER_SENSOR_DOUBLESPEEDENABLED

USER_SENSOR_EDRTYPE

USER_SENSOR_EDRCOLUMNDIVIDER

USER_STREAMS_POINTSCOUNT

USER_ROI_SIZE

FACT_GENERAL_PROTOCOLREV = 0

FACT_GENERAL_DEVICETYPE

FACT_GENERAL_SERIAL

FACT_GENERAL_PCBSERIAL

FACT_GENERAL_LIFETIME

FACT_GENERAL_WORKTIME

FACT_GENERAL_STARTSCOUNT

FACT_GENERAL_FIRMWAREREV

FACT_GENERAL_HARDWAREREV

FACT_GENERAL_FSBLREV

FACT_GENERAL_CUSTOMERID

FACT_GENERAL_FPGAFREQ

FACT_GENERAL_SMR

FACT_GENERAL_MR

FACT_GENERAL_XSMR

FACT_GENERAL_XEMR

FACT_GENERAL_PIXDIVIDER

FACT_GENERAL_PROFDIVIDER

FACT_GENERAL_OEMDEVNAME

FACT_GENERAL_AUTHSTATUS

FACT_SENSOR_NAME

FACT_SENSOR_WIDTH

FACT_SENSOR_HEIGHT

FACT_SENSOR_PIXFREQ

FACT_SENSOR_FRMCONSTPART

FACT_SENSOR_FRMPERLINEPART

FACT_SENSOR_FPSOREXP

FACT_SENSOR_MINEXPOSURE

FACT_SENSOR_MAXEXPOSURE

FACT_SENSOR_IMGFLIP

FACT_NETWORK_MACADDR

FACT_NETWORK_FORCEAUTONEGTIME

FACT_NETWORK_WEBSOCKSERVICEPORT

FACT_NETWORK_WEBSOCKDATAPORT

FACT_NETWORK_WEBSOCKMATHPORT

FACT_LASER_WAVELENGTH

FACT_LASER_KOEFF1

FACT_LASER_KOEFF2

FACT_LASER_MINVALUE

FACT_LASER_MAXVALUE

FACT_PROFILES_MAXDUMPSIZE

FACT_EIP_IDENTITY_VENDORID

FACT_EIP_IDENTITY_DEVICETYPE

FACT_EIP_IDENTITY_PRODUCTCODE

FACT_EIP_IDENTITY_REV

FACT_EIP_TCPINTRF_CAPABILITY

FACT_EIP_TCPINTRF_PHY_PATHSIZE

FACT_EIP_TCPINTRF_PHY_CLASSID

FACT_EIP_TCPINTRF_PHY_INSTNUMBER

FACT_EIP_TCPINTRF_PHY_ATTRNUMBER

FACT_EIP_INTRFTYPE

FACT_EIP_INTRFCAPABILITY_BITS

FACT_EIP_INTRFCAPABILITY_SPEEDDUPCOUNT

FACT_EIP_INTRFCAPABILITY_SPEEDDUPLEX_SPEED

FACT_EIP_INTRFCAPABILITY_SPEEDDUPLEX_DUPLEX

FACT_SENSOR_ANALOGGAIN

FACT_SENSOR_DIGITALGAIN

FACT_SENSOR_BLACKODD

FACT_SENSOR_BLACKEVEN

FACT_SENSOR_HDRPIECEWISEDIV1

FACT_SENSOR_HDRPIECEWISEDIV2

FACT_SENSOR_INITREGS

USER_GENERAL_DEVICESTATE

USER_GENERAL_DEVICENAME

USER_GENERAL_SAVELOG

USER_SYSMON_FPGATEMP

USER_SYSMON_PARAMSCHANGED

USER_SYSMON_TEMPSENS00

USER_SYSMON_TEMPSENS00MAX

USER_SYSMON_TEMPSENS00MIN

USER_SYSMON_TEMPSENS01

USER_SYSMON_TEMPSENS01MAX

USER_SYSMON_TEMPSENS01MIN

USER_SYSMON_TEMPSENS10

USER_SYSMON_TEMPSENS10MAX

USER_SYSMON_TEMPSENS10MIN

USER_SYSMON_TEMPSENS11

USER_SYSMON_TEMPSENS11MAX

USER_SYSMON_TEMPSENS11MIN

USER_SENSOR_SYNCSOURCE

USER_SENSOR_FRAMERATE

USER_SENSOR_MAXFRAMERATE

USER_SENSOR_EXPOSURECONTROL

USER_SENSOR_EXPOSURE1

USER_SENSOR_EXPOSURE2

USER_SENSOR_EXPOSURE3

USER_SENSOR_EXPOSURE4

USER_SENSOR_MAXEXPOSURE

USER_ROI_ENABLED

USER_ROI_ACTIVE

USER_ROI_POSMODE

USER_ROI_POS

USER_ROI_MAXPOS

USER_ROI_REQPROFSIZE

USER_NETWORK_SPEED

USER_NETWORK_REQUIREDSPEED

USER_NETWORK_AUTONEG

USER_NETWORK_IP

USER_NETWORK_MASK

USER_NETWORK_GATEWAY

USER_NETWORK_HOSTIP

USER_NETWORK_HOSTPORT

USER_NETWORK_WEBPORT

USER_NETWORK_SERVICEPORT

USER_STREAMS_UDPENABLED

USER_STREAMS_FORMAT

USER_STREAMS_INCLUDEINTENSITY

USER_PROCESSING_THRESHOLD

USER_PROCESSING_PROFPERSEC

USER_PROCESSING_MEDIANMODE

USER_PROCESSING_BILATERALMODE

USER_PROCESSING_PEAKMODE

USER_PROCESSING_FLIP

USER_LASER_ENABLED

USER_LASER_VALUE

USER_TRIGGER_SYNC_SOURCE

USER_TRIGGER_SYNC_STRICTENABLED

USER_TRIGGER_SYNC_DIVIDER

USER_TRIGGER_SYNC_DELAY

USER_TRIGGER_COUNTER_TYPE

USER_TRIGGER_COUNTER_MAXVALUEENABLED

USER_TRIGGER_COUNTER_MAXVALUE

USER_TRIGGER_COUNTER_RESETTIMERENABLED

USER_TRIGGER_COUNTER_RESETTIMERVALUE

USER_TRIGGER_COUNTER_VALUE

USER_INPUT1_ENABLED

USER_INPUT1_MODE

USER_INPUT2_ENABLED

USER_INPUT2_MODE

USER_INPUT3_ENABLED

USER_INPUT3_MODE

USER_INPUT1_SAMPLES

USER_INPUT2_SAMPLES

USER_INPUT3_SAMPLES

USER_OUTPUT1_ENABLED

USER_OUTPUT1_MODE

USER_OUTPUT1_PULSEWIDTH

USER_OUTPUT2_ENABLED

USER_OUTPUT2_MODE

USER_OUTPUT2_PULSEWIDTH

USER_DUMP_ENABLED

USER_DUMP_CAPACITY

USER_DUMP_SIZE

USER_DUMP_TIMESTAMP

USER_DUMP_VIEW3D_MOTIONTYPE

USER_DUMP_VIEW3D_YSOURCE

USER_DUMP_VIEW3D_YSTEP

USER_DUMP_VIEW3D_PAINTMODE

USER_DUMP_VIEW3D_DECIMATION

USER_EIP_TCPPORT

USER_EIP_UDPPORT

USER_EIP_TCP_TTL

USER_EIP_TCP_TIMEOUT

USER_EIP_TCP_MULTICAST_ALLOC

USER_EIP_TCP_MULTICAST_NUM

USER_EIP_TCP_MULTICAST_ADDR

USER_COMPATIBILITY_RF625ENABLED

USER_COMPATIBILITY_RF625TCPPORT

USER_SENSOR_DOUBLESPEEDENABLED

USER_SENSOR_EDRTYPE

USER_SENSOR_EDRCOLUMNDIVIDER

USER_STREAMS_POINTSCOUNT

USER_ROI_SIZE

Key usage example:

```
{
...Initialize sdk library
...Search for RF627old
}

// Establish connection to the RF627 device by Service Protocol.
scanners[i]->connect();

// read params from RF627 device by Service Protocol.
scanners[i]->read_params();

// Get parameter of Device Name
param_t* name = scanners[i]->get_param(PARAM_NAME_KEY::USER_GENERAL_
↪DEVICENAME);
if (name->type == param_value_types[(int)PARAM_VALUE_TYPE::STRING_PARAM_TYPE])
{
   std::string str_name = name->get_value<value_str>();
   std::cout << "Current Device Name \t: " << str_name     << std::endl;
}

// Get parameter of Device IP Addr
param_t* ip_addr = scanners[i]->get_param(PARAM_NAME_KEY::USER_NETWORK_IP);
if (ip_addr->type == param_value_types[(int)PARAM_VALUE_TYPE::UINT32_ARRAY_
↪PARAM_TYPE])
{
   std::vector <uint32_t> ip = ip_addr->get_value<array_uint32>();
   std::cout << "Current Device IP\t: ";
}

// Get parameter of Laser Enabled
param_t* laser_enabled = scanners[i]->get_param(PARAM_NAME_KEY::USER_LASER_
↪ENABLED);
if (laser_enabled->type == param_value_types[(int)PARAM_VALUE_TYPE::UINT_
↪PARAM_TYPE])
{
   bool isEnabled = laser_enabled->get_value<value_uint32>();
   std::cout<<"Current Laser State\t: "<<(isEnabled?"ON":"OFF")<<std::endl;
}
```

For a more detailed description of each parameter and its properties, see *PARAMETERS*

set_param()

The function of setting a specific parameter. When this function is called, the transferred parameter is set in the local parameter list in the SDK. To send changes to the scanner, you must call the `write_params` function.

bool `SDK::SCANNERS::RF62X::rf627old::`**`set_param`**(param_t **param*)

set_param - set parameter

Return true on success, else - false

Parameters

- `param`: - prt to parameter

Usage example:

```
{
...Initialize sdk library
...Search for RF627old
}

// Establish connection to the RF627 device by Service Protocol.
scanners[i]->connect();

// read params from RF627 device by Service Protocol.
scanners[i]->read_params();

// Get parameter of Device Name
param_t* name = scanners[i]->get_param(PARAM_NAME_KEY::USER_GENERAL_
↪DEVICENAME);
if (name->type == param_value_types[(int)PARAM_VALUE_TYPE::STRING_PARAM_TYPE])
{
   std::string str_name = name->get_value<value_str>();
   std::cout << "Current Device Name \t: " << str_name     << std::endl;

   // Add "_TEST" to the ending of the current name
   str_name += "_TEST";
   name->set_value<value_str>(str_name);
   std::cout << "New Device Name \t: " << str_name         << std::endl;
   std::cout << "----------------------------------------"<< std::endl;

   scanners[i]->set_param(name);
}

// Get parameter of Device IP Addr
param_t* ip_addr = scanners[i]->get_param(PARAM_NAME_KEY::USER_NETWORK_IP);
if (ip_addr->type == param_value_types[(int)PARAM_VALUE_TYPE::UINT32_ARRAY_
↪PARAM_TYPE])
{
   std::vector <uint32_t> ip = ip_addr->get_value<array_uint32>();
   std::cout << "Current Device IP\t: ";
   for(auto i: ip) std::cout<<std::to_string(i)<<".";std::cout<<std::endl;

   // Change last digit of IP address (e.g. 192.168.1.30 -> 192.168.1.31)
   ip[3]++;
   ip_addr->set_value<array_uint32>(ip);
   std::cout << "New Device IP\t: ";
   for(auto i: ip) std::cout<<std::to_string(i)<<".";std::cout<<std::endl;
   std::cout << "----------------------------------------"<< std::endl;

   scanners[i]->set_param(ip_addr);
}

// Get parameter of Laser Enabled
param_t* laser_enabled = scanners[i]->get_param(PARAM_NAME_KEY::USER_LASER_
↪ENABLED);
if (laser_enabled->type == param_value_types[(int)PARAM_VALUE_TYPE::UINT_
↪PARAM_TYPE])
{
   bool isEnabled = laser_enabled->get_value<value_uint32>();
   std::cout<<"Current Laser State\t: "<<(isEnabled?"ON":"OFF")<<std::endl;
```

(continues on next page)

(continued from previous page)

```
    // Change the current state to the opposite
    isEnabled = !isEnabled;
    laser_enabled->set_value<value_uint32>(!isEnabled);
    std::cout<<"New Laser State\t: "<<(isEnabled?"ON":"OFF")<<std::endl;
    std::cout << "----------------------------------------"<< std::endl;

    scanners[i]->set_param(laser_enabled);
}

//  Write changes parameters to the device's memory
scanners[i]->write_params();

// Disconnect from scanner.
scanners[i]->disconnect();
```

write_params()

The function of writing local parameters from the SDK to the scanner. When this function is called, a list of local parameters is sent from the SDK to the scanner.

bool `SDK::SCANNERS::RF62X::rf627old::write_params`(PROTOCOLS *protocol* = PROTOCOLS::CURRENT)

write_params - Write current parameters to device's memory

Return true on success

Parameters

- `protocol`: - protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example:

```
{
...Initialize sdk library
...Search for RF627old
}

// Establish connection to the RF627 device by Service Protocol.
scanners[i]->connect();

// Read params from RF627 device by Service Protocol.
scanners[i]->read_params();

{
...Some steps to change scanner's parameters
}

//  Write changes parameters to the device's memory
scanners[i]->write_params();

// Disconnect from scanner.
scanners[i]->disconnect();
```

send_cmd()

Function for sending commands to the scanner

bool SDK::SCANNERS::RF62X::*rf627old*::**send_cmd**(**const** char ***command_name*, int *arg_count*, ...)

For a more detailed description of the commands and their properties, see *General device commands*

2.4 Wrapper API in C#

This wrapper is a .NET library written in C#, which makes it easy to develop applications in C#, Visual Basic .NET, C++/CLI, and JScript .NET

To use it in .NET projects, the developer needs to collect or download the dynamic program library **rf62Xsdk.dll**, then add the library to the project references and collect or download the library **rf62Xcore.dll** by adding it in the folder to the project executable file.

To download the library, see *the latest releases of the wrapper in C#*
To compile the library, see *compile and run the wrapper in C#*.

2.4.1 SDK initialization

The file `rf62Xsdk.cs` is the main file of the program interface (API) for developing C# programs and defines the functionality of the wrapper library for rf62Xcore. `rf62Xsdk.cs` contains the following set of classes and functions for initializing the SDK:

SdkInit()

SDK initialization function. Must be called once before further calls to any library functions:

static bool SDK.SCANNERS.RF62X.SdkInit()
SdkInit - Initialize sdk library

Must be called once before further calls to any library functions

Return true if success.

SdkCleanup()

Function for cleaning resources allocated using the `SdkInit` function:

static void SDK.SCANNERS.RF62X.SdkCleanup()
SdkCleanup - Cleanup resources allocated with sdk_init() function

SdkVersion()

Function to get the current version of the SDK:

static string SDK.SCANNERS.RF62X.SdkVersion()
SdkVersion - Return info about SDK version

Return SDK version

2.4.2 Class rf627old

This class is defined in the file `rf62Xsdk.cs` and provides an interface for working with RF627Old series scanners

class RF627old
RF627old - This class is the main interface for working with RF627-old series scanners.

Search()

Function to search for RF627 devices available on the network

static List<RF627old> SDK.SCANNERS.RF62X.RF627old.Search(PROTOCOLS_TYPES protoco
Search for *RF627old* devices over network

Return List of *RF627old* devices

Parameters

- `protocol`: protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example:

```
// Start initialization of the library core
RF62X.SdkInit();

// Print return rf62X sdk version
Console.WriteLine("SDK version: {0}", RF62X.SdkVersion());
Console.WriteLine("========================================");

// Search for RF627old devices over network
List<RF62X.RF627old> Scanners = RF62X.RF627old.Search();

// Print count of discovered RF627Old in network by Service Protocol
Console.WriteLine("Discovered {0} scanners", Scanners.Count);
```

GetInfo()

Function for receiving information about the scanner from the Hello packet

HelloInfo SDK.SCANNERS.RF62X.RF627old.GetInfo(PROTOCOLS_TYPES protocol = PROTOCOLS_

Get information about scanner from hello packet

Return Hello_info on success

Parameters

- `protocol`: protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example:

```
// Start initialization of the library core
RF62X.SdkInit();

// Search for RF627old devices over network
List<RF62X.RF627old> Scanners = RF62X.RF627old.Search();

// Print count of discovered RF627Old in network by Service Protocol
Console.WriteLine("Discovered {0} scanners", Scanners.Count);

for (int i = 0; i < Scanners.Count; i++)
{
   RF62X.HelloInfo info = Scanners[i].GetInfo();

   Console.WriteLine("\n\n\nID scanner's list: {0}", i);
   Console.WriteLine("----------------------------------------");
   Console.WriteLine("Device information: ");
   Console.WriteLine("* Name\t: {0}", info.device_name);
   Console.WriteLine("* Serial\t: {0}", info.serial_number);
   Console.WriteLine("* IP Addr\t: {0}", info.ip_address);
   Console.WriteLine("* MAC Addr\t: {0}", info.mac_address);

   Console.WriteLine("Working ranges: ");
   Console.WriteLine("* Zsmr, mm\t: {0}", info.z_smr);
   Console.WriteLine("* Zmr , mm\t: {0}", info.z_mr);
   Console.WriteLine("* Xsmr, mm\t: {0}", info.x_smr);
   Console.WriteLine("* Xemr, mm\t: {0}", info.x_emr);

   Console.WriteLine("\nVersions: ");
   Console.WriteLine("* Firmware\t: {0}", info.firmware_version);
   Console.WriteLine("* Hardware\t: {0}", info.hardware_version);
   Console.WriteLine("----------------------------------------");
}

// Cleanup resources allocated with sdk_init()
RF62X.SdkCleanup();
```

Connect()

Function for establishing connection with the RF627 Series Scanner

bool SDK.SCANNERS.RF62X.RF627old.Connect(PROTOCOLS_TYPES protocol = PROTOCOLS_TY
Establish connection to the *RF627old* device

Return true on success

Parameters

- `protocol`: protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example:

```
// Start initialization of the library core
RF62X.SdkInit();

// Search for RF627old devices over network
List<RF62X.RF627old> Scanners = RF62X.RF627old.Search();

// Print count of discovered RF627Old in network by Service Protocol
Console.WriteLine("Discovered {0} scanners", Scanners.Count);

for (int i = 0; i < Scanners.Count; i++)
{
   // Establish connection to the RF627 device by Service Protocol.
   if (Scanners[i].Connect())
      Console.WriteLine("Connected to scanner №{0} successfully", i);
}

// Cleanup resources allocated with sdk_init()
RF62X.SdkCleanup();
```

Disconnect()

Function to close a previously established connection with the RF627 Series Scanner

bool SDK.SCANNERS.RF62X.RF627old.Disconnect(PROTOCOLS_TYPES protocol = PROTOCOLS
Close connection to the device

Return true on success

Parameters

- `protocol`: protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example:

```
// Start initialization of the library core
RF62X.SdkInit();

// Search for RF627old devices over network
List<RF62X.RF627old> Scanners = RF62X.RF627old.Search();

// Print count of discovered RF627Old in network by Service Protocol
Console.WriteLine("Discovered {0} scanners", Scanners.Count);
```

(continues on next page)

(continued from previous page)

```
// Establish connection to the RF627 device by Service Protocol.
for (int i = 0; i < Scanners.Count; i++)
   Scanners[i].Connect();

{
...some actions with scanners
}

for (int i = 0; i < Scanners.Count; i++)
   Scanners[i].Disconnect();
```

GetProfile()

Function for receiving a profile from scanners of the RF627 series

Profile SDK.SCANNERS.RF62X.RF627old.GetProfile(PROTOCOLS_TYPES protocol = PROTOCOLS

Get 2D measurement from scanner's data stream

Return Profile

Parameters

- `protocol`: protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example:

```
// Start initialization of the library core
RF62X.SdkInit();

// Search for RF627old devices over network
List<RF62X.RF627old> Scanners = RF62X.RF627old.Search();

// Print count of discovered RF627Old in network by Service Protocol
Console.WriteLine("Discovered {0} scanners", Scanners.Count);

// foreach over an scanners list
for (int i = 0; i < Scanners.Count; i++)
{
   // Establish connection to the RF627 device by Service Protocol.
   Scanners[i].Connect();

   // Get profile from scanner's data stream by Service Protocol.
   RF62X.Profile profile = Scanners[i].GetProfile();
   if (profile.header != null)
   {
      Console.WriteLine("Profile information: ");
      switch (profile.header.data_type)
      {
      case RF62X.PROFILE_TYPE.PIXELS_NORMAL:
         Console.WriteLine("* DataType\t: PIXELS");
         Console.WriteLine("* Count\t: {0}", profile.pixels.Count);
         break;
      case RF62X.PROFILE_TYPE.PROFILE_NORMAL:
         Console.WriteLine("* DataType\t: PROFILE");
```

(continues on next page)

(continued from previous page)

```
            Console.WriteLine("* Size\t: {0}", profile.points.Count);
            break;
         case RF62X.PROFILE_TYPE.PIXELS_INTERPOLATED:
            Console.WriteLine("* DataType\t: PIXELS");
            Console.WriteLine("* Count\t: {0}", profile.pixels.Count);
            break;
         case RF62X.PROFILE_TYPE.PROFILE_INTERPOLATED:
            Console.WriteLine("* DataType\t: PROFILE");
            Console.WriteLine("* Size\t: {0}", profile.points.Count);
            break;
         default:
            break;
         }
         Console.WriteLine("Profile was successfully received!");
         Console.WriteLine("----------------------------------------");
      }else
      {
         Console.WriteLine("Profile was not received!");
         Console.WriteLine("----------------------------------------");
      }

      // Disconnect from scanner.
      Scanners[i].Disconnect();
   }

   // Cleanup resources allocated with sdk_init()
   RF62X.SdkCleanup();
```

ReadParams()

Function for receiving current scanner settings. When this function is called, the SDK reads out all relevant parameters from the scanner, saving them in the form of a «parameter list» for further work.

bool SDK.SCANNERS.RF62X.RF627old.ReadParams(PROTOCOLS_TYPES protocol = PROTOCOLS

Read parameters from device to internal structure. This structure is accessible via *GetParam()* functions

Return true on success

Parameters

- `protocol`: protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example:

```
// Start initialization of the library core
RF62X.SdkInit();

// Search for RF627old devices over network
List<RF62X.RF627old> Scanners = RF62X.RF627old.Search();

// Print count of discovered RF627Old in network by Service Protocol
Console.WriteLine("Discovered {0} scanners", Scanners.Count);
```

(continues on next page)

(continued from previous page)

```
// foreach over an scanners list
for (int i = 0; i < Scanners.Count; i++)
{
   // Establish connection to the RF627 device by Service Protocol.
   Scanners[i].Connect();

   // read params from RF627 device by Service Protocol.
   Scanners[i].ReadParams();

   {
   ...some actions with params
   }

   // Disconnect from scanner.
   Scanners[i].Disconnect();
}
```

GetParam()

The function of obtaining a specific parameter by its name (key). When this function is called, the SDK searches for the desired parameter from the last read when the function was called ReadParams. In case the requested parameter is absent in a specific scanner, the function will return null.

dynamic SDK.SCANNERS.RF62X.RF627old.GetParam(string nameKey)

Search parameters by his name

Return param on success, else - null

Parameters

- `nameKey`: name of parameter

Usage example:

```
{
...Initialize sdk library
...Search for RF627old
}

// Establish connection to the RF627 device by Service Protocol.
Scanners[i].Connect();

// read params from RF627 device by Service Protocol.
Scanners[i].ReadParams();

// Get parameter of Device Name
RF62X.Param<string> name = Scanners[i].GetParam("user_general_deviceName");
if (name != null)
{
   string strName = name.GetValue();
   Console.WriteLine("\n\nCurrent Device Name \t: {0}", strName);
}

// Get parameter of Device IP Addr
```

(continues on next page)

(continued from previous page)

```
RF62X.Param<List<uint>> ipAddr = Scanners[i].GetParam("user_network_ip");
if (ipAddr != null)
{
   List<uint> ip = ipAddr.GetValue();
   Console.WriteLine("Current Device IP Addr\t: {0}.{1}.{2}.{3}", ip[0],␣
↪ip[1], ip[2], ip[3]);
}

// Get parameter of Laser Enabled
RF62X.Param<uint> laserEnabled = Scanners[i].GetParam("user_laser_enabled");
if (laserEnabled != null)
{
   bool isLaserEnabled = Convert.ToBoolean(laserEnabled.GetValue());
   Console.WriteLine("Current Laser State\t: {0}", isLaserEnabled ? "ON" :
↪"OFF");
}
```

For more convenient work with parameters, you can use the corresponding «keys» (parameter name key, parameter type and access to the parameter).

dynamic SDK.SCANNERS.RF62X.RF627old.GetParam(Params.Description paramInfo)

Search parameters by his info

Return param on success, else - null

Parameters

- `paramInfo`: info of parameter

To do this, in the file `rt62Xtypes.cs` there are `class`:

class Params

Public Static Functions

static List<Description> SDK.SCANNERS.RF62X.Params.GetParamsDescriptionList(

class Description

Public Functions

Type SDK.SCANNERS.RF62X.Params.Description.GetParamType()

Public Members

string SDK.SCANNERS.RF62X.Params.Description.Key

string SDK.SCANNERS.RF62X.Params.Description.Type

class User

class Compatibility

Property

```
property SDK::SCANNERS::RF62X::Params::rf625Enabled
property SDK::SCANNERS::RF62X::Params::rf625TCPPort
class Dump
```

Property

```
property SDK::SCANNERS::RF62X::Params::enabled
property SDK::SCANNERS::RF62X::Params::capacity
property SDK::SCANNERS::RF62X::Params::size
property SDK::SCANNERS::RF62X::Params::timeStamp
class View3D
```

Property

```
property SDK::SCANNERS::RF62X::Params::motionType
property SDK::SCANNERS::RF62X::Params::ySource
property SDK::SCANNERS::RF62X::Params::yStep
property SDK::SCANNERS::RF62X::Params::paintMode
property SDK::SCANNERS::RF62X::Params::decimation
class Eip
```

Property

```
property SDK::SCANNERS::RF62X::Params::tcpPort
property SDK::SCANNERS::RF62X::Params::udpPort
property SDK::SCANNERS::RF62X::Params::tcpTTL
property SDK::SCANNERS::RF62X::Params::tcpTimeout
property SDK::SCANNERS::RF62X::Params::multicastAlloc
property SDK::SCANNERS::RF62X::Params::multicastNum
property SDK::SCANNERS::RF62X::Params::multicastAddr
class General
```

Property

```
property SDK::SCANNERS::RF62X::Params::deviceState
property SDK::SCANNERS::RF62X::Params::deviceName
property SDK::SCANNERS::RF62X::Params::saveLog
```

```
class Inputs1
```

Property

```
property SDK::SCANNERS::RF62X::Params::enabled
property SDK::SCANNERS::RF62X::Params::mode
property SDK::SCANNERS::RF62X::Params::samples
```

```
class Inputs2
```

Property

```
property SDK::SCANNERS::RF62X::Params::enabled
property SDK::SCANNERS::RF62X::Params::mode
property SDK::SCANNERS::RF62X::Params::samples
```

```
class Inputs3
```

Property

```
property SDK::SCANNERS::RF62X::Params::enabled
property SDK::SCANNERS::RF62X::Params::mode
property SDK::SCANNERS::RF62X::Params::samples
```

```
class Laser
```

Property

```
property SDK::SCANNERS::RF62X::Params::enabled
property SDK::SCANNERS::RF62X::Params::value
property SDK::SCANNERS::RF62X::Params::preset
property SDK::SCANNERS::RF62X::Params::params_mask
```

```
class NetWork
```

Property

```
property SDK::SCANNERS::RF62X::Params::speed
property SDK::SCANNERS::RF62X::Params::requiredSpeed
property SDK::SCANNERS::RF62X::Params::autoNeg
property SDK::SCANNERS::RF62X::Params::ip
property SDK::SCANNERS::RF62X::Params::mask
property SDK::SCANNERS::RF62X::Params::gateway
property SDK::SCANNERS::RF62X::Params::hostIP
property SDK::SCANNERS::RF62X::Params::hostPort
property SDK::SCANNERS::RF62X::Params::webPort
property SDK::SCANNERS::RF62X::Params::servicePort
```

```
class Outputs1
```

Property

```
property SDK::SCANNERS::RF62X::Params::enabled
property SDK::SCANNERS::RF62X::Params::mode
property SDK::SCANNERS::RF62X::Params::pulseWidth
```

```
class Outputs2
```

Property

```
property SDK::SCANNERS::RF62X::Params::enabled
property SDK::SCANNERS::RF62X::Params::mode
property SDK::SCANNERS::RF62X::Params::pulseWidth
```

```
class Processing
```

Property

```
property SDK::SCANNERS::RF62X::Params::threshold
property SDK::SCANNERS::RF62X::Params::profPerSec
property SDK::SCANNERS::RF62X::Params::medianMode
property SDK::SCANNERS::RF62X::Params::bilateralMode
property SDK::SCANNERS::RF62X::Params::peakMode
property SDK::SCANNERS::RF62X::Params::flip
```

```
class Roi
```

Property

```
property SDK::SCANNERS::RF62X::Params::enable
property SDK::SCANNERS::RF62X::Params::active
property SDK::SCANNERS::RF62X::Params::posMode
property SDK::SCANNERS::RF62X::Params::pos
property SDK::SCANNERS::RF62X::Params::maxPos
property SDK::SCANNERS::RF62X::Params::reqProfSize
property SDK::SCANNERS::RF62X::Params::size
```

```
class Sensor
```

Property

```
property SDK::SCANNERS::RF62X::Params::syncSource
property SDK::SCANNERS::RF62X::Params::framerate
property SDK::SCANNERS::RF62X::Params::maxFramerate
property SDK::SCANNERS::RF62X::Params::exposureControl
property SDK::SCANNERS::RF62X::Params::user_sensor_exposure1
property SDK::SCANNERS::RF62X::Params::user_sensor_exposure2
property SDK::SCANNERS::RF62X::Params::user_sensor_exposure3
property SDK::SCANNERS::RF62X::Params::user_sensor_exposure4
property SDK::SCANNERS::RF62X::Params::maxExposure
property SDK::SCANNERS::RF62X::Params::doubleSpeedEnabled
property SDK::SCANNERS::RF62X::Params::edrType
property SDK::SCANNERS::RF62X::Params::edrColumnDivider
```

```
class Streams
```

Property

```
property SDK::SCANNERS::RF62X::Params::udpEnable
property SDK::SCANNERS::RF62X::Params::format
property SDK::SCANNERS::RF62X::Params::includeIntensity
property SDK::SCANNERS::RF62X::Params::pointsCount
```

```
class SysMon
```

Property

```
property SDK::SCANNERS::RF62X::Params::fpgaTemp
property SDK::SCANNERS::RF62X::Params::paramsChanged
property SDK::SCANNERS::RF62X::Params::tempSens00
property SDK::SCANNERS::RF62X::Params::tempSens00Max
property SDK::SCANNERS::RF62X::Params::tempSens00Min
property SDK::SCANNERS::RF62X::Params::tempSens01
property SDK::SCANNERS::RF62X::Params::tempSens01Max
property SDK::SCANNERS::RF62X::Params::tempSens01Min
property SDK::SCANNERS::RF62X::Params::tempSens10
property SDK::SCANNERS::RF62X::Params::tempSens10Max
property SDK::SCANNERS::RF62X::Params::tempSens10Min
property SDK::SCANNERS::RF62X::Params::tempSens11
property SDK::SCANNERS::RF62X::Params::tempSens11Max
property SDK::SCANNERS::RF62X::Params::tempSens11Min
class Trigger

class Counter
```

Property

```
property SDK::SCANNERS::RF62X::Params::type
property SDK::SCANNERS::RF62X::Params::maxValueEnabled
property SDK::SCANNERS::RF62X::Params::maxValue
property SDK::SCANNERS::RF62X::Params::resetTimerEnabled
property SDK::SCANNERS::RF62X::Params::resetTimerValue
property SDK::SCANNERS::RF62X::Params::value
class Sync
```

Property

```
property SDK::SCANNERS::RF62X::Params::source
property SDK::SCANNERS::RF62X::Params::strictEnabled
property SDK::SCANNERS::RF62X::Params::divider
property SDK::SCANNERS::RF62X::Params::delay
```

Key usage example:

```
{
...Initialize sdk library
...Search for RF627old
}

// Establish connection to the RF627 device by Service Protocol.
Scanners[i].Connect();

// read params from RF627 device by Service Protocol.
Scanners[i].ReadParams();

// Get parameter of Device Name
RF62X.Param<string> name = Scanners[i].GetParam(RF62X.Params.User.General.
↪deviceName);
if (name != null)
{
   string strName = name.GetValue();
   Console.WriteLine("\n\nCurrent Device Name \t: {0}", strName);
}

// Get parameter of Device IP Addr
RF62X.Param<List<uint>> ipAddr = Scanners[i].GetParam(RF62X.Params.User.
↪NetWork.ip);
if (ipAddr != null)
{
   List<uint> ip = ipAddr.GetValue();
   Console.WriteLine("Current Device IP Addr\t: {0}.{1}.{2}.{3}", ip[0],␣
↪ip[1], ip[2], ip[3]);
}

// Get parameter of Laser Enabled
RF62X.Param<uint> laserEnabled = Scanners[i].GetParam(RF62X.Params.User.Laser.
↪enabled);
if (laserEnabled != null)
{
   bool isLaserEnabled = Convert.ToBoolean(laserEnabled.GetValue());
   Console.WriteLine("Current Laser State\t: {0}", isLaserEnabled ? "ON" :
↪"OFF");
}
```

For a more detailed description of each parameter and its properties, see *PARAMETERS*

SetParam()

The function of setting a specific parameter. When this function is called, the transferred parameter is set in the local parameter list in the SDK. To send changes to the scanner, you must call the `write_params` function.

bool SDK.SCANNERS.RF62X.RF627old.SetParam(dynamic param)

Update parameter in internal structure

Return true on success, else - false

Parameters

- `param`: Updated parameter

Usage example:

```
{
...Initialize sdk library
...Search for RF627old
}

// Establish connection to the RF627 device by Service Protocol.
Scanners[i].Connect();

// read params from RF627 device by Service Protocol.
Scanners[i].ReadParams();

// Get parameter of Device Name
RF62X.Param<string> name = Scanners[i].GetParam(RF62X.Params.User.General.
↪deviceName);
if (name != null)
{
   string strName = name.GetValue();
   Console.WriteLine("\n\nCurrent Device Name \t: {0}", strName);

   // Add "_TEST" to the ending of the current name
   strName += "_TEST";
   name.SetValue(strName);
   Console.WriteLine("New Device Name \t: {0}", strName);
   Console.WriteLine("----------------------------------------");

   Scanners[i].SetParam(name);
}

// Get parameter of Device IP Addr
RF62X.Param<List<uint>> ipAddr = Scanners[i].GetParam(RF62X.Params.User.
↪NetWork.ip);
if (ipAddr != null)
{
   List<uint> ip = ipAddr.GetValue();
   Console.WriteLine("Current Device IP Addr\t: {0}.{1}.{2}.{3}", ip[0],␣
↪ip[1], ip[2], ip[3]);

   // Change last digit of IP address (e.g. 192.168.1.30 -> 192.168.1.31)
   ip[3]++;
   ipAddr.SetValue(ip);
   Console.WriteLine("New Device IP Addr\t: {0}.{1}.{2}.{3}", ip[0], ip[1],␣
↪ip[2], ip[3]);
   Console.WriteLine("----------------------------------------");

   Scanners[i].SetParam(ipAddr);
}

// Get parameter of Laser Enabled
RF62X.Param<uint> laserEnabled = Scanners[i].GetParam(RF62X.Params.User.Laser.
↪enabled);
if (laserEnabled != null)
{
   bool isLaserEnabled = Convert.ToBoolean(laserEnabled.GetValue());
   Console.WriteLine("Current Laser State\t: {0}", isLaserEnabled ? "ON" :
↪"OFF");
```

(continues on next page)

(continued from previous page)

```
    // Change the current state to the opposite
    isLaserEnabled = !isLaserEnabled;
    laserEnabled.SetValue((uint)(Convert.ToUInt32(isLaserEnabled)));
    Console.WriteLine("New Laser State\t\t: {0}", isLaserEnabled ? "ON" : "OFF
↪");
    Console.WriteLine("----------------------------------------");

    Scanners[i].SetParam(laserEnabled);
}

//  Write changes parameters to the device's memory
Scanners[i].WriteParams();
```

WriteParams()

The function of writing local parameters from the SDK to the scanner. When this function is called, a list of local parameters is sent from the SDK to the scanner.

bool SDK.SCANNERS.RF62X.RF627old.WriteParams(PROTOCOLS_TYPES protocol = PROTOCOL

Write current parameters to device's memory

Return true on success

Parameters

- `protocol`: protocol's type (Service Protocol, ENIP, Modbus-TCP)

Usage example:

```
{
...Initialize sdk library
...Search for RF627old
}

// Establish connection to the RF627 device by Service Protocol.
Scanners[i].Connect();

// read params from RF627 device by Service Protocol.
Scanners[i].ReadParams();

{
...Some steps to change scanner's parameters
}

//  Write changes parameters to the device's memory
Scanners[i].WriteParams();

// Disconnect from scanner.
Scanners[i].Disconnect();
```

2.5 Wrapper API in Python

This library allows you to simplify the development of Python applications

To use it in Python projects, the developer must include the necessary library py-files in his project, as well as the dynamic program library rf62Xsdk.dll. Interaction with the dynamic library occurs through the built-in Python module ctypes. The module allows you to call functions of the dynamic program library rf62Xsdk.dll from Python, using the data structures of the C language (see rf62X_types.h) directly in Python.

To download the library rf62Xsdk.dll see *latest C wrapper releases*.
To compile the library rf62Xsdk.dll see *compile and run a C wrapper*.

2.5.1 SDK initialization

File PYSDK.py is the main program interface (API) file for developing programs in Python and defines the functionality of the "wrapper" library for rf62Xsdk.dll. PYSDK.py contains the following set of development classes and functions for initializing the SDK:

sdk_init()

SDK initialization function. Must be called once before any further calls to any library functions:

TODO sdk_cleanup()

Function for clearing resources allocated using the `sdk_init` function:

sdk_version()

Function for getting the current SDK version

search_scanners()

Function for searching RF62X devices over the network. It enumerates all available network adapters on the current system and sends "Hello" requests. The result is a list of discovered devices.

select_scanner()

The function of selecting a specific scanner from the list of scanners found after scanning the network in `search_scanners`. One field of information about the scanner is set as a parameter (see `get_info_about_scanner`): device_name, serial_number, ip_address, mac_address, profile_port, service_port, firmware_version, hardware_version, z_begin, z_range, x_begin, x_end

Usage example:

```
list = search(protocol=const_protocol.kSERVICE)
current_scanner=select_scanner(list, ip_address='192.168.1.30')
```

2.5.2 Rf627_device class

This class is defined in the file PYSDK.py and provides an interface for working with RF62X series scanners

__init__()

Constructor

__str__()

Present information about an object as a readable string

connect_to_scanner()

Function for establishing connection with the scanner

disconnect_from_scanner()

Function to close a previously established connection to the RF62X series scanner

get_profile2D_from_scanner()

Function for obtaining a profile from RF62X series scanners

get_info_about_scanner()

Function for getting information about the scanner from the hello package

get_parameter()

Function for getting a specific parameter by its name. When this function is called, the “kernel” searches for the required parameter from the last read when calling the function: ref: *read_params_from_scanner*. If the requested parameter is absent in a specific scanner, the function will return None.

For more convenient work with parameters, you can use the appropriate «keys» (parameter name key, parameter type and parameter access). To do this, in the file PYSDK.py the following `enum` are located:

set_parameter()

Function for setting a specific parameter. The new value new_value can be set in two ways. Method 1: the new value is passed as a variable of type (dict) with the changed value field

(variable['value']=new_value)

Method 2: the new value is passed as a value (variable = new_value), however, the second parameter must be passed - the name («key») of the parameter being changed. For more convenient work with parameters, you can use the appropriate «keys» (parameter name key, parameter type and parameter access). To do this, in the file PYSDK.py the following `enum` are located:

When this function is called, the passed parameter is set in the local parameter list in the «core». To send changes to the scanner, call the `write_params_to_scanner` function.

read_params_from_scanner()

The function of obtaining the current parameters of the scanner. When this function is called, the "kernel" reads all the actual parameters from the scanner, saving them as a "list of parameters" for further work.

write_params_to_scanner()

The function of writing local parameters from the «core» to the scanner. When this function is called, the list of local parameters is sent from the «core» to the scanner.

TODO send_cmd()

Function of sending commands to the scanner

For a more detailed description of commands and their properties see *General device commands*

2.5.3 Rf627_Error class

This class is defined in the file PYSDK.py and provides an interface for working with RF62X series scanners

CHAPTER

THREE

EXAMPLES OF USING

3.1 Examples for C

3.1.1 Find RF62X devices

Below is an example of searching for RF627Old series scanners on the network

```
#include <network.h>

#include <stdio.h>
#include <stdlib.h>

#include <rf62Xcore.h>
#include <rf62X_sdk.h>
#include <rf62X_types.h>

int main()
{

   // Initialize sdk library
   score_init();

   // Print return rf627 sdk version
   printf("SDK version: %s\n", sdk_version());
   printf("========================================\n");

   // Create value for scanners vector's type
   vector_t* scanners = (vector_t*)calloc(1, sizeof (vector_t));
   // Initialization vector
   vector_init(&scanners);

   uint32_t host_ip_addr = ntohl(inet_addr("192.168.1.1"));
   uint32_t host_mask = ntohl(inet_addr("255.255.255.0"));

   // call the function to change adapter settings inside the library.
   set_platform_adapter_settings(host_mask, host_ip_addr);

   // Search for RF627-old devices over network by Service Protocol.
   search_scanners(scanners, kRF627_OLD, kSERVICE);

   // Print count of discovered RF627Old in network by Service Protocol
   printf("Discovered: %d rf627-old\n", (int)vector_count(scanners));

   for (int i = 0; i < (int)vector_count(scanners); i++)
```

(continues on next page)

(continued from previous page)

```
    {
        hello_information info = get_info_about_scanner(vector_get(scanners,i),␣
↪kSERVICE);

        printf("\n\n\nID scanner's list: %d\n", i);
        printf("----------------------------------------\n");
        printf("Device information: \n");
        printf("* Name\t: %s\n", info.rf627old.hello_info_service_protocol->
↪device_name);
        printf("* Serial\t: %d\n", info.rf627old.hello_info_service_protocol->
↪serial_number);
        printf("* IP Addr\t: %d.%d.%d.%d\n",
                info.rf627old.hello_info_service_protocol->ip_address[0],
                info.rf627old.hello_info_service_protocol->ip_address[1],
                info.rf627old.hello_info_service_protocol->ip_address[2],
                info.rf627old.hello_info_service_protocol->ip_address[3]);
        printf("* MAC Addr\t: %d:%d:%d:%d:%d:%d\n",
                info.rf627old.hello_info_service_protocol->mac_address[0],
                info.rf627old.hello_info_service_protocol->mac_address[1],
                info.rf627old.hello_info_service_protocol->mac_address[2],
                info.rf627old.hello_info_service_protocol->mac_address[3],
                info.rf627old.hello_info_service_protocol->mac_address[4],
                info.rf627old.hello_info_service_protocol->mac_address[5]);

        printf("\nWorking ranges: \n");
        printf("* Zsmr, mm\t: %d\n", info.rf627old.hello_info_service_protocol->
↪z_begin);
        printf("* Zmr , mm\t: %d\n", info.rf627old.hello_info_service_protocol->
↪z_range);
        printf("* Xsmr, mm\t: %d\n", info.rf627old.hello_info_service_protocol->
↪x_begin);
        printf("* Xemr, mm\t: %d\n", info.rf627old.hello_info_service_protocol->
↪x_end);

        printf("\nVersions: \n");
        printf("* Firmware\t: %d\n", info.rf627old.hello_info_service_protocol->
↪firmware_version);
        printf("* Hardware\t: %d\n", info.rf627old.hello_info_service_protocol->
↪hardware_version);
        printf("----------------------------------------\n");
    }
}
```

Below is the output of the application upon successful detection of the scanner on the network:

```
SDK version: 1.3.0
========================================
Discovered: 1 RF627Old

ID scanner's list: 0
----------------------------------------
Device information:
* Name      : RF627
* Serial    : 190068
* IP Addr   : 192.168.1.32
* MAC Addr  : 00:0a:35:6e:07:f5
```

(continues on next page)

(continued from previous page)

```
Working ranges:
* Zsmr, mm  : 70
* Zmr , mm  : 50
* Xsmr, mm  : 30
* Xemr, mm  : 42

Versions:
* Firmware  : 19.11.12
* Hardware  : 18.6.20
----------------------------------------
Press any key to continue . . .
```

You can open and compile this example with **Qt Creator**:

- Download the CMakeLists.txt file from the **examples/C/RF627_old/RF627_search** folder via **File > Open File or Project** (select the CMakeLists.txt file)
- Select the compiler (MinGW, MSVC2017, Clang) and click **Configure Project**
- Run the project

3.2 Examples for C++

3.2.1 Search for RF62X devices

Below is an example of searching the RF627Old series of scanners on the network

```cpp
#include <rf62Xsdk.h>
#include <rf62Xtypes.h>
#include <string>
#include <iostream>

using namespace SDK::SCANNERS::RF62X;

int main()
{

   // Initialize sdk library
   sdk_init();

   // Print return rf627 sdk version
   std::cout << "SDK version: " << sdk_version()                << std::endl;
   std::cout << "========================================="     << std::endl;

   // Create value for scanners vector's type
   std::vector<rf627old*> list;
   // Search for RF627old devices over network
   list = rf627old::search(PROTOCOLS::SERVICE);

   // Print count of discovered RF627Old in network by Service Protocol
   std::cout << "Discovered: " << list.size() << " RF627Old"   << std::endl;

   for (size_t i = 0; i < list.size(); i++)
   {
      rf627old::hello_info info = list[i]->get_info();
```

(continues on next page)

(continued from previous page)

```
        std::cout << "\n\n\nID scanner's list: " << i            << std::endl;
        std::cout << "----------------------------------------" << std::endl;
        std::cout << "Device information: "                      << std::endl;
        std::cout << "* Name\t: "     << info.device_name()      << std::endl;
        std::cout << "* Serial\t: "   << info.serial_number()    << std::endl;
        std::cout << "* IP Addr\t: "  << info.ip_address()       << std::endl;
        std::cout << "* MAC Addr\t: " << info.mac_address()      << std::endl;

        std::cout << "\nWorking ranges: "                        << std::endl;
        std::cout << "* Zsmr, mm\t: " << info.z_smr()            << std::endl;
        std::cout << "* Zmr , mm\t: " << info.z_mr()             << std::endl;
        std::cout << "* Xsmr, mm\t: " << info.x_smr()            << std::endl;
        std::cout << "* Xemr, mm\t: " << info.x_emr()            << std::endl;

        std::cout << "\nVersions: "                              << std::endl;
        std::cout << "* Firmware\t: " << info.firmware_version() << std::endl;
        std::cout << "* Hardware\t: " << info.hardware_version() << std::endl;
        std::cout << "----------------------------------------" << std::endl;
    }

    system("pause");
}
```

The following is the output of the application upon successful detection of the scanner on the network:

```
SDK version: 1.3.0
========================================
Discovered: 1 RF627Old

ID scanner's list: 0
----------------------------------------
Device information:
* Name      : RF627
* Serial    : 190068
* IP Addr   : 192.168.1.32
* MAC Addr  : 00:0a:35:6e:07:f5

Working ranges:
* Zsmr, mm  : 70
* Zmr , mm  : 50
* Xsmr, mm  : 30
* Xemr, mm  : 42

Versions:
* Firmware  : 19.11.12
* Hardware  : 18.6.20
----------------------------------------
Press any key to continue . . .
```

You can open and compile this example with **Qt Creator**:

- Download the CMakeLists.txt file from the **examples/Cpp/RF627_old/RF627_search** folder through **File > Open File or Project** (select the CMakeLists.txt file)
- Select the compiler (MinGW, MSVC2017, Clang) and click **Configure Project**

- Run the project

3.2.2 Getting a scanner profile

The following is an example of retrieving profiles from RF627Old Series Scanners

```
#include <rf62Xsdk.h>
#include <rf62Xtypes.h>
#include <string>
#include <iostream>

using namespace SDK::SCANNERS::RF62X;

int main()
{

    // Initialize sdk library
   sdk_init();

   // Print return rf627 sdk version
   std::cout << "SDK version: " << sdk_version()                << std::endl;
   std::cout << "========================================"      << std::endl;


    // Create value for scanners vector's type
   std::vector<rf627old*> list;
   // Search for RF627old devices over network
   list = rf627old::search(PROTOCOLS::SERVICE);


   // Print count of discovered RF627Old in network by Service Protocol
   std::cout << "Discovered: " << list.size() << " RF627Old"    << std::endl;


   // Iterate over all discovered RF627Old in network, connect to each of
   // them and get a profile.
   for(size_t i = 0; i < scanners.size(); i++)
   {
      rf627old::hello_info info = list[i]->get_info();

      // Print information about the scanner to which the profile belongs.
      std::cout << "\n\n\nID scanner's list: " << i             << std::endl;
      std::cout << "---------------------------------------" << std::endl;
      std::cout << "Device information: "                      << std::endl;
      std::cout << "* Name\t: "     << info.device_name()      << std::endl;
      std::cout << "* Serial\t: "   << info.serial_number()    << std::endl;
      std::cout << "* IP Addr\t: "  << info.ip_address()       << std::endl;

      // Establish connection to the RF627 device by Service Protocol.
      list[i]->connect();

      // Get profile from scanner's data stream by Service Protocol.
      profile2D_t* profile = list[i]->get_profile2D();
      if (profile != nullptr)
      {
         std::cout << "Profile information: "                  << std::endl;
         switch (profile->header.data_type) {
```

(continues on next page)

(continued from previous page)

```
        case (uint8_t)PROFILE_DATA_TYPE::PIXELS:
            std::cout << "* DataType\t: "<< "PIXELS"            << std::endl;
            std::cout << "* Count\t: " << profile->pixels.size()<< std::endl;
            break;
        case (uint8_t)PROFILE_DATA_TYPE::PROFILE:
            std::cout << "* DataType\t: "<< "PROFILE"           << std::endl;
            std::cout << "* Size\t: "  << profile->points.size()<< std::endl;
            break;
        case (uint8_t)PROFILE_DATA_TYPE::PIXELS_INTRP:
            std::cout << "* DataType\t: "<< "PIXELS_INTRP"      << std::endl;
            std::cout << "* Count\t: " << profile->pixels.size()<< std::endl;
            break;
        case (uint8_t)PROFILE_DATA_TYPE::PROFILE_INTRP:
            std::cout << "* DataType\t: "<< "PROFILE_INTRP"     << std::endl;
            std::cout << "* Size\t: "  << profile->points.size()<< std::endl;
            break;
        }
        std::cout << "Profile was successfully received!"       << std::endl;
        std::cout << "----------------------------------------"<< std::endl;
      }else
      {
        std::cout << "Profile was not received!"                << std::endl;
        std::cout << "----------------------------------------"<< std::endl;
      }

   }

   system("pause");
}
```

The following is the result of the output of the application upon successful receipt of the profile from the scanner:

```
SDK version: 1.3.0
========================================
Discovered: 1 RF627Old


ID scanner's list: 0
----------------------------------------
Device information:
* Name      : RF627
* Serial    : 190068
* IP Addr   : 192.168.1.32
Profile information:
* DataType  : PROFILE
* Size      : 648
Profile was successfully received!
----------------------------------------
Press any key to continue . . .
```

You can open and compile this example with **Qt Creator**:

- Download the CMakeLists.txt file from the **examples/Cpp/RF627_old/RF627_profile** folder through **File > Open File or Project** (select the CMakeLists.txt file)
- Select the compiler (MinGW, MSVC2017, Clang) and click **Configure Project**

- Run the project

3.2.3 Getting and setting parameters

The following is an example of getting and changing the name of the scanner, setting the IP address, changing the state of the laser (turning it on or off):

```
#include <rf62Xsdk.h>
#include <rf62Xtypes.h>
#include <iostream>
#include <string>

using namespace SDK::SCANNERS::RF62X;

int main()
{

   // Initialize sdk library
   sdk_init();

   // Print return rf62X SDK version
   std::cout << "SDK version: " << sdk_version()                    <<␣
↪std::endl;
   std::cout << "========================================="         <<␣
↪std::endl;


   // Create value for scanners vector's type
   std::vector<rf627old*> scanners;
   // Search for RF627old devices over network
   scanners = rf627old::search(PROTOCOLS::SERVICE);


   // Print count of discovered RF627Old in network by Service Protocol
   std::cout << "Discovered: " << scanners.size() << " RF627Old"  <<␣
↪std::endl;


   // Iterate over all discovered RF627Old in network, connect to each of
   // them and read/set parameters.
   for(size_t i = 0; i < scanners.size(); i++)
   {

      rf627old::hello_info info = scanners[i]->get_info();

      std::cout << "\n\n\nID scanner's list: " << i               <<␣
↪std::endl;
      std::cout << "----------------------------------------"    <<␣
↪std::endl;

      // Establish connection to the RF627 device by Service Protocol.
      scanners[i]->connect();

      // read params from RF627 device by Service Protocol.
      scanners[i]->read_params();

      // Get parameter of Device Name
```

(continues on next page)

(continued from previous page)

```
      param_t* name = scanners[i]->get_param(PARAM_NAME_KEY::USER_GENERAL_
↪DEVICENAME);
      if (name->type == param_value_types[(int)PARAM_VALUE_TYPE::STRING_PARAM_
↪TYPE])
      {
         std::string str_name = name->get_value<value_str>();
         std::cout << "Current Device Name \t: " << str_name     << std::endl;

         // Add "_TEST" to the ending of the current name
         str_name += "_TEST";
         name->set_value<value_str>(str_name);
         std::cout << "New Device Name \t: " << str_name         << std::endl;
         std::cout << "----------------------------------------"<< std::endl;

         scanners[i]->set_param(name);
      }

      // Get parameter of Device IP Addr
      param_t* ip_addr = scanners[i]->get_param(PARAM_NAME_KEY::USER_NETWORK_
↪IP);
      if (ip_addr->type == param_value_types[(int)PARAM_VALUE_TYPE::UINT32_
↪ARRAY_PARAM_TYPE])
      {
         std::vector <uint32_t> ip = ip_addr->get_value<array_uint32>();
         std::cout << "Current Device IP\t: ";
         for(auto i: ip) std::cout<<std::to_string(i)<<".";std::cout<
↪<std::endl;

         // Change last digit of IP address (e.g. 192.168.1.30 -> 192.168.1.
↪31)
         ip[3]++;
         ip_addr->set_value<array_uint32>(ip);
         std::cout << "New Device IP\t: ";
         for(auto i: ip) std::cout<<std::to_string(i)<<".";std::cout<
↪<std::endl;
         std::cout << "----------------------------------------"<< std::endl;

         scanners[i]->set_param(ip_addr);
      }

      // Get parameter of Laser Enabled
      param_t* laser_enabled = scanners[i]->get_param(PARAM_NAME_KEY::USER_
↪LASER_ENABLED);
      if (laser_enabled->type == param_value_types[(int)PARAM_VALUE_
↪TYPE::UINT_PARAM_TYPE])
      {
         bool isEnabled = laser_enabled->get_value<value_uint32>();
         std::cout<<"Current Laser State\t: "<<(isEnabled?"ON":"OFF")<
↪<std::endl;

         isEnabled = !isEnabled;
         // Change the current state to the opposite
         laser_enabled->set_value<value_uint32>(!isEnabled);
         std::cout<<"New Laser State\t: "<<(isEnabled?"ON":"OFF")<<std::endl;
         std::cout << "----------------------------------------"<< std::endl;

         scanners[i]->set_param(laser_enabled);
```

(continues on next page)

(continued from previous page)

```
        }

        //  Write changes parameters to the device's memory
        scanners[i]->write_params();

    }

    system("pause");

}
```

The following is the output from the application upon successful installation of the new parameters:

```
SDK version: 1.3.0
========================================
Discovered: 1 RF627Old


ID scanner's list: 0
----------------------------------------
Current Device Name  : RF627
New Device Name      : RF627_TEST
----------------------------------------
Current Device IP    : 192.168.1.32.
New Device IP        : 192.168.1.33.
----------------------------------------
Current Laser State  : ON
New Laser State      : OFF
----------------------------------------

Press any key to continue . . .
```

You can open and compile this example with **Qt Creator**:

- Download the CMakeLists.txt file from the **examples/Cpp/RF627_old/RF627_params** folder through **File > Open File or Project** (select the CMakeLists.txt file)
- Select the compiler (MinGW, MSVC2017, Clang) and click **Configure Project**
- Run the project

3.3 Examples for C#

3.3.1 Search for RF62X devices

Below is an example of searching the RF627Old series of scanners on the network

```
using System;
using System.Collections.Generic;
using SDK.SCANNERS;

namespace RF627_search
{
    class Program
    {
```

(continues on next page)

(continued from previous page)

```
        static void Main(string[] args)
        {

            // Start initialization of the library core
            RF62X.SdkInit();

            // Print return rf62X sdk version
            Console.WriteLine("SDK version: {0}", RF62X.SdkVersion());
            Console.WriteLine("========================================");

            // Search for RF627old devices over network
            Console.WriteLine("- Start searching device");
            List<RF62X.RF627old> Scanners = RF62X.RF627old.Search();
            Console.WriteLine("+ {0} scanners detected", Scanners.Count);

            for (int i = 0; i < Scanners.Count; i++)
            {
                RF62X.HelloInfo info = Scanners[i].GetInfo();

                Console.WriteLine("\n\n\nID scanner's list: {0}", i);
                Console.WriteLine("----------------------------------------");
                Console.WriteLine("Device information: ");
                Console.WriteLine("* Name\t: {0}", info.device_name);
                Console.WriteLine("* Serial\t: {0}", info.serial_number);
                Console.WriteLine("* IP Addr\t: {0}", info.ip_address);
                Console.WriteLine("* MAC Addr\t: {0}", info.mac_address);

                Console.WriteLine("Working ranges: ");
                Console.WriteLine("* Zsmr, mm\t: {0}", info.z_smr);
                Console.WriteLine("* Zmr , mm\t: {0}", info.z_mr);
                Console.WriteLine("* Xsmr, mm\t: {0}", info.x_smr);
                Console.WriteLine("* Xemr, mm\t: {0}", info.x_emr);

                Console.WriteLine("\nVersions: ");
                Console.WriteLine("* Firmware\t: {0}", info.firmware_version);
                Console.WriteLine("* Hardware\t: {0}", info.hardware_version);
                Console.WriteLine("----------------------------------------");
            }

            Console.WriteLine("{0}Press any key to end \"Search-test\"",␣
↪Environment.NewLine);
            Console.ReadKey();

            // Cleanup resources allocated with SdkInit()
            RF62X.SdkCleanup();
        }
    }
}
```

3.3.2 Getting a scanner profile

The following is an example of retrieving profiles from RF627Old Series Scanners

```
using System;
using System.Collections.Generic;
using SDK.SCANNERS;

namespace RF627_profile
{
   class Program
   {
      static void Main(string[] args)
      {

         // Start initialization of the library core
         RF62X.SdkInit();


         // Search for RF627old devices over network
         Console.WriteLine("- Start searching device");
         List<RF62X.RF627old> Scanners = RF62X.RF627old.Search();
         Console.WriteLine("+ {0} scanners detected", Scanners.Count);


         // foreach over an scanners list
         for (int i = 0; i < Scanners.Count; i++)
         {
            RF62X.HelloInfo info = Scanners[i].GetInfo();

            Console.WriteLine("\n\n\nID scanner's list: {0}", i);
            Console.WriteLine("----------------------------------------");
            Console.WriteLine("Device information: ");
            Console.WriteLine("* Name\t: {0}", info.device_name);
            Console.WriteLine("* Serial\t: {0}", info.serial_number);
            Console.WriteLine("* IP Addr\t: {0}", info.ip_address);

            // Establish connection to the RF627 device by Service Protocol.
            Scanners[i].Connect();

            // Get profile from scanner's data stream by Service Protocol.
            RF62X.Profile profile = Scanners[i].GetProfile();
            if (profile.header != null)
            {
               Console.WriteLine("Profile information: ");
               switch (profile.header.data_type)
               {
                  case RF62X.PROFILE_TYPE.PIXELS_NORMAL:
                     Console.WriteLine("* DataType\t: PIXELS");
                     Console.WriteLine("* Count\t: {0}", profile.pixels.
↪Count);
                     break;
                  case RF62X.PROFILE_TYPE.PROFILE_NORMAL:
                     Console.WriteLine("* DataType\t: PROFILE");
                     Console.WriteLine("* Size\t: {0}", profile.points.Count);
                     break;
                  case RF62X.PROFILE_TYPE.PIXELS_INTERPOLATED:
                     Console.WriteLine("* DataType\t: PIXELS");
```

(continues on next page)

(continued from previous page)

```
                    Console.WriteLine("* Count\t: {0}", profile.pixels.
↪Count);
                    break;
                case RF62X.PROFILE_TYPE.PROFILE_INTERPOLATED:
                    Console.WriteLine("* DataType\t: PROFILE");
                    Console.WriteLine("* Size\t: {0}", profile.points.Count);
                    break;
                default:
                    break;
            }
            Console.WriteLine("Profile was successfully received!");
            Console.WriteLine("----------------------------------------");
        }else
        {
            Console.WriteLine("Profile was not received!");
            Console.WriteLine("----------------------------------------");
        }

        // Disconnect from scanner.
        Scanners[i].Disconnect();
      }

      Console.WriteLine("{0}Press any key to end \"Search-test\"",␣
↪Environment.NewLine);
      Console.ReadKey();

      // Cleanup resources allocated with SdkInit()
      RF62X.SdkCleanup();
    }
  }
}
```

3.3.3 Getting and setting parameters

The following is an example of getting and changing the name of the scanner, setting the IP address, changing the state of the laser (turning it on or off):

```
using System;
using System.Collections.Generic;
using SDK.SCANNERS;

namespace RF627_params
{
  class Program
  {
    static void Main(string[] args)
    {

      // Start initialization of the library core
      RF62X.SdkInit();

      // Search for RF627old devices over network
      Console.WriteLine("- Start searching device");
      List<RF62X.RF627old> Scanners = RF62X.RF627old.Search();
      Console.WriteLine("+ {0} scanners detected", Scanners.Count);
```

(continues on next page)

(continued from previous page)

```
        // foreach over an scanners list
        for (int i = 0; i < Scanners.Count; i++)
        {
            // Establish connection to the RF627 device by Service Protocol.
            Scanners[i].Connect();

            // read params from RF627 device by Service Protocol.
            Scanners[i].ReadParams();

            // Get parameter of Device Name
            RF62X.Param<string> name = Scanners[i].GetParam(RF62X.Params.User.
↪General.deviceName);
            if (name != null)
            {
                string strName = name.GetValue();
                Console.WriteLine("\n\nCurrent Device Name \t: {0}", strName);

                // Add "_TEST" to the ending of the current name
                strName += "_TEST";
                name.SetValue(strName);
                Console.WriteLine("New Device Name \t: {0}", strName);
                Console.WriteLine("----------------------------------------");

                Scanners[i].SetParam(name);
            }

            // Get parameter of Device IP Addr
            RF62X.Param<List<uint>> ipAddr = Scanners[i].GetParam(RF62X.
↪Params.User.NetWork.ip);
            if (ipAddr != null)
            {
                List<uint> ip = ipAddr.GetValue();
                Console.WriteLine("Current Device IP Addr\t: {0}.{1}.{2}.{3}",␣
↪ip[0], ip[1], ip[2], ip[3]);

                // Change last digit of IP address (e.g. 192.168.1.30 -> 192.
↪168.1.31)
                ip[3]++;
                ipAddr.SetValue(ip);
                Console.WriteLine("New Device IP Addr\t: {0}.{1}.{2}.{3}",␣
↪ip[0], ip[1], ip[2], ip[3]);
                Console.WriteLine("----------------------------------------");

                Scanners[i].SetParam(ipAddr);
            }

            // Get parameter of Laser Enabled
            RF62X.Param<uint> laserEnabled = Scanners[i].GetParam(RF62X.
↪Params.User.Laser.enabled);
            if (laserEnabled != null)
            {
                bool isLaserEnabled = Convert.ToBoolean(laserEnabled.
↪GetValue());
                Console.WriteLine("Current Laser State\t: {0}", isLaserEnabled␣
↪? "ON" : "OFF");
```

(continues on next page)

(continued from previous page)

```
                // Change the current state to the opposite
                isLaserEnabled = !isLaserEnabled;
                laserEnabled.SetValue((uint)(Convert.
↪ToUInt32(isLaserEnabled)));
                Console.WriteLine("New Laser State\t\t: {0}", isLaserEnabled ?
↪"ON" : "OFF");
                Console.WriteLine("----------------------------------------");

                Scanners[i].SetParam(laserEnabled);
            }

            //  Write changes parameters to the device's memory
            Scanners[i].WriteParams();

            // Disconnect from scanner.
            Scanners[i].Disconnect();
        }
        Console.WriteLine("{0}Press any key to end \"Parameters-test\"",␣
↪Environment.NewLine);
        Console.ReadKey();

        // Cleanup resources allocated with SdkInit()
        RF62X.SdkCleanup();
    }
  }
}
```

3.4 Examples for Python

The source code that demonstrates how to work with the scanner is located in the demo.py file. Below is a description of each part of the code.

Import of the functions of the “wrapper” library for rf62Xsdk.dll, which are located in the PYSDK.py file:

```
# enums
from PYSDK import const_protocol, const_scanner_type
from PYSDK import const_parameter_name, const_old_data_type
# PySDK classes
from PYSDK import rf627_Error, rf627_device
# PySDK functions
from PYSDK import sdk_init, get_sdk_version, search_scanners, select_scanner
```

3.4.1 Find RF62X devices

Below is an example of searching for RF627Old series scanners on the network and selecting a scanner with a specific ip-address.

```
# ===================== RF627 Python SDK demo ================
# Initialize sdk library
sdk_init()
print('sdk version', get_sdk_version()) # Print sdk version
```

(continues on next page)

(continued from previous page)

```
# ==================== RF627_search ========================
#

# Search for RF627old devices over network
list = search_scanners(protocol=const_protocol.kSERVICE,
                       scanner_type=const_scanner_type.kRF627_OLD)
# Print count of discovered rf627-old in network by Service Protocol
print('Discovered: ', len(list), ' rf627-old')
# Print info of all discovered devices in network by Service Protocol
for i in range(len(list)):
    print('Scanner index : ', i)
    print(list[i])

# Select device by ip. Note: device can be selected by
# device_name, serial_number, etc. (see select_scanner documentation)
current_scanner=select_scanner(list, ip_address='192.168.1.30')
```

Below is the output of the application upon successful detection of the scanner on the network:

```
sdk version 1.3.7
Discovered: 1 rf627-old
Scanner index : 0

---rf627 Device info---
device_name = ABC
serial_number = 190171
ip_address = 192.168.1.30
mac_address = 00:0a:35:6e:4a:9a
profile_port = 50001
service_port = 50011
firmware_version = 19.11.15
hardware_version = 18.6.20
z_begin = 76
z_range = 100
x_begin = 48
x_end = 82
scanner_type = 1
protocol_type = 1

Selected scanner by ip_address 192.168.1.30
```

3.4.2 Retrieving Scanner Information and Scanner Profile

After the scanner is selected, it retrieves information about the scanner and reads the profile from the RF627Old series scanner.

```
if current_scanner:
    # Print current_scanner device info
    print('Current scanner info')
    hello = current_scanner.get_info_about_scanner()
    print('Device information fields: ')
    print('* Name\t: ', hello['device_name'])
    print('* Serial\t: ', hello['serial_number'])
    print('* IP Addr\t: ', hello['ip_address'])
    print('* MAC Addr\t: ', hello['mac_address'])
```

(continues on next page)

(continued from previous page)

```
    print('\nWorking ranges:')
    print('* Zsmr, mm\t: ', hello['z_begin'])
    print('* Zmr , mm\t: ', hello['z_range'])
    print('* Xsmr, mm\t: ', hello['x_begin'])
    print('* Xemr, mm\t: ', hello['x_end'])
    print('\nVersions: ')
    print('* Firmware\t: ', hello['firmware_version'])
    print('* Hardware\t: ', hello['hardware_version'])
    print('\n')

    # ====================RF627_profile=========================
    # Get profile from scanner's data stream by Service Protocol.
    # Profile conversion from ctypes to Python dictionary
    # in py_get_profile2D_from_scanner() function.

    prof = current_scanner.get_profile2D_from_scanner()

    if prof:
        datatype = prof['header']['data_type']
        print('Profile information: ')
        print('* DataType\t: ', const_old_data_type(datatype))

        if 'pixels_count' in prof:
            # pixels_format
            print('* Count\t: ', prof['pixels_count'])

        if 'points_count' in prof:
            # profile_format
            print('* Count\t: ', prof['points_count'])

        if 'intensity_count' in prof:
            # intensity
            print('* DataType\t: Intensity')
            print('* Count\t: ', prof['intensity_count'])
        print('Profile was successfully received!')
    else:
        print('Profile was not received!')
```

Below is the output of the application upon successful receipt of information about the scanner and reading the profile from the scanner:

```
Current scanner info
Device information fields:
* Name          :  ABC
* Serial        :  190171
* IP Addr       :  192.168.1.30
* MAC Addr      :  00:0a:35:6e:4a:9a

Working ranges:
* Zsmr, mm      :  76
* Zmr , mm      :  100
* Xsmr, mm      :  48
* Xemr, mm      :  82

Versions:
* Firmware      :  19.11.15
* Hardware      :  18.6.20
```

(continues on next page)

(continued from previous page)

```
Profile information:
* DataType      :  CONST_rf627_old_data_type_t.DTY_ProfileNormal
* Count         :  648

* DataType      : Intensity
* Count         :  648

Profile was successfully received!
```

3.4.3 Getting and setting parameters

Below is an example of obtaining and changing the name of the scanner, setting the IP address, changing the laser state (enabling or disabling) in two ways:

```
# ====================RF627_params=========================
#  Read params from RF627 device by Service Protocol.
current_scanner.read_params_from_scanner()

device_name = current_scanner.get_parameter(
                        const_parameter_name.USER_GENERAL_DEVICENAME)
if device_name:
    print('Old Device Name is \t:', device_name['value'])
    #  Add '_TEST' to the ending of the current name
    device_name['value']+='_TEST'
    #  Change Device Name, Method 1: update 'value' field of get_parameter()
    #  result and pass it to set_parameter
    current_scanner.set_parameter(device_name)
    device_name_new1 = current_scanner.get_parameter(
                        const_parameter_name.USER_GENERAL_DEVICENAME)
    print('Method 1: New Device Name is \t:', device_name_new1['value'])
    #  Add '_TEST2' to the ending of the current name
    new_name=device_name['value']+"2"
    #  Change Device Name, Method 2: set new value and pass it
    #  to set_parameter together with parameter name
    current_scanner.set_parameter(
                    new_name,const_parameter_name.USER_GENERAL_DEVICENAME)
    device_name_new2 = current_scanner.get_parameter(
                        const_parameter_name.USER_GENERAL_DEVICENAME)
    print('Method 2: New Device Name2 is \t:', device_name_new2['value'])

#  Get parameter of Device IP Addr.
ip = current_scanner.get_parameter(const_parameter_name.USER_NETWORK_IP)
if ip:
    print('Current Device IP \t:', ip['value'])
    #  Change last digit of IP address (e.g. 192.168.1.30 -> 192.168.1.31)
    ip['value'][3] = 31
    #  Method 1
    current_scanner.set_parameter(
                        ip['value'],const_parameter_name.USER_NETWORK_IP)
    ip_new1 = current_scanner.get_parameter(
                                const_parameter_name.USER_NETWORK_IP)
    print('Method 1: New Device IP \t:', ip_new1['value'])
    #  Method 2
```

(continues on next page)

(continued from previous page)

```
    current_scanner.set_parameter(ip)
    ip_new2 = current_scanner.get_parameter(
                                const_parameter_name.USER_NETWORK_IP)
    print('Method 2: New Device IP \t:', ip_new2['value'])

# Get parameter of Laser Enabled
is_laser_enabled = current_scanner.get_parameter(
                                const_parameter_name.USER_LASER_ENABLED)
if is_laser_enabled:
    print('Current Laser State\t: ', is_laser_enabled['value'])
    # switch Laser Enabled
    is_laser_enabled['value'] = not is_laser_enabled['value']
    current_scanner.set_parameter(is_laser_enabled)
    is_laser_enabled_new = current_scanner.get_parameter(
                                const_parameter_name.USER_LASER_ENABLED)
    print('New Laser State\t: ', is_laser_enabled_new['value'])

    # Write current parameters to device's memory
    # current_scanner.write_params_to_scanner()  #commented out in demo
else:
    print('Could not read scanner parameters')
```

Below is the output of the application when the new parameters are successfully set:

```
Old Device Name is              : ABC
Method 1: New Device Name is    : ABC_TEST
Method 2: New Device Name2 is   : ABC_TEST2
Current Device IP               : [192, 168, 1, 30]
Method 1: New Device IP         : [192, 168, 1, 31]
Method 2: New Device IP         : [192, 168, 1, 31]
Current Laser State             :  1
New Laser State                 :  0
```

3.5 Examples for LabVIEW

3.5.1 Environment setup

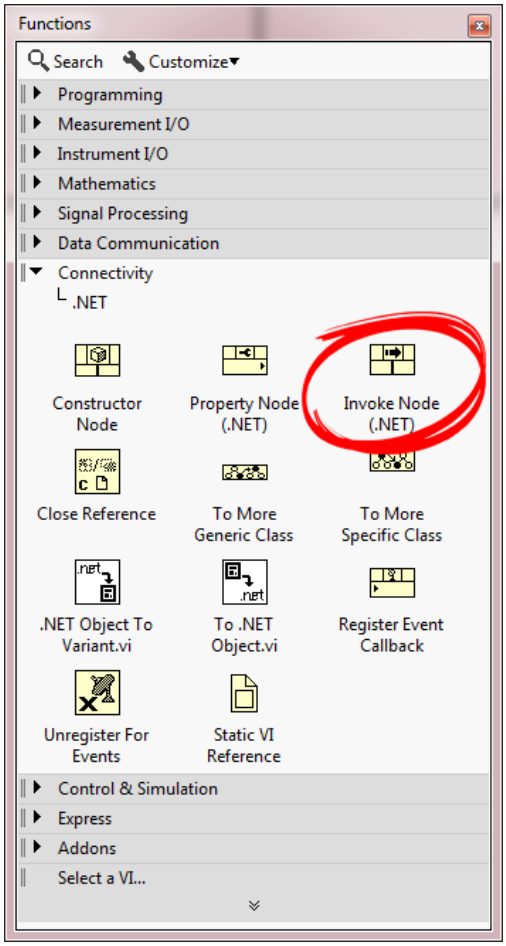
Before creating applications in the LabVIEW IDE for working with laser scanners of the RF62X series, you should:

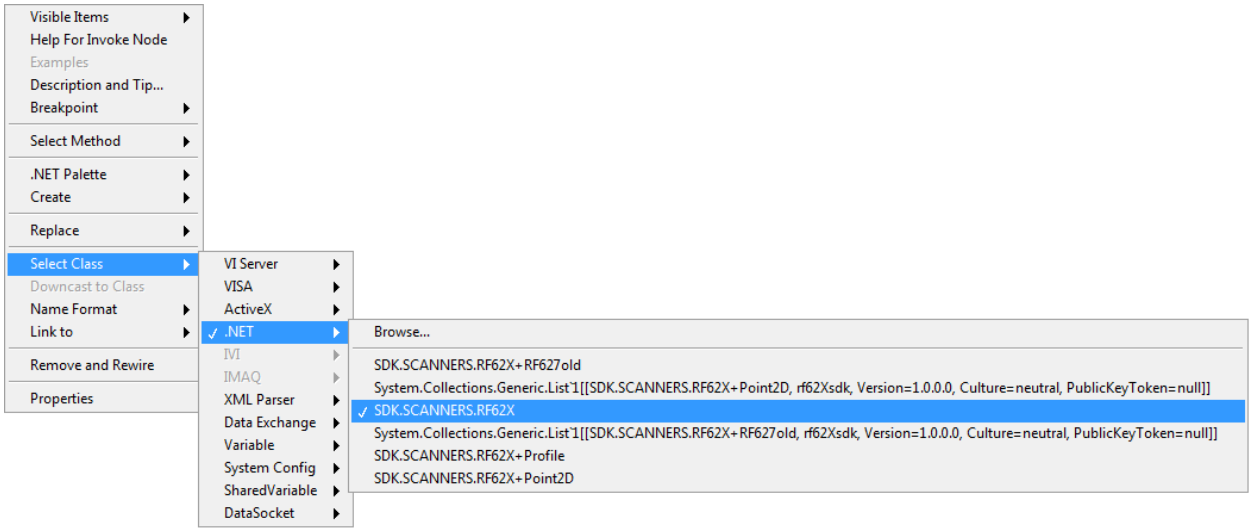
1) Add **two libraries** to the LabVIEW project folder: - the main library **rf62Xcore.dll** (see *Compiling the «Core» in C*) - the wrapper library **rf62Xsdk.dll** for .NET, written in C# (see *Compiling a wrapper in C#*)

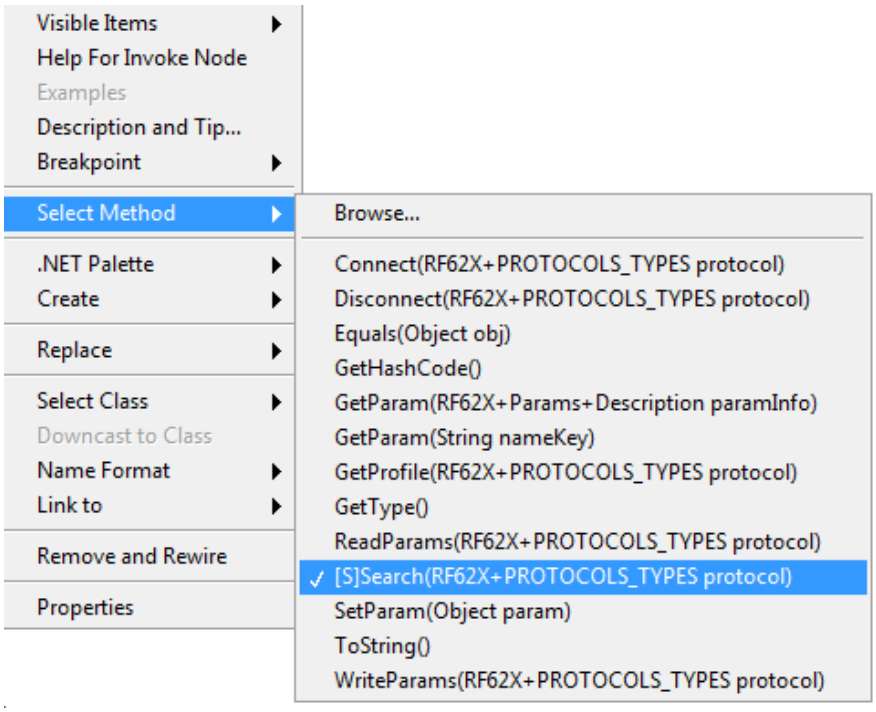
2) For invoking methods from **rf62Xsdk.dll**, when designing an algorithm in LabVIEW, the **Invoke Node (.Net)** component is required, which is located in the **Connectivity->.Net** section

3) In the context menu of the added component **Invoke Node (.Net)** you need to specify the library **rf62Xsdk.dll** and select the class **SDK.SCANNERS.RF62X**:

4) To call a specific method from the **rf62Xsdk.dll** library in the LabVIEW IDE, you need to open the **Class Method** section in the context menu of the added component **Invoke Node (.Net)**:

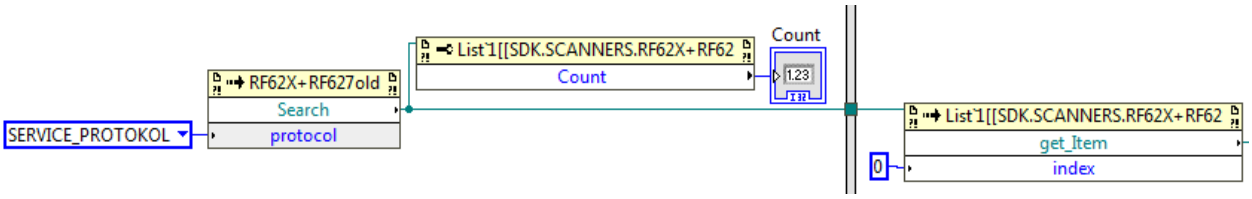

Functions
Search
Customize
Programming
Measurement I/O
Instrument I/O
Mathematics
Signal Processing
Data Communication
Connectivity
.NET
Constructor Node
Property Node (.NET)
Invoke Node (.NET)
Close Reference
To More Generic Class
To More Specific Class
.NET Object To Variant.vi
To .NET Object.vi
Register Event Callback
Unregister For Events
Static VI Reference
Control & Simulation
Express
Addons
Select a VI...


Visible Items
Help For Invoke Node
Examples
Description and Tip...
Breakpoint
Select Method
.NET Palette
Create
Replace
Select Class
Downcast to Class
Name Format
Link to
Remove and Rewire
Properties
VI Server
VISA
ActiveX
.NET
IVI
IMAQ
XML Parser
Data Exchange
Variable
System Config
SharedVariable
DataSocket
Browse...
SDK.SCANNERS.RF62X+RF627old
System.Collections.Generic.List`1[[SDK.SCANNERS.RF62X+Point2D, rf62Xsdk, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null]]
SDK.SCANNERS.RF62X
System.Collections.Generic.List`1[[SDK.SCANNERS.RF62X+RF627old, rf62Xsdk, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null]]
SDK.SCANNERS.RF62X+Profile
SDK.SCANNERS.RF62X+Point2D



3.5.2 Search for RF62X devices

Below is an example of searching the RF627Old series of scanners on the network



3.5.3 Getting a scanner profile

The following is an example of retrieving profiles from the scanner RF627Old series

3.5.4 Getting and setting parameters

The following is an example of getting and changing the name of the scanner, setting the IP address, changing the state of the laser (turning it on or off):

3.6 Examples for MatLab

3.6.1 Search for RF62X devices

Below is an example of searching the RF627Old series of scanners on the network

```
clc
dll_in_matlab = NET.addAssembly('rf62Xsdk.dll');
dll_in_matlab.Classes
```

(continues on next page)

(continued from previous page)

```
clc;
import SDK.SCANNERS.*
import SDK.SCANNERS.RF62X.*
import SDK.SCANNERS.RF62X+RF627old.*
import System.Collections.Generic.*

% Initialize sdk library
RF62X.SdkInit();

% Print return rf62X sdk version
RF62X.SdkVersion()

% Search for RF627old devices over network
list=Search()

% Cleanup resources allocated with SdkInit()
RF62X.SdkCleanup()
```

3.6.2 Getting a scanner profile

The following is an example of retrieving profiles from RF627Old Series Scanners

3.6.3 Getting and setting parameters

The following is an example of getting and changing the name of the scanner, setting the IP address, changing the state of the laser (turning it on or off):

Documentation	www.riftek.com
Website	www.riftek.com
Document version	1.4.0 от 31/08/2020
Library versions	1.4.0 от 31/08/2020

INDEX

G

H

I

M

N

P

R

S

T

U

W