Telit wireless solutions

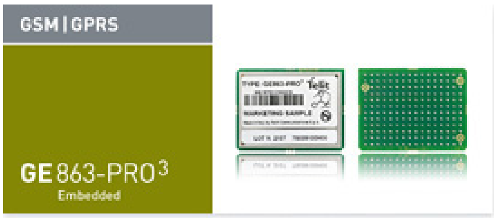
# Short Range Libraries User Guide

**1VV0300861** Rev.1 - 09/07/10

Making machines talk.

# Applicable Products and SW version

This document is relating to the following products:



GE863-PRO$^{3}$

GG863-SR ZigBee
GG863-SR PRO
GG863-SR W-MBUS

and to the following SR Libraries version:

| SR Library | Version |
|---|---|
| ZigBee | 28.00.01 |
| Mesh Lite | 27.00.02 |
| Wireless M-Bus | 2F.00.00 |

# Contents

## DISCLAIMER

The information contained in this document is the proprietary information of Telit Communications S.p.A. and its affiliates (“TELIT”). The contents are confidential and any disclosure to persons other than the officers, employees, agents or subcontractors of the owner or licensee of this document, without the prior written consent of Telit, is strictly prohibited.

Telit makes every effort to ensure the quality of the information it makes available. Notwithstanding the foregoing, Telit does not make any warranty as to the information contained herein, and does not accept any liability for any injury, loss or damage of any kind incurred by use of or reliance upon the information.

Telit disclaims any and all responsibility for the application of the devices characterized in this document, and notes that the application of the device must comply with the safety standards of the applicable country, and where applicable, with the relevant wiring rules.

Telit reserves the right to make modifications, additions and deletions to this document due to typographical errors, inaccurate information, or improvements to programs and/or equipment at any time and without notice. Such changes will, nevertheless be incorporated into new editions of this application note.

# 1 Introduction

## 1.1 Scope

This user guide details information about Short Range APIs available for platform based on Telit GE863 PRO$^3$.

## 1.2 Audience

This User Guide is intended for software developers who develop applications on the ARM processor of platform based on Telit GE863 PRO$^3$.

## 1.3 Contact Information, Support

Our aim is to make this guide as helpful as possible. Keep us informed of your comments and
suggestions for improvements.

For general contact, technical support, report documentation errors and to order manuals, contact
Telit's Technical Support Center at:

TS-EMEA@telit.com
or
http://www.telit.com/en/products/technical-support-center/contact.php

Telit appreciates feedback from the users of our information.

## 1.4 Open Source Licenses

Linux system is made up of many Open Source device drivers licensed as follows:

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
675 Mass Ave, Cambridge, MA 02139, USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Please refer to the following web page for the full text of the license:

http://www.gnu.org/licenses/gpl-2.0.html

## 1.5 Product Overview

These libraries aim to simplify Telit customer application development that needs to interact with a short range network.

## 1.6 Document Organization

This manual contains the following chapters:

- "Chapter 1: Introduction" provides a scope for this manual, target audience, technical contact information, and text conventions.
- "Chapter 2: System requirements" provides a description of operative context for Short Range Libraries and its general architecture.
- "Chapter 3: Libraries setup" gives guidelines to setup a project which involves Short Range Libraries.
- "Chapter 4: Short Range Libraries" describes short range libraries architecture, provides a list of available API and describes deeply every function and data type defined into the libraries.

### 1.6.1 How to Use

If you are new to this product, it is highly recommended to start reading the Telit GE863-PRO3 Linux Development Environment User Guide, the Telit GE863-PRO3 Linux SW User Guide manuals and this document in their entirety in order to understand the concepts and specific features provided by Short Range Libraries.

# 1.7 Text Conventions

This section lists the paragraph and font styles used for the various types of information presented in this user guide.

| Format | Content |
|---|---|
| Courier | Linux shell commands, filesystem paths, example C source code, function interfaces and data type definitions |

# 1.8 Acronyms

| Acronym | Meaning |
|---|---|
| ZBIPGW | ZigBee IP Gateway product |
| ZB | ZigBee short range communication technology |
| SR | Short Range |
| LR | Long Range |
| ML | Mesh Lite |
| MB | M-Bus |

# 1.9 Related Documents

## 1.9.1 Internal

The following Telit documents are related to this user guide:

IR[1] TelitGE863PRO3 Linux Development Environment 1VV0300780
IR[2] TelitGE863PRO3 Linux GSM Library User Guide 1vv0300782
IR[3] Telit M-ONE Protocol Stack User Guide 1vv0300819
IR[4] Wireless M-Bus User Guide 1vv0300828

All documentation can be downloaded from Telit’s official web site www.telit.com if not otherwise indicated.

## 1.9.2 External

The following external documents are related to this user guide:

ER[1] IEEE Std 802.15.4-2003
ER[2] ZigBee Specification 053474r18
ER[3] ZigBee Cluster Library Specification 075123r01
ER[4] Wireless M-Bus standard EN 13757-4

## 1.10 Document Change Log

| Revision | Date | Changes |
|---|---|---|
| ISSUE#0 | 16/07/09 | First draft |
| ISSUE#1 | 09/07/10 | Section 4.2 Changed<br>Section 4.3 Changed<br>Section 4.4 Changed<br>Section 4.5 Added |
|  |  |  |
|  |  |  |

# 2 System requirements

## 2.1 Hardware

There are mainly two kinds of operational context for the short range libraries:

1. The GG863-SR terminal
2. The GE863 PRO$^3$ module with additional TelitRF short range hardware

Both situations are described in the following paragraphs.

### 2.1.1 GG863-SR

The GG863-SR terminal contains a fully featured GSM/GPRS communications module, a standalone ARM9 CPU and a TelitRF ZigBee or Mesh or M-Bus module.
It makes possible to manage two kinds of communication technologies in the same product: for long range network (GSM/GPRS) and for short range network (ZigBee, Mesh or M-Bus).
Software developers can use the functions of short range Libraries to configure, manage and use short range hardware resource.

### 2.1.2 GE863 PRO$^3$

The GE863 PRO$^3$ contains a fully featured GSM/GPRS communications section and a standalone ARM9 CPU.
Additional short range hardware (TelitRF ZigBee, Mesh or M-Bus module) could be connected to the system through the serial interface (refer to IR[1] for more details). Once added to the system, the short range hardware resource could be configured and managed using Short Range Libraries.

## 2.2 Software

The Short Range Libraries should be used with Linux OS for GE863 PRO$^3$, which is provided by Telit.
In order to create a project which involves the Short Range Libraries also pthread library

shall be included. Refer to chapter 3 for more information about project setup.

# 3 Libraries setup

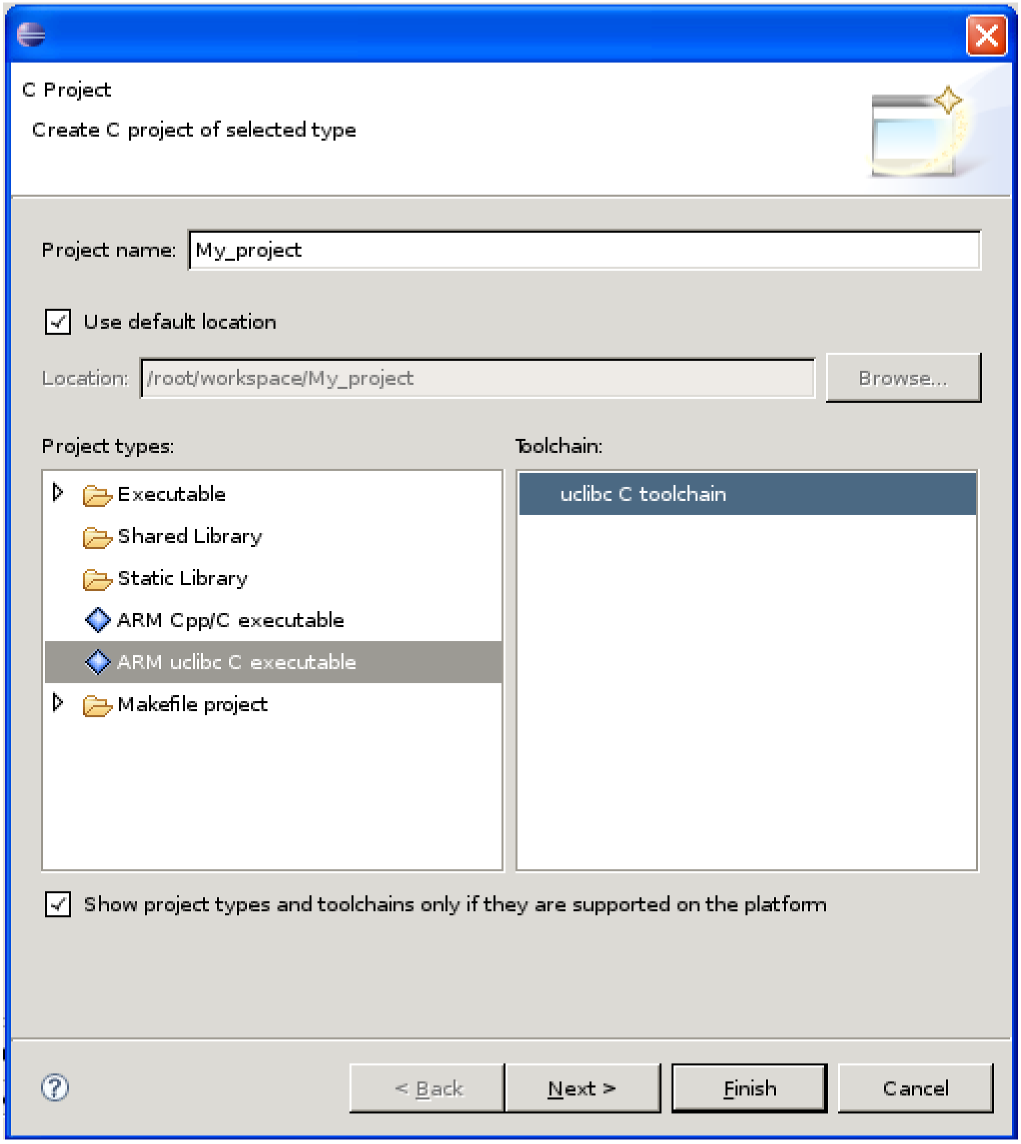
It is possible to add the SR-Library on your development environment simply inserting the header files and the library, within the /opt/crosstools/telit/include/ and /opt/crosstools/telit/lib directories respectively:

1. Start the Linux console (Windows Start Menu → All programs → Telit Development Platform → Console).
2. Copy the library typing: cp /mnt/windows/<PATH>/libSr_Zb_Library.a /opt/crosstools/telit/lib (FOR ZIGBEE)
3. Copy the library typing: cp /mnt/windows/<PATH>/libSr_Ml_Library.a /opt/crosstools/telit/lib (FOR MESHLITE)
4. Copy the library typing: cp /mnt/windows/<PATH>/libSr_Mb_Library.a /opt/crosstools/telit/lib (FOR M-BUS)
5. Copy the header file typing: cp /mnt/windows/<PATH>/SRlibrary.h /opt/crosstools/telit/include
6. Copy the header file typing: cp /mnt/windows/<PATH>/SRdata.h /opt/crosstools/telit/include
7. Copy the header file typing: cp /mnt/windows/<PATH>/SRZBlibrary.h /opt/crosstools/telit/include (ONLY FOR ZIGBEE)
8. Copy the header file typing: cp /mnt/windows/<PATH>/SRZBdata.h /opt/crosstools/telit/include (ONLY FOR ZIGBEE).
9. Copy the header file typing: cp /mnt/windows/<PATH>/SRMBlibrary.h /opt/crosstools/telit/include (ONLY FOR M-BUS)

where <PATH> is the folder of Windows where you have stored the new version of the library files.

## 3.1 How to build a simple application with SR-Libraries

Open your "Telit Customized Eclipse" starting from "Telit Development Platform" and create a New Project "ARM uclibc C executable" as shown in Figure 3.1.



C Project

Create C project of selected type

Project name: My_project

☑ Use default location

Location: /root/workspace/My_project

Project types:

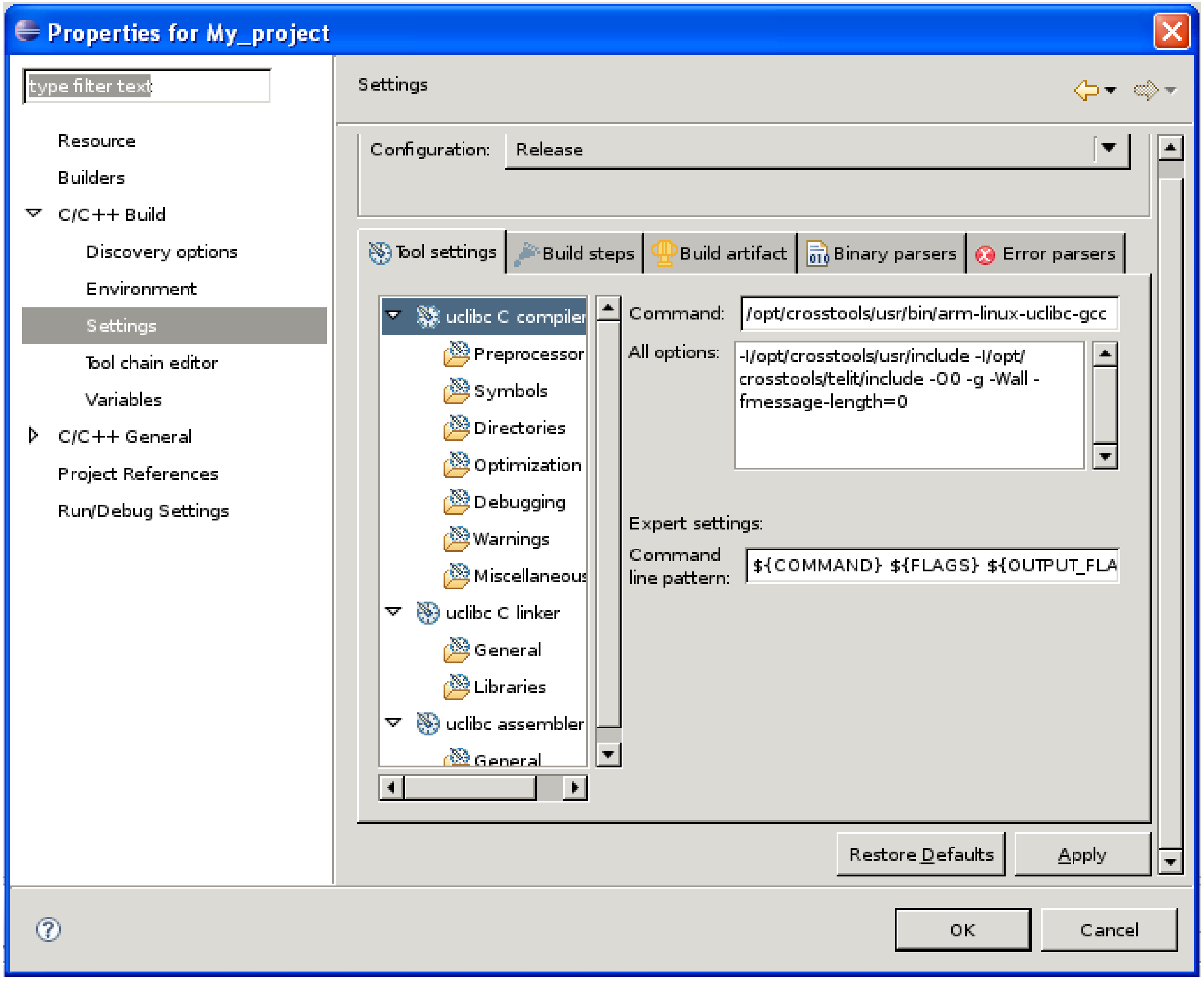
Toolchain:

uclibc C toolchain

☑ Show project types and toolchains only if they are supported on the platform

**Figure 3.1**

Open new project Properties window end select C/C++ Build -> Setting as shown in Figure 3.2.



Configuration: Release

Command: /opt/crosstools/usr/bin/arm-linux-uclibc-gcc

All options: -I/opt/crosstools/usr/include -I/opt/crosstools/telit/include -O0 -g -Wall -fmessage-length=0

Expert settings:

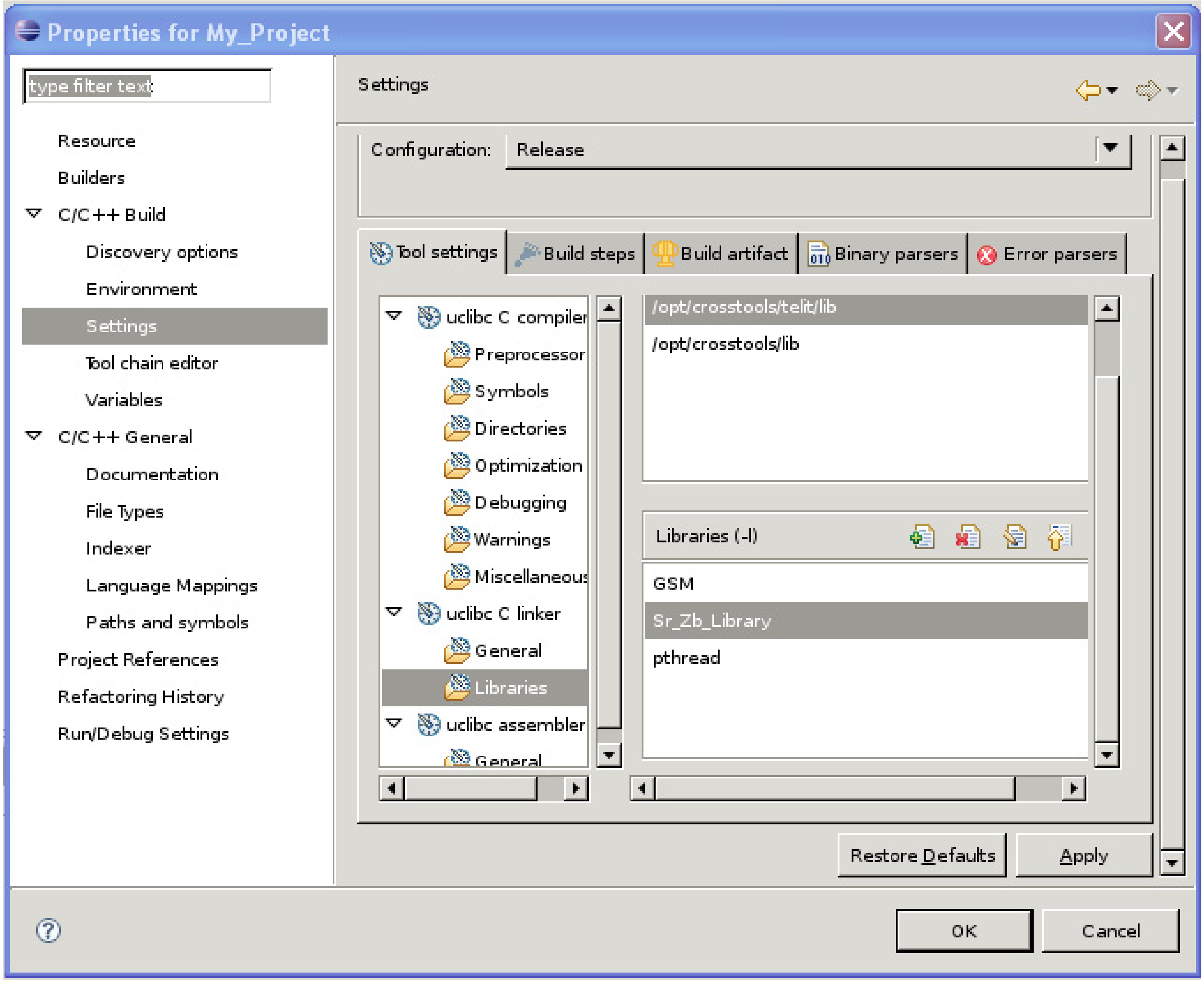
Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLA

**Figure 3.2**

Add in the uclib C linker -> Libraries add the folliwing libraries:

- Sr_Zb_Library (FOR ZIGBEE)
- Sr_Ml_Library (FOR MESHLITE)
- Sr_Mb_Library (FOR M-BUS)
- pthread

as shown in Figure 3.3. (it refers to a project based on ZigBee technology).



Properties for My_Project

Settings

Configuration: Release

Tool settings | Build steps | Build artifact | Binary parsers | Error parsers

uclibc C compiler > Libraries

/opt/crosstools/telit/lib
/opt/crosstools/lib

Libraries (-l)

GSM
Sr_Zb_Library
pthread

**Figure 3.3**

Then click on “Apply” to make changes effective and on “OK” to close the “Properties” window. Now the project is ready for build an application based upon SR-Libraries.

# 4 Short Range Libraries

## 4.1 Introduction

Short range libraries are a group of libraries that allow managing short range technologies supported by platform based on Telit GE863 PRO[3]. Every library is formed by two parts:

- **Generic functionalities**: this part is common to every short range library and provides the basic functionalities to configure, start, and scan a network, to reset the short range module and to send and receive data.
- **Specific functionalities**: every short range library has a different specific part of functionalities depending on the specific technology.

Until now the Short Range Library is available for ZigBee (libSr_Zb_Library.a), for MeshLite (libSr_Ml_Library.a) and for Wireless M-Bus (libSr_Mb_Library.a) technologies.

## 4.2 Generic API

### 4.2.1 Description

Generic API provides the basic functionalities that are common to all short range technologies. These functionalities are:

- Initialize the system to communicate with the short range hardware
- Configure network parameters
- Start the network
- Scan the network
- Reset the short range system
- Send and receive data

#### 4.2.1.1 Data Types

Data types defined for the generic part of every Short Range library are in header file "SRdata.h".

#### 4.2.1.1.1 Basic Types

The basic types defined in “SRdata.h” are shown and described in Table 4.1.

| Variable | Type | Description |
|---|---|---|
| `SR_SCAN_TYPE_TAG` | `UINT8` | It is the type used to indicate the scan type to `SR_ScanNet`, available values are described in Table 4.12 |
| `SR_STACK_IND_ID_TAG` | `UINT16` | It is the type used to indicate the stack event identifier to the `SR_STACK_CALLBACK_FP`, available values are described in Table 4.20 |
| `SR_VERSION_T` | `char [20]` | String returned by `SR_Ver` to provide library version |

**Table 4.1**

##### 4.2.1.1.1.1 SR_VERSION_T

`SR_VERSION_T` is used by `SR_Ver` to pass the library version.
The string returned is composed as follow:
“*XX.YY.ZZ.KKJ*” e.g.: 27.00.01.RC4
XX: Technology ID, available tech types are listed in Table 4.2
YY: Major number
ZZ: Minor number
KK: Version type (internal use only)
J: Version type number (internal use only)

| Description | ID |
|---|---|
| ZigBee Identifier | 28 |
| MeshLite Identifier | 27 |
| M-Bus identifier | 2F |

**Table 4.2**

#### 4.2.1.1.2 Enumerations

The enumerations defined in “SRdata.h” are listed in Table 4.3.

| Enum | Description |
|---|---|
| `SR_RESET_TYPE_E` | Provides available reset types |

| | |
|---|---|
| `SR_MODULE_TYPE_E` | Provides available module types |
| `SR_STATUS_TYPE_E` | Provides available values returned by every library function |

**Table 4.3**

#### 4.2.1.1.2.1 SR_RESET_TYPE_E

`SR_RESET_TYPE_E` is used by `SR_Reset` to indicate what type of reset will be done.
The `SR_RESET_TYPE_E` values are described in Table 4.4.

| Name | Value | Description |
|---|---|---|
| `SR_RT_HARD` | `0x00` | It is the identifier for a hard reset |
| `SR_RT_SOFT` | `0x01` | It is the identifier for a soft reset |

**Table 4.4**

#### 4.2.1.1.2.2 SR_MODULE_TYPE_E

`SR_MODULE_TYPE_E` is used by `SR_ScanNet` to indicate what type of module has been found.
The values are described in Table 4.5.

| Name | Value | Description |
|---|---|---|
| `COORDINATOR` | `0x01` | It is the identifier for a Coordinator |
| `ROUTER` | `0x02` | It is the identifier for a Router |
| `ENDDEVICE` | `0x03` | It is the identifier for a Enddevice |

**Table 4.5**

NB: Only for the ZigBee technology: 0x02 identifies a node of the tree and 0x03 a leaf.

#### 4.2.1.1.2.3 SR_STATUS_TYPE_E

`SR_STATUS_TYPE_E` is the type returned by each API function. The values are described in Table 4.6.

| Name | Value | Description |
|---|---|---|
| `SR_STATUS_SUCCESS` | `0` | Generic success value returned by a function |
| `SR_STATUS_ERROR` | `-1` | Generic error value returned by a function |
| `SR_STATUS_TIMEOUT` | `-2` | Error value returned by a function when a timeout occurs |
| `SR_STATUS_BAD_PARAM` | `-3` | Error value returned by a function when a wrong parameter is passed by the user |

| | | |
|---|---|---|
| `SR_STATUS_NWK_ALREADY_RUNNING` | `-4` | Error value returned by a function when try a SR_StartNet without stopping the existing SR network |
| `SR_STATUS_NWK_ALREADY_STOPPED` | `-5` | Error value returned by SR_Reset(), SR_ScanNet(), SR_SendData() functions when there is not a SR network running. |
| `SR_STATUS_BAD_CONF_PARAM` | `-6` | Error value returned when in the configuration file a wrong param is read |

**Table 4.6**

#### 4.2.1.1.3 Structures

The structures defined in “SRdata.h” are listed in Table 4.7.

| Name | Description |
|---|---|
| `SR_DATA_PACKET_T` | Used to send and receive data packets |
| `SR_SCAN_RES_T` | Used to return scan result |
| `SR_SCAN_INFO_T` | Used to hold scan info of a single node |

**Table 4.7**

##### 4.2.1.1.3.1 SR_DATA_PACKET_T

`SR_DATA_PACKET_T` is used by `SR_SendData`, `SR_ReceiveData` and `SR_DATA_CALLBACK_FP` to send or receive data packets.
The fields of `SR_DATA_PACKET_T` structure are described in Table 4.8.

| Field Name | Field Type | Description |
|---|---|---|
| `SRnwkAddr` | `UINT16` | If the SR_SendData API is used it is the network address of the destination node. If the SR_ReceiveData API or the SR_DATA_CALLBACK is used it is the network address of the source node. For the M-Bus library, the network address corresponds to the Manufacturer Id. |
| `SRpar1` | `UINT16` | First and second byte of A-Field (only for M-Bus) |
| `SRpar2` | `UINT16` | Third and fourth byte of A-Field (only for M-Bus) |
| `SRpar3` | `UINT8` | Fifth byte of A-Field (only for M-Bus) |
| `SRpar4` | `UINT8` | Sixth byte of A-Field (only for M-Bus) |
| `SRpar5` | `UINT8[5]` | Reserved for future usage |

| | | |
|---|---|---|
| SRlength | UINT16 | Number of data bytes.<br>***Note:***<br>-Range 1-84 for ZigBee (without fragmentation service)<br>-Range 1-241 for ZigBee (with fragmentation service, at the moment, due to a limitation of the ZigBee firmwares the fragmentation service is not managed)<br>-Range 1-660 for MeshLite (when used by SR_sendData)<br>-Range 1-250 for MeshLite (when used by SR_Receive_Data and DATA_CALLBACK)<br>-Range 2-247 for M-Bus |
| SRdata | UINT8[680] | Data buffer |

**Table 4.8**

***Important:*** The maximum value of SRlength depends on the specific short range technology. Table 4.9 explains limits for different short range technologies.

| SR Technology | Max SRlength value | Notes |
|---|---|---|
| ZigBee | 84 or 241 | Refer to paragraph § 4.3.1.2.1 for futher details. |
| MeshLite | -660<br>-250 | - when used by SR_sendData<br>- when when used by SR_Receive_Data and DATA_CALLBACK |
| M-Bus | 247 | Refer to paragraph § 4.5.4 for futher details. |

**Table 4.9**

#### 4.2.1.1.3.2 SR_SCAN_RES_T

SR_SCAN_RES_T is used by SR_ScanNet to pass information about every node found in the network.
It should be allocated by the the application that uses the library.

The fields of SR_SCAN_RES_T structure are described in Table 4.10.

| Field Name | Field Type | Description |
|---|---|---|
| SRnodeAwakeCount | UINT16 | Number of nodes awake |
| SRnodeSleepCount | UINT16 | Number of nodes that can sleep |
| SRnodeAddresses_pp | SR_SCAN_INFO_T** | Pointer to the list of information about awake devices. It is allocated by the library |

| | | |
|---|---|---|
| SRnodeSleepAddresses_pp | SR_SCAN_INFO_T** | Pointer to the list of information about sleeping devices. It is allocated by the library |

**Table 4.10**

***Note:*** The memory held by this structure shall be freed using SR_ScanResFree.

#### 4.2.1.1.3.3 SR_SCAN_INFO_T

SR_SCAN_INFO_T holds all the information about a node of the network; it is used by SR_SCAN_RES_T to pass the information of every node found in the network.
The fields of SR_SCAN_INFO_T structure are described in Table 4.11.

| Field Name | Field Type | Description |
|---|---|---|
| SRnwkAddr | UINT16 | Network address |
| SRhwAddrLen | UINT8 | Length of hardware address |
| SRhwAddr | UINT8[12] | Hardware address that is technology dependent (Little Endian) |
| SRparentNwkAddr | UINT16 | Parent network address |
| SRtype | SR_MODULE_TYPE_E | Module type; available values are listed in Table 4.5. |
| SRchildrenNum | UINT8 | Number of direct children |

**Table 4.11**

### 4.2.1.1.4 Symbolic Constants

The symbolic constants defined in “SRdata.h” are listed in  Table 4.12.

Table 4.12 describes symbolic constants defined for generic scan types available for SR_ScanNet.

| Name | Value | Description |
|---|---|---|
| SR_SCAN_TYPE_DISCOVERY | 0x00 | It is the identifier to discover every node in the network. |

**Table 4.12**

***Note:*** General scan type IDs are in the range 0x00-0x0F

### 4.2.1.1.5 Macros

Table 4.13 describes macros to redefine names of basic types provided by `sys/types.h` in crosstools for Pro[3] platform.

| Name | Type | Description |
|---|---|---|
| `INT8` | `char` | 8 bit integer |
| `INT16` | `short` | 16 bit integer |
| `INT32` | `int` | 32 bit integer |
| `INT64` | `quad` | 64 bit integer |
| `UINT8` | `u_char` | Unsigned 8 bit integer |
| `UINT16` | `u_short` | Unsigned16 bit integer |
| `UINT32` | `u_int` | Unsigned 32 bit integer |
| `UINT64` | `u_quad` | Unsigned 64 bit integer |

**Table 4.13**

### 4.2.1.1.6 Callbacks

Callbacks defined in “SRdata.h“ are listed in Table 4.14.

| Function Pointer | Type | Description |
|---|---|---|
| `SR_DATA_CALLBACK_FP` | `Function pointer` | It is the type that defines the data callback. |
| `SR_STACK_CALLBACK_FP` | `Function pointer` | It is the type that defines the stack event callback. |

**Table 4.14**

#### 4.2.1.1.6.1 SR_DATA_CALLBACK_FP

`SR_DATA_CALLBACK_FP` is used by the `SR_Init` to register the name of the callback to manage data.
The definition of `SR_DATA_CALLBACK_FP` is:

```
void (*SR_DATA_CALLBACK_FP)    (SR_DATA_PACKET_T *SRrecPacket_p  )
```

> ***Important:*** The callback task must not be blocking, for example infinite cicle, otherwise the library will not be able to receive other packets.

The input parameters shall be:

`< SRrecPacket_p >` It is the pointer to the data packet structure

> ***Important:*** the pointer `SRrecPacket_p` will be unallocated by the library when the callback returns so it can not be assigned to another pointer. In other words only data pointed by the pointer can be used.

#### 4.2.1.1.6.2 SR_STACK_CALLBACK_FP

`SR_STACK_CALLBACK_FP` is used by the `SR_Init` to register the name of the callback to manage stack event. This callback has no effect on MeshLite and M-Bus because these technologies do not generate stack indications.
The definition of `SR_STACK_CALLBACK_FP` is:

```
Void(*SR_STACK_CALLBACK_FP)     (SR_STACK_IND_ID_TAG SRstackIndId,
                                void *SRstackIndPar_p )
```

> ***Important:*** The callback task must not be blocking, for example infinite cicle, otherwise the library will not be able to receive other packets.

The input parameters shall be:

`< SRstackIndId >` It identifies the stack event received in order to understand the type of the structure pointed by `SRstackIndPar_p`.
ZigBee stack event are described in Table 4.20.

`< SRstackIndPar_p >` It will hold the pointer to the structure that holds the stack event parameters. The type of the structure pointed is defined by `SRstackIndId`

> ***Important:*** the pointer `SRstackIndPar_p` will be deallocated by the library when the callback returns so it can not be assigned to another pointer.

A list of stack events supported until now is shown in Table 4.15.

| Stack Event ID | Struct Type Passed | Technology | Reference |
|---|---|---|---|
| `SRZB_STACK_IND_DEV_ANN` | `SRZB_DEV_ANNCE_T` | ZigBee | Table 4.20 |

**Table 4.15**

### 4.2.2 Functions Summary

Functions provided by Generic API are listed in Table 4.16.

| Type | Function Name | Description |
|---|---|---|
| Generic API | `SR_Init` | Initialize the short range subsystem |
| | `SR_Close` | Release short range resources |
| | `SR_StartNet` | Start the short range network |
| | `SR_Reset` | Reset the short range hardware |
| | `SR_ScanNet` | Scan the short range network |
| | `SR_SendData` | Send data toward a short range node |
| | `SR_ReceiveData` | Receive data from the short range network |
| | `SR_Ver` | Retrieve Short Range Library version |
| | `SR_ScanResFree` | Free memory holding scan result |

**Table 4.16**

## 4.2.3 Functions Description

### 4.2.3.1 SR_Init

This function allocates and initializes all the Short Range Library resources.
`SR_Init` allows registering two callbacks, one manages stack event packets (valid only for ZigBee) and the other manages data packets.
The configuration parameters for the network are passed through a configuration file (SRtech.conf) that is technology dependent.
The description how to write SRtech.conf file is provided in a specific sub paragraph of every specific technology (ZigBee, MeshLite and M-Bus).

> ***NOTE:*** If the stack event callback is registered by the user the header file for the specific technology has to be included because stack events are technology dependent (E.g. SRZBdata.h for the ZigBee technology).

> ***NOTE:*** `SR_Init` has to be called before every other function of the library otherwise any other call to another function of the library (excluded `SR_Ver` and SR_`ScanResFree` APIs) will return `SR_STATUS_ERROR`.

***NB:*** `SR_Init` will return an error if the "Sr_XX_Library" has been already initialized.

#### 4.2.3.1.1 Prototype

The prototype of `SR_Init` is:

```
SR_STATUS_TYPE_E SR_Init(   SR_DATA_CALLBACK_FP
                            SrdataCallback,
                            SR_STACK_CALLBACK_FP
                            SRstackCallback,
                            UINT8 *SRpathConfDir_p  )
```

#### 4.2.3.1.2 Parameters

The input parameters are three:

`< SrdataCallback >` Is the callback to manage data packet received from the short range network. If it is NULL the data shall be read using the function `SR_ReceiveData`. If it is a function pointer every call to `SR_ReceiveData` will return `SR_STATUS_ERROR`

`< SRstackCallback >` Is the callback to manage stack events received from short range stack of the module managed through the library. If it is NULL stack events will not be managed (valid only for ZigBee technology)

`< SRpathConfDir_p >` Is a string (it shall be terminated with the “\0“ character) that provide the absolute path of the directory that holds the configuration file SRtech.conf. If it is NULL the function will search in the directory that holds the application which is using the library for a file named SRtech.conf

The interfaces of `SRdataCallback` and `SRstackCallback` are described in § 4.2.1.1.6.1 and § 4.2.1.1.6.2, respectively.

#### 4.2.3.1.3 Return Values

The function returns `SR_STATUS_SUCCESS` if the initialization succeeds. Otherwise it returns `SR_STATUS_BAD_PARAM` if there are some errors in the parameters passed to the function, or `SR_STATUS_BAD_CONF_PARAM` if parameters specified in file SRtech.conf are invalid. More details about configuration files are in § 4.3.1.2 for ZigBee, in § 4.4.1.2 for MeshLite and in § 4.5.3 for M-Bus.
If the file SRtech.conf doesn’t exist or it can’t be opened, a `SR_STATUS_BAD_PARAM` error will be returned.
For other types of error `SR_STATUS_ERROR` will be returned.

#### 4.2.3.1.4 Example

```
/********** Defines the callback function for data events
***********/
void DataCallBack(SR_DATA_PACKET_T *SRrecPacket_p)
{
        UINT8 i = 0;

        printf("\n\r  DATA CALL BACK \n\r");

        printf("\n\rNwk addr of source node is
        %x\n\r",((SR_DATA_PACKET_T*)(SRrecPacket_p))->SRnwkAddr);

        printf("\n\rData lenght is %d\n\r",
        ((SR_DATA_PACKET_T*)(SRrecPacket_p))->SRlength);

        printf("\n\rData received :\n\r");
        for(i=0;i<(((SR_DATA_PACKET_T*)(SRrecPacket_p))-
        >SRlength);i++)
        {
                printf("\n\r%x\n\r",
                ((SR_DATA_PACKET_T*)(SRrecPacket_p))->SRdata[i]);
        }

        return;
}
/********** Define the callback function for stack events
***********/
void StackCallBack(SR_STACK_IND_ID_TAG SRstackIndId, void
*SRstackIndPar_p)
{
        printf("\n\r STACK CALL BACK RUNNING \n\r");

        printf("\n\rSR_STACK_IND_ID is %x\n\r", SRstackIndId);

        if(SRstackIndId == SRZB_STACK_IND_DEV_ANN)
        {
                printf("\n\rA new device has joined the network\n\r");

                printf("\n\rNwk addr is
                %x\n\r",((SRZB_DEV_ANNCE_T*)(SRstackIndPar_p))-
                >SRZBnwkAddr);

                printf("\n\rHw address is %llx\n\r",
                ((SRZB_DEV_ANNCE_T*)(SRstackIndPar_p))->SRZBieeeAddr);
```

```
                printf("\n\rCapability is %x\n\r",
                ((SRZB_DEV_ANNCE_T*)(SRstackIndPar_p))-
                >SRZBcapability);
        }

        return;
}

/************ Call the SR_Init() inside the main() function
*********/
void SR_Init_Example(void)
{
        SR_STATUS_TYPE_E eReturnCode = SR_STATUS_ERROR;
        UINT8 path[200];

        /* Declare function pointers */
        SR_STACK_CALLBACK_FP stackCallback;
        SR_DATA_CALLBACK_FP dataCallback;

        /* Assign the value to function pointers */
        stackCallback = StackCallBack;
        dataCallback = DataCallBack;

        /* Clear the path variable */
        memset(path,0,sizeof(path));
        /* Assign the value to path variable */
        strcpy((char *)path,"/");

        /* Call the SR_Init() */
        if ((eReturnCode = SR_Init(dataCallback, stackCallback,
        path)) ==   SR_STATUS_SUCCESS)
        {
                /* System has been initialized */
                ;
        }
        else
        {
                /* System has not been initialized */
                ;
        }

        return;
}
```

### 4.2.3.2 SR_Close

This function allows closing the communication with the short range technology and releasing every resource allocated with a previous call to SR_Init. If there is a short range network running, it will not be stopped with SR_Close , in order to stop the SR netwok the SR_Reset API shall be used.
However, due to closing of the communication, all messages coming from the SR network will be lost.
After a SR_Close it is possible to reconnect to a SR Network already running using the SR_Init function.

**NB**: SR_Close will return an error if there is no resource to release.

#### 4.2.3.2.1 Prototype

The prototype of SR_Close is:

```
SR_STATUS_TYPE_E SR_Close()
```

#### 4.2.3.2.2 Parameters

SR_Close does not have parameters.

#### 4.2.3.2.3 Return Values

The function returns SR_STATUS_SUCCESS if the closure succeeds otherwise it returns SR_STATUS_ERROR.

#### 4.2.3.2.4 Example

```
void SR_Close_Example(void)
{
SR_STATUS_TYPE_E eReturnCode = SR_STATUS_ERROR;

if((eReturnCode = SR_Close()) == SR_STATUS_SUCCESS)
{
    /* System resources have been released */
}
else
{
    /* System resources have not been released */
}

return;
}
```

### 4.2.3.3 SR_StartNet

This function allows starting the short range network.

#### 4.2.3.3.1 MeshLite behaviour

When the network used is based on MeshLite technology, the `SR_StartNet` can be called in each moment after a `SR_Init`. There is no limitation related to the use of this function.
The only reasons of failure of the `SR_StartNet` function are due to possible communication error with the coordinator or to configuration error.

#### 4.2.3.3.2 ZigBee behaviour

If the ZigBee technology is used, the `SR_StartNet` will be effective only when there isn't SR Network already running; otherwise `SR_STATUS_NWK_ALREADY_RUNNING` will be returned and `SR_StartNet` will have no effect.

#### 4.2.3.3.3 M-Bus behaviour

Since in Wireless M-Bus technology a network can not be started or stopped, `SR_StartNet` returns always `SR_STATUS_ERROR`.

#### 4.2.3.3.4 Prototype

The prototype of `SR_StartNet` is:

```
SR_STATUS_TYPE_E SR_StartNet()
```

#### 4.2.3.3.5 Parameters

`SR_StartNet` does not have parameters.

#### 4.2.3.3.6 Return Values

The function returns `SR_STATUS_SUCCESS` if the network start succeed otherwise it returns `SR_STATUS_ERROR`.

#### 4.2.3.3.7 Example

```
int SR_StartNet_Example(void)
{
SR_STATUS_TYPE_E eReturnCode = SR_STATUS_ERROR;

if((eReturnCode = SR_StartNet()) == SR_STATUS_SUCCESS)
{
    /* Short range network has been started */
}
```

```
else
{
    /* Short range network has not been started */
}

return 0;
}
```

#### 4.2.3.4 SR_Reset

This function allows resetting the short range hardware. The reset could be hard (`SR_RT_HARD)` or soft (`SR_RT_SOFT).`
A hard reset stops the network, resets every configuration parameters on the short range module to the default factory values and reloads the values from configuration file (SRtech.conf) in the Library. In this way, if the user changes the parameters in the SRtech.conf file, at the next `SR_StartNet` new parameters will be set on the coordinator.
The soft reset stops only the network and resets every configuration parameters on the short range module to the default factory values.

##### 4.2.3.4.1 MeshLite behaviour

When the network used is based on MeshLite technology, the `SR_Reset` can be called in each moment after a `SR_Init`. There is no limitation related to the use of this function. Hard reset means that the routing table will be cleared, all the registers will be restored to the factory default and that the values from config file will be reloaded in the ML_Library. Soft reset performs the same operations as hard reset except the reloading of values from config file.
Please also note that, using MeshLite technology, the `SR_Reset` will not result in a network stop: the coordinator will not see the network until it receives new association frames from end devices. At the end of this new association process, which may last from 0 to 40 minutes, the network will be restored.

##### 4.2.3.4.2 ZigBee behaviour

If the ZigBee technology is used, the `SR_Reset` will be effective only when there is a SR Network running; otherwhise `SR_STATUS_NWK_ALREADY_STOPPED` will be returned and `SR_Reset` will have no effect.
If at least a Router is associated to the ZigBee network, this function will not stop the network.
In this case `SR_Reset` removes association between Coordinator and ZigBee network, resets every configuration parameters and, if hard reset is used, reloads values from configuration file in the SR library.
In order to stop the network each router shall be switched off or reset one by one, acting

directly through its serial interface.

#### 4.2.3.4.3 M-Bus behaviour

Since in Wireless M-Bus technology a network can not be started or stopped, `SR_Reset` does not change the network status; moreover, configuration parameters of the short range module are not reset to their default value. If the data callback is not used, every data packet received from the short range module but not handled by a call to `SR_ReceiveData` will be lost.

#### 4.2.3.4.4 Prototype

The prototype of `SR_Reset` is:

```
SR_STATUS_TYPE_E SR_Reset(  SR_RESET_TYPE_E SRresetType )
```

#### 4.2.3.4.5 Parameters

The input parameter is:

| | |
|---|---|
| `< SRresetType >` | If it is set to `SR_RT_SOFT` a soft reset is made else if it is set to `SR_RT_HARD` a hard reset is made |

#### 4.2.3.4.6 Return Values

The function returns `SR_STATUS_BAD_PARAM` if a parameter passed by the user is wrong, `SR_STATUS_SUCCESS` if the network reset succeed otherwise it returns `SR_STATUS_ERROR`.

#### 4.2.3.4.7 Example

```
void SR_Reset_Example(void)
{
    SR_STATUS_TYPE_E eReturnCode = SR_STATUS_ERROR;

    SR_RESET_TYPE_E SRresetType = SR_RT_HARD;

    eReturnCode = SR_Reset(SRresetType);

    /* Check result */
    if (eReturnCode  ==   SR_STATUS_SUCCESS)
    {
        printf("\n\r Reset OK \n\r");
    }
    else
    {
        printf("\n\r Reset NOT OK \n\r");
    }


    return;
}
```

### 4.2.3.5 SR_ScanNet

This function allows scanning the short range network. The scan means discover devices in the short range network, also the coordinator will be returned in the modules list. The coordinator module has not parent, then the field SRparentNwkAddr does not matter (it is possible to find the same id of the coordinator into its field SRparentNwkAddr). Depending on the specific technology different types of scan are available. This function returns error when using M-Bus, because scanning is not possible with this technology.

#### 4.2.3.5.1 Prototype

The prototype of `SR_ScanNet` is:

```
    SR_STATUS_TYPE_E SR_ScanNet(  SR_SCAN_TYPE_TAG SRscanType,

                                  SR_SCAN_RES_T *SRscanRes_p,

                                  UINT32 SRtimeOut )
```

#### 4.2.3.5.2 Parameters

The input parameters are:

| | |
|---|---|
| < SRscanType > | It can depend on the short range technology; all the allowed values are shown in Table 4.12 |
| < SRscanRes_p > | It is a pointer to a structure that holds the scan result. It should be allocated by the application that uses the library. |
| < SRtimeOut > | Timeout in seconds. Its meaning depends on the `SRscanType` and it is described in Table 4.17. The value of this timeout shall not be 0 and it shall be smaller than 36000 (seconds).<br>In the MeshLite version is used as timeout for each command sent to each module. |

| **SRscanType** | **Technology** | **SRtimeOut meaning** |
|---|---|---|
| SR_SCAN_TYPE_DISCOVERY | All | It is the maximum wait time for a response from a device during a scan |

**Table 4.17**

> ***Important:*** Before calling the `SR_ScanNet` the user shall call the macro `SR_New_SR_SCAN_RES_T(MyVar)`: it declares and initializes the variable MyVar in the correct way. After a successful call to the function, if some remote device has been discovered, memory pointed by `SRscanRes_p` shall be freed by the user using `SR_ScanResFree` before a new call to the function, otherwise `SR_STATUS_BAD_PARAM` will be returned.

### 4.2.3.5.3 Return Values

The function returns `SR_STATUS_SUCCESS` if the network scan succeed, it returns `SR_STATUS_ERROR` if an error occurs or `SR_STATUS_BAD_PARAM` if some parameter is wrong. Also if there aren’t remote devices to discover the function returns `SR_STATUS_SUCCESS` but the lists of discovered devices will be empty except for the awake modules list, it will contain the coordinator.
If the ZigBee technology is used and there isn’t short range network running the function returns `SR_STATUS_NWK_ALREADY_STOPPED`.
If M-Bus is used, the function returns `SR_STATUS_ERROR`.

### 4.2.3.5.4 Example

```
void SR_ScanNet_Example(void)
{
    SR_STATUS_TYPE_E eReturnCode = SR_STATUS_ERROR;

    /* Declare and initialize a variable that will contain the
result of the SR_ScanNet() */
    SR_New_SR_SCAN_RES_T(SRscanRes);
    /* It means:
SR_SCAN_RES_T SRscanRes =
{
  .SRnodeAwakeCount = 0,
  .SRnodeSleepCount = 0,
  .SRnodeAddresses_pp = NULL,
  .SRnodeSleepAddresses_pp = NULL,
}
     */
    SR_SCAN_TYPE_TAG SRscanType= SR_SCAN_TYPE_DISCOVERY;
    UINT32 SRtimeout = 10;
    UINT8 i = 0;
    UINT8 j = 0;

    eReturnCode = SR_ScanNet(SRscanType, &SRscanRes, SRtimeout);

    if (eReturnCode == SR_STATUS_SUCCESS)
    {
        printf("\n\rThere are %d awake devices\n\r",
SRscanRes.SRnodeAwakeCount);

        for (i=0; i<(SRscanRes.SRnodeAwakeCount); i++)
        {
            printf("\n\r Awake device %d has nwk address %x \n\r and
hw address     :\n\r",(i+1), (SRscanRes.SRnodeAddresses_pp[i])-
>SRnwkAddr);
            for (j=0; (j<(SRscanRes.SRnodeAddresses_pp[i])-
>SRhwAddrLen);       j++)
            {
                printf(" - %x\n\r",
(SRscanRes.SRnodeAddresses_pp[i])->SRhwAddr[j]);
            }
            printf("\n\r    Its parent has network address %x\n\r",
(SRscanRes.SRnodeAddresses_pp[i])->SRparentNwkAddr);

            printf("\n\r    Its node type is %d\n\r",
(SRscanRes.SRnodeAddresses_pp[i])->SRtype);

            printf("\n\r    It have %d children\n\r", (SRscanRes.
SRnodeAddresses_pp[i])->SRchildrenNum);
        }

        printf("\n\rThere are %d sleeping devices\n\r",
SRscanRes.SRnodeSleepCount);

        for (i=0; i<(SRscanRes.SRnodeSleepCount); i++)
        {

            for (j=0; (j<(SRscanRes.SRnodeSleepAddresses_pp[i])-
>SRhwAddrLen);  j++)
            {
                printf("\n\r Sleeping device %d has nwk address %x
\n\r", (i+1),    (SRscanRes.SRnodeSleepAddresses_pp[i])->SRnwkAddr);
            }
```

```
            printf("\n\r    Its parent has network address 
%x\n\r",(SRscanRes.  SRnodeSleepAddresses_pp[i])->SRparentNwkAddr);
        }
    }
    else
    {
        /* An error occured during the network scan */
        ;
    }

    /********* Free system resources **********/

    /* Free the array of devices discovered by the SR_ScanNet() */
    eReturnCode = SR_ScanResFree(&SRscanRes);
    if (eReturnCode == SR_STATUS_SUCCESS)
    {
        /* Array of devices returned by the SR_ScanNet() has been 
freed */
        ;
    }
    else
    {
        /* An error occured. Array of devices has not been freed */
        ;
    }

    return;
}
```

### 4.2.3.6 SR_SendData

This function allows sending data to another node or to itself.

Only for ZigBee technology:
If the fragmentation service has been activated, `SR_SendData` accepts a SR_DATA_PACKET_T parameter with `Srlength` field up to 241 (at the moment, due to a limitation of the ZigBee firmwares the fragmentation service is not not managed). Otherwise the max value of `Srlength` field will be 84.
In order to activate the fragmentation service refer to paragraph § 4.3.1.2.1

Only for MeshLite technology:
Using the SR_SendData in order to send data, the user shall send data with a maximum length of MaxPacketDataLength bytes (refer to paragraph 4.4.1.2) and he shall wait for at least a delay of 100ms before sending next packet data. Otherwise unexpected behavior

can happen

#### 4.2.3.6.1 Prototype

The prototype of `SR_SendData` is:

```
SR_STATUS_TYPE_E SR_SendData(      SR_DATA_PACKET_T
                                   *SRdataPacket_p,

                                   UINT32 SRtimeOut )
```

#### 4.2.3.6.2 Parameters

The input parameters are:

`< SRdataPacket_p >` It is a pointer to a structure that holds the destination network address and the data to send, this structure is described in Table 4.8 It should be allocated by the application that uses the library.

`< SRtimeOut >` Timeout in seconds. If it is 0 the function checks for the confirm from lower layers only once without retry (in this case a timeout error may be returned). If it is bigger than 0 the function waits for confirm up to the timeout or up to an error message from lower layers. This behaviour is valid only for ZigBee, in the MeshLite and M-Bus versions this parameter is not used.

#### 4.2.3.6.3 Return Values

The function returns `SR_STATUS_SUCCESS` if the data sending succeeds otherwise it returns `SR_STATUS_ERROR` if an error occurs or `SR_STATUS_TIMEOUT` if the time out expired or `SR_STATUS_BAD_PARAM` if some parameter is wrong.
If ZigBee technology is used and there isn't short range network running the function returns `SR_STATUS_NWK_ALREADY_STOPPED`.
If MeshLite or M-Bus technologies are used there is no way to know if the packet has been sent to an existing module. In MeshLite the Coordinator sends the packet broadcast but it can't know if the recipient module exists in the network. It is a limit of these types of technology; hence, once the data has been sent in the air, MeshLite and M-Bus SR libraries will return always `SR_STATUS_SUCCESS`.

#### 4.2.3.6.4 Example

```
void SR_SendData_Example(void)
```

```
{
    SR_STATUS_TYPE_E eReturnCode = SR_STATUS_ERROR;
    UINT32 SRtimeout = 10;
    SR_DATA_PACKET_T *SRdataPacket_Send;

    /* Allocate space for the struct that will contains data to send
*/
    SRdataPacket_Send = malloc(sizeof(SR_DATA_PACKET_T));

    /* Set the network address of the recipient */
    SRdataPacket_Send->SRnwkAddr = 0x796F;

    /* Set the length of data to send. NB: the max value allowed is
256 */
    SRdataPacket_Send->SRlength = 0x05;

    /* Set a generic data message */
    SRdataPacket_Send->SRdata[0] = 0x31;
    SRdataPacket_Send->SRdata[1] = 0x32;
    SRdataPacket_Send->SRdata[2] = 0x33;
    SRdataPacket_Send->SRdata[3] = 0x34;
    SRdataPacket_Send->SRdata[4] = 0x35;

    if((eReturnCode = SR_SendData(SRdataPacket_Send, SRtimeout)) ==
SR_STATUS_SUCCESS)
    {
        /* Data packet has been sent correctly */
        ;
    }
    else
    {
        /* An error occured during send data */
        ;
    }

    /********* Free system resources **********/

    free(SRdataPacket_Send);

    return;
}
```

#### 4.2.3.7 SR_ReceiveData

This function allows receiving data from any node of the SR network. If the `SR_DATA_CALLBACK_FP` is passed to `SR_Init` this function cannot be used and will return every time `SR_STATUS_ERROR`.

***Important:*** The function retrieves the first data message received (one by one), and sets the field `SRnwkAddr` to the network address of the sender.

Only for ZigBee technology:
At the moment, due to a limitation of the ZigBee firmwares the length of the data packet shall not be bigger than 84 bytes.

Only for MeshLite technology:
if a node sends a packet data of length bigger than 250 bytes, the user will receive maximun 250 bytes at avery SR_ReceiveData (to receive all data, call SR_ReceiveData more times)

#### 4.2.3.7.1 Prototype

The prototype of `SR_ReceiveData` is:

```
SR_STATUS_TYPE_E SR_ReceiveData      SR_DATA_PACKET_T *SRdataPacket_p,
(
                                     UINT32 SRtimeOut )
```

#### 4.2.3.7.2 Parameters

The input parameters are:

`< SRdataPacket_p >` It is a pointer to a structure that will hold the network address of the sender and the data received from the network, this structure is described in Table 4.8. It should be allocated by the application that uses the library.

`< SRtimeOut >` Time out in seconds. If it is 0 the function checks for incoming data only once without retry (in this case a timeout error may be returned). If it is bigger than 0 the function waits for incoming data up to the timeout or up to an error message from lower layers.

#### 4.2.3.7.3 Return Values

The function returns `SR_STATUS_SUCCESS` if the receiving succeeds otherwise it returns

SR_STATUS_ERROR if an error occurs or SR_STATUS_TIMEOUT if the time out expired or SR_STATUS_BAD_PARAM if some parameter is wrong.
If the ZigBee technology is used and there isn't short range network running the function returns SR_STATUS_NWK_ALREADY_STOPPED.

#### 4.2.3.7.4 Example

```
void SR_ReceiveData_Example()
{
SR_STATUS_TYPE_E eReturnCode = SR_STATUS_ERROR;
UINT32 SRtimeout = 10;
SR_DATA_PACKET_T * SRdataPacket_Receive;
UINT8 i = 0;

/* Allocate space for the struct that will contains received data */
SRdataPacket_Receive = malloc(sizeof(SR_DATA_PACKET_T));

if((eReturnCode = SR_ReceiveData(SRdataPacket_Receive, SRtimeout))
== SR_STATUS_SUCCESS)
{

    printf("\n\rNwk addr of source node is
%x\n\r",((SR_DATA_PACKET_T*)(SRdataPacket_Receive))->SRnwkAddr);

    printf("\n\rData lenght is %d\n\r",
((SR_DATA_PACKET_T*)(SRdataPacket_Receive))->SRlength);

    printf("\n\rData received :\n\r");
    for(i=0;i<(((SR_DATA_PACKET_T*)(SRdataPacket_Receive))-
>SRlength);i++)
    {
      printf("\n\r%x\n\r",
((SR_DATA_PACKET_T*)(SRdataPacket_Receive))->SRdata[i]);
    }
}
else
{
    /* An error occured while receiving data */

    printf("\n\r SR_ReceiveData() returns %d \n\r", eReturnCode);
}

/********* Free system resources **********/

free(SRdataPacket_Receive);
return;
}
```

## 4.2.3.8 SR_Ver

This function allows to retrieve the version of the library.

### 4.2.3.8.1 Prototype

The prototype of `SR_Ver` is:

```
        SR_STATUS_TYPE_E SR_Ver(   SR_VERSION_T SRversion )
```

### 4.2.3.8.2 Parameters

The input parameters are:

`< SRversion_p >` It is an array of chars (string) that will hold information about version of short range library. The description of the structure is in 4.2.1.1.1.1

### 4.2.3.8.3 Return Values

The function returns `SR_STATUS_SUCCESS` if succeeds otherwise it returns `SR_STATUS_ERROR`.

### 4.2.3.8.4 Example

```
void SR_Ver_Example(void)
{
     SR_STATUS_TYPE_E eReturnCode = SR_STATUS_ERROR;
     SR_VERSION_T SRversion;

     if((eReturnCode = SR_Ver(SRversion)) == SR_STATUS_SUCCESS)
     {
          /* Information about library version has been retrieved */
          printf("\n\rMain version is %s\n\r", SRversion);
     }
     else
     {
          /* An error occured. Information about library version has
not been      retrieved */
          ;
```

```
        }
}
```

### 4.2.3.9 SR_ScanResFree

This function doesn't interact with the SR network, it only allows to free the lists of devices contained in the struct that holds scan result (`SR_SCAN_RES_T`).
It is necessary to call `SR_ScanResFree` after a successful call to the function `SR_ScanNet`, if some remote device has been discovered and before a new call to the function `SR_ScanNet`, otherwise `SR_STATUS_BAD_PARAM` will be returned by the scan function.

#### 4.2.3.9.1 Prototype

The prototype of `SR_ScanResFree` is:

```
   SR_STATUS_TYPE_E SR_ScanResFree(   SR_SCAN_RES_T *SRscanRes_p   )
```

#### 4.2.3.9.2 Parameters

The input parameter is:

< SRscanRes_p > It is the pointer previously passed to `SR_ScanNet`

#### 4.2.3.9.3 Return Values

The function returns `SR_STATUS_SUCCESS` if succeeds otherwise it returns `SR_STATUS_ERROR`. If M-Bus is used, the function always returns `SR_STATUS_ERROR`.

#### 4.2.3.9.4 Example

Refer to 0 for an example of usage with the SR_ScanNet() function.

# 4.3 ZigBee Specific API

## 4.3.1 Description

ZigBee Specific API provides the specific functionalities of the ZigBee technology.

### 4.3.1.1 Data Types

Data types defined for the ZigBee part of Short Range library are in header file "SRZBdata.h".

### 4.3.1.1.1 Structures

The structures defined in "SRZBdata.h" are listed in Table 4.18.

| Name | Description |
|---|---|
| `SRZB_DEV_ANNCE_T` | This structure holds data related to a new ZigBee device that has just joined the network. |

**Table 4.18**

#### 4.3.1.1.1.1 SRZB_DEV_ANNCE_T

`SRZB_DEV_ANNCE_T` is used by `SR_STACK_CALLBACK_FP` to pass data related to the received Device_Annce stack event (see §2.4.3.1.11 of ER[2]).
The fields of `SRZB_DEV_ANNCE_T` structure are described in Table 4.19.

| Field Name | Field Type | Description |
|---|---|---|
| `SRZBnwkAddr` | `UINT16` | Network address of the device that has joined the network |
| `SRZBieeeAddr` | `UINT64` | The IEEE802.15.4 address of the device that has joined the network (Big Endian) |
| `SRZBcapability` | `UINT8` | The capability of the device that has joined the network (see §2.4.3.1.11 of ER[2]) for more details. |

**Table 4.19**

### 4.3.1.1.2 Symbolic Constants

The symbolic constants defined in "SRZBdata.h" are listed in Table 4.20. describes symbolic constants defined for stack event identifier available for `SR_STACK_CALLBACK_FP`.

| Name | Value | Description |
|---|---|---|
| `SRZB_STACK_IND_DEV_ANN` | `0x1000` | It is the identifier of a Device_Annce stack event, see §2.4.3.1.11 ER[2] |

**Table 4.20**

***Note:*** ZigBee stack indication IDs are in the range 0x1000-0x1FFF

### 4.3.1.2 Configuration File

The SRtech.conf contains the parameters needed by the SR-Library to create a ZigBee network and to interact with ZigBee nodes. It is formatted as follows:

```
…
# It is a comment
[GROUP_TAG_1]
# It is a comment
KEY_TAG_1 = VALUE_1
# It is a comment
KEY_TAG_2 = VALUE_2
…
…
# It is a comment
[GROUP_TAG_2]
# It is a comment
KEY_TAG_1 = VALUE_1
# It is a comment
KEY_TAG_2 = VALUE_2
…
…
```

Where:

- `[GROUP_TAG_1]`is the identifier of a group of parameters that follow this tag.
- `KEY_TAG_1` is the identifier of a parameter
- `VALUE_1` is the value of the parameter

Comments are indicated with the `#` character.

The following table shows the parameters required by the ZigBee technology and their valid ranges.

| GROUP_TAG | KEY_TAG | M/O/U | Valid Range | Description |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| [Network] | `PanID` | `M` | `0 - 65535` | It is the identifier of the ZigBee network. |
| | `ChannelID` | `M` | `11 - 26` | It identifies the physical channel used by the ZigBee network |
| [UART] | `SerialPort` | `Unused` | | |
| [DataRequestParam] | `DstEndPoint` | `M` | `0 - 255` | It is the End Point of the remote ZigBee node |
| | `ProfileId` | `M` | `0 - 65535` | It is the identifier of the profile implemented in the coordinator |
| | `ClusterId` | `M` | `0 - 65535` | It identifies the commad sent to the remote ZigBee node. Refer to the ZigBee Cluster Library document for more details. |
| | `SrcEndPoint` | `M` | `0 - 255` | It is the End Point of the ZigBee Coordinator |
| | `TxOption` | `M` | `0 - 255` | Refer to paragraph § 4.3.1.2.1for details |
| | `Radius` | `M` | `0 - 255` | Refer to TelitRF document for ZigBee |

**Table 4.21**

Legenda:
M – Mandatory
O – Optional
U – Unused

NB: The order of KEY_TAGs inside a GROUP_TAG is not important. It is not possible to use the same name for two or more KEY_TAGs inside a single GROUP_TAG.

***Important:*** If one of mandatory values are not set or set with a wrong value the `SR_Init` function returns an error.
***Pay attention***: Do not insert space or other characters after the value; it can cause error in the `SR_Init` function.

### 4.3.1.2.1 Transmission options

The `TxOption` field of the configuration file sets the transmission options for the data message to send toward a remote ZigBee node. It will be read using the operator “bitwise AND” with the value 0x0D in order to enable one or more of the following features:

- Bit 0 : Security enabled transmission
- Bit 2: Acknowledged transmission
- Bit 3: Fragmentation service enabled

The Table 4.22 summarizes the usage of `TxOption` field:

| `TxOption` value | Operator | Result | Feature enabled |
|---|---|---|---|
| 0byyyyyyy1 | & 0x0D | 0b0000yy01 | Security transmission |
| 0byyyyy1yy | & 0x0D | 0b0000y10y | Acknowledged transmission |
| 0byyyy1yyy | & 0x0D | 0b00001y0y | Fragmentation service |

**Table 4.22**

Where y can be 0 or 1.

Fragmentation service splits a large data packet in smaller ones, in order to allow transmission over the air.

When sending `SR_DATA_PACKET_T` with `Srlength` field bigger than 84, the Fragmentation service shall be enabled. Otherwise the `SR_SendData` function will return an error.
This limitation doesn't concern the `SR_ReceiveData` function and `SR_DATA_CALLBACK_FP` callback. For these function, the `Srlength` field of `SR_DATA_PACKET_T` parameter, will be up to 241.

For details about Security transmission and Acknowledged transmission refer to ER[2].

At the moment, due to a limitation of the ZigBee firmwares the Fragmentation service is not managed. The length of the data packet shall not be bigger than 84 bytes for `SR_SendData`, `SR_ReceiveData` and `SR_DATA_CALLBACK_FP` callback, otherwise unexpected behaviour can happen.

# 4.4 MeshLite Specific API

## 4.4.1 Description

### 4.4.1.1 Data Types

Specific data types are not defined.

### 4.4.1.2 Configuration File

The SRtech.conf contains the parameters needed by the SR-Library to create a MeshLite network and to interact with MeshLite nodes. It is formatted as follows:

```
…
# It is a comment
[GROUP_TAG_1]
# It is a comment
KEY_TAG_1 = VALUE_1
# It is a comment
KEY_TAG_2 = VALUE_2
…
…
# It is a comment
[GROUP_TAG_2]
# It is a comment
KEY_TAG_1 = VALUE_1
# It is a comment
KEY_TAG_2 = VALUE_2
…
…
```

Where:

- `[GROUP_TAG_1]`is the identifier of a group of parameters that follow this tag.
- `KEY_TAG_1` is the identifier of a parameter
- `VALUE_1` is the value of the parameter

Comments are indicated with the `#` character.

The following table shows the parameters required by the MeshLite technology and their valid ranges.

| GROUP_TAG | KEY_TAG | M/O/U | Valid Range | Description |
|---|---|---|---|---|
| [Network] | `NetPeriod` | M | 0 - 65000 | Refer to ML documentation |
| | `BaseTime` | M | 3 - 7 | Refer to ML documentation |
| | `FrameSize` | M | 0 - 2 | Refer to ML documentation |

| | | | | |
|---|---|---|---|---|
| | NetId | M | 0 - 255 | Refer to ML documentation |
| [UART] | SerialPort | Unused | | |
| [DATA] | MaxPacketDataLength | M | 1 - 660 | Maximun packet data length for SR_SendData (for SR_ReceiveData and DATA_CALLBACK the Maximun packet data length is 250) |

**Table 4.23**

***Important:*** If one of these values are not set or set with a wrong value the SR_Init function returns an error.
***Pay attention***: Do not insert space or other characters after the value; it can cause error in the SR_Init function.

#### 4.4.1.3 How to send and receive raw data using MeshLite technology

As reported in IR[3], it is not possible to add any customized software on the Mesh Lite module. The only way to use these modules is to connect another “external CPU” to the serial port and then implement a custom protocol using Mesh Lite serial protocol features. The Mesh Lite serial protocol has the following format:

| | | | | |
|---|---|---|---|---|
| Byte Header | LSB Address | MSB address | PAYLOAD | CR |

In this format there is not any information about packet length or CRC, than the only way to recognize the end of packet is to wait for a carriage return character. For this reason the user shall not insert the '0x0D' as data into the data packet; if it will be necessary he shall implement and use a bit stuffing/destuffing algorithm to hide the '0x0D' character into the data stream packet, both when he sends data from GG863 to end device and when he sends data from end device to GG863. When the user sends data using the API SR_SendData he shall insert 0x0D as last byte of SRdata field of SR_DATA_PACKET_T structure. When the user receives data using the function SR_ReceiveData, he does not receive ‘0x0D’ as last byte of SRdata field. The SR library provides to remove this special character.

# 4.5 M-Bus Specific API

## 4.5.1 Description

M-Bus specific API provides the specific functionalities of the M-Bus technology.

### 4.5.1.1 Data Types

Data types defined for the M-Bus part of Short Range library are in header file “SRMBlibrary.h”.

#### 4.5.1.1.1 Enumerations

The enumerations defined in “SRMBlibrary.h” are listed in Table 4.24.

| Enum | Description |
|---|---|
| `SR_MODE_E` | Provides available operating modes for the M-Bus module |

**Table 4.24**

##### 4.5.1.1.1.1 SR_MODE_E

`SR_MODE_E` is used by `SR_SwitchMode` to indicate which operating mode will be activated.
The `SR_MODE_E` values are described in Table 4.25.

| Name | Value | Description |
|---|---|---|
| `COMMAND_MODE` | 1 | It is the identifier for command mode |
| `DATA_MODE` | 2 | It is the identifier for data mode |

**Table 4.25**

## 4.5.2 Functions Description

### 4.5.2.1 SR_SendCommand

This function sends an AT command to a module in command mode and waits for a received response. If this function is called to send the “ATO\r” command to enter data mode, no command is sent and `SR_STATUS_ERROR` is returned; to enter data mode, `SR_SwitchMode` must be used.

#### 4.5.2.1.1 Prototype

The prototype of `SR_SendCommand` is:

```
SR_STATUS_TYPE_E SR_SendCommand(  UINT8 SRbuffer[256],
                                  UINT32 SRtimeout)
```

#### 4.5.2.1.2 Parameters

The input parameters are:

`< SRbuffer >` Is the buffer which contains the AT command to send and where the received response is stored

`< SRtimeout >` Is the timeout for the response in seconds. If it is 0 the function waits until a response is received. If it is bigger than 0 the function waits for a response up to `SRtimeOut` seconds

#### 4.5.2.1.3 Return Values

The function returns `SR_STATUS_SUCCESS` if the command is sent and a response is received, otherwise it returns `SR_STATUS_ERROR` if an error occurred or `SR_STATUS_TIMEOUT` if the timeout expired.

#### 4.5.2.1.4 Example

```
void SR_SendCommand_Example(void)
{
UINT8 buf[256];
SR_STATUS_TYPE_E eReturnCode = SR_STATUS_ERROR;

strcpy((char *)buf, "ATS192?\r");
if((eReturnCode = SR_SendCommand(buf, 10)) == SR_STATUS_SUCCESS)
{
    printf("Received response: %s", (char *)buf);
}
else
{
    /* No response has been received */
}

return;
}
```

#### 4.5.2.2 SR_SwitchMode

This function allows allows switching from command mode to data mode and vice versa; when command mode is entered, all received M-Bus frames are discarded and every call to `SR_SendData` or `SR_ReceiveData` will return `SR_STATUS_ERROR`; when data mode is entered, every call to `SR_SendCommand` will return `SR_STATUS_ERROR`.

##### 4.5.2.2.1 Prototype

The prototype of `SR_SwitchMode` is:

```
        SR_STATUS_TYPE_E SR_SwitchMode(   SR_MODE_E SRmode)
```

##### 4.5.2.2.2 Parameters

The input parameter is:

| | |
|---|---|
| `< SRmode >` | It specifies which operating mode the module must be put in |

##### 4.5.2.2.3 Return Values

The function returns `SR_STATUS_SUCCESS` if swiching to the requested mode succeeds, otherwise it returns `SR_STATUS_ERROR` if an error occurred.

##### 4.5.2.2.4 Example

```
void SR_SwitchMode_Example(void)
{
SR_MODE_E eMode = COMMAND_MODE;
SR_MODE_E eReturnCode = SR_STATUS_ERROR;

if((eReturnCode = SR_SwitchMode(eMode)) == SR_STATUS_SUCCESS)
{
    /* Switching to command mode succeeded */
}
else
{
    /* Switching to command mode failed */
}

return;
}
```

### 4.5.3 Configuration file

SRtech.conf contains the parameters needed by the SR-Library to communicate with the M-Bus module. It is formatted as follows:

```
…
# It is a comment
[GROUP_TAG_1]
# It is a comment
KEY_TAG_1 = VALUE_1
# It is a comment
KEY_TAG_2 = VALUE_2
…
…
# It is a comment
[GROUP_TAG_2]
# It is a comment
KEY_TAG_1 = VALUE_1
# It is a comment
KEY_TAG_2 = VALUE_2
…
…
```

Where:

- `[GROUP_TAG_1]` is the identifier of a group of parameters that follow this tag.
- `KEY_TAG_1` is the identifier of a parameter
- `VALUE_1` is the value of the parameter

Comments are indicated with the `#` character.

The following table shows the parameters required by the short range library for M-Bus.

| GROUP_TAG | KEY_TAG | M/O/U | Description |
|---|---|---|---|
| [UART] | `SerialPort` | M | Serial port connected to short range module |

**Table 4.26**

***Important:*** If one of these values are not set or set with a wrong value the `SR_Init` function returns an error.
***Pay attention***: Do not insert space or other characters after the value; it can cause an error in the `SR_Init` function.

### 4.5.4 M-Bus Frame format and serial communication

Wireless M-Bus frames are composed of different blocks.
The first block is formatted as follows:

| | | | | |
|---|---|---|---|---|
| L-Field (1 byte) | C-Field (1 byte) | M-Field (2 bytes) | A-Field (6 bytes) | CRC (2 bytes) |

The second block is formatted as follows:

| | | |
|---|---|---|
| CI-Field (1 byte) | Payload (max 15 bytes) | CRC (2 bytes) |

The third and subsequent blocks are formatted as follows:

| | |
|---|---|
| Payload (max 16 bytes) | CRC (2 bytes) |

Only the first and second blocks are mandatory for a given frame. When M-Bus frames are sent and received with the short range library, the different fields of the `SR_DATA_PACKET_T` structure map to the fields of the M-Bus frame as explained in Table 4.8; for multi-byte values, the least significant byte is transmitted first.

The `SRdata` field of `SR_DATA_PACKET_T` contains the C-Field and CI-Field in the first and second byte, and the payload in the next bytes; only the C-Field and CI-Field are mandatory in an M-Bus frame. The `SRlength` field of `SR_DATA_PACKET_T` indicates the number of bytes contained in `SRdata` and is the sum of the lengths of C-Field, CI-Field and payload; its minimum value in a valid M-Bus frame is 2, corresponding to a frame without payload.

Telit M-Bus modules allow choosing different formats for frames exchanged through the serial port. The short range library uses the format where serial frames contain the same fields as M-Bus frames (except for CRC bytes, which are added by the M-Bus module); that means setting the value 31 to both register 401 and register 402 of the M-Bus module (this is done when calling `SR_Init` or performing a hard reset). For details on the configuration registers of the M-Bus module refer to the Telit Wireless M-Bus user guide. The L-Field of frames to be sent is calculated by the short range library by adding 8 to the `SRlength` value; conversely, for received frames the `SRlength` value is calculated from the L-Field by subtracting 8.

Telit M-Bus modules can use different serial baud rates. Since 19200 is the default value, the short range library uses that speed to communicate with the M-Bus module; this means that if a module is configured at a different baud rate the short range library does not work with it.

Configuration of the M-Bus module resulting in a different format of frames exchanged through the serial port will prevent operation of the short range library. For example, if a low power mode is activated requiring a wake-up character to be sent to the module at the beginning of each frame, the short range library is unable to communicate with the module in data mode: when using low power operation, the wakeup pin of the module must be asserted before calling `SR_SendData`.