NanoService Device Library C 1.0

Table of contents

1 Introduction

This document describes how the NanoService C Device library and its included examples are used.

2 Overview

NSDL-C consists of a full-featured CoAP library that can easily be integrated with any kind of UDP socket interface and a set of example servers. CoAP support in NSDL-C is provided by the libCoap library, which consists of three functional parts:

1. CoAP protocol
    a. Handles confirmable CoAP messages resending and acknowledgement
    b. Checks validity of received CoAP message header part
    c. Resource Observation support
    d. Block Transfer support
2. CoAP message building
    a. Builds Packet data from User's given CoAP message structure
3. CoAP message parsing
    a. Parses CoAP message structure from received Packet data

There are following kinds of possibilities to use libCoap:

1. The complete protocol library is used: CoAP protocol, CoAP message builder and CoAP message parser are included in libCoap
2. Just header parsing and building parts are used: CoAP message builder and CoAP message parser are included in libCoap

For reducing code size, the following features can be left from compiling with compiling switches:

- Message resending (Define name: SN_COAP_RESENDING_MAX_MSGS_COUNT)

- Message duplication detection (Define name: SN_COAP_DUPLICATION_MAX_MSGS_COUNT)
- Message block transfer (Define name: SN_COAP_BLOCKWISE_MAX_PAYLOAD_SIZE)

For inclusion in a project, there are two Header files that define the needed functions:

- sn_coap_protocol.h
- sn_coap_header.h

From those Header files these functions are available:

- From sn_coap_protocol.h:

  1. sn_coap_protocol_init()
  2. sn_coap_build()
  3. sn_coap_parse()
  4. sn_coap_exec()
  5. sn_coap_packet_debug()
  6. sn_coap_build_response()
  7. sn_coap_register()
  8. sn_coap_register_update()
  9. sn_coap_deregister()

- From sn_coap_header.h:

  10. sn_coap_builder_and_parser_init()
  11. sn_coap_builder()
  12. sn_coap_parser()
  13. sn_coap_builder_calc_needed_packet_data_size()
  14. sn_coap_builder_release_allocated_send_msg_mem()
  15. sn_coap_parser_release_allocated_coap_msg_mem()

3 Ports

libCoap is written purely in C, and is portable across different microcontroller platforms, as well as PC platforms. The following ports are currently included with the library.

3.1 x86_gcc

/libCoap/x86_gcc

This port is for x86 based PCs using the standard gcc compiler. The examples provided with the library also make use of this port. See the Makefile in the example server implementations for an example of how to include libCoap into a gcc based project.

3.2 Custom ports

If your license include source code of libCoap, it is also possible to port the library to your own microcontroller architecture and compiler toolchain. To do this, follow the following general steps. Specifics will of course depend on your toolchain.

1. Create a new sub-directory /libCoap/Arch_Compiler in the project
2. If your toolchain uses make, then copy the x86_gcc Makefile and make the needed modifications for you compiler and linker
3. Once the library is compiling, thorough testing and verification will be needed. If port-specific code changes are needed, using suitable #ifdef #endif defines is recommended so as not to break other ports.

For some projects, it maybe be more straightforward to include the libCoap headers and code into your project directly instead of building a static library first.

4 Examples

Several examples of CoAP servers for use with NanoService Platform are included in the package for x86 Linux with support for IPv6.

4.1 Connected Home Example

This example is a simple command-line Linux CoAP server that registers with NanoService Platform, and de-registers when ctrl-c is pressed. The server emulates a power measurement node with a relay for use in the Connected Home Reference App.

The code consists of a command-line parameter parser in main.c and the CoAP server in connected-home.c. The example makes use of the libCoap parsing and building functions along with NSP registration and de-registration functions.

To make the example simply:

```
cd nsdl-c-xxx
cd connected-home
make
```

To use the example together with the Connected Home demo setup:

```
./runConnectedHomeDemo.sh
cd nsdl-c-xxx
cd connected-home
./connected-home
```

4.2 Lighting Example

This example is a simple command-line Linux CoAP server that registers with NanoService Platform, and de-registers when ctrl-c is pressed. The server emulates a street lighting node with a relay for use in the Lighting Reference App.

The code consists of a command-line parameter parser in main.c and the CoAP server in lighting.c. The example makes use of the libCoap parsing and building functions along with NSP registration and de-registration functions.

To make the example simply:

```
cd nsdl-c-xxx
cd lighting
make
```

To use the example together with the Connected Home demo setup:

```
./runLightingDemo.sh
cd nsdl-c-xxx
cd lighting
./lighting
```

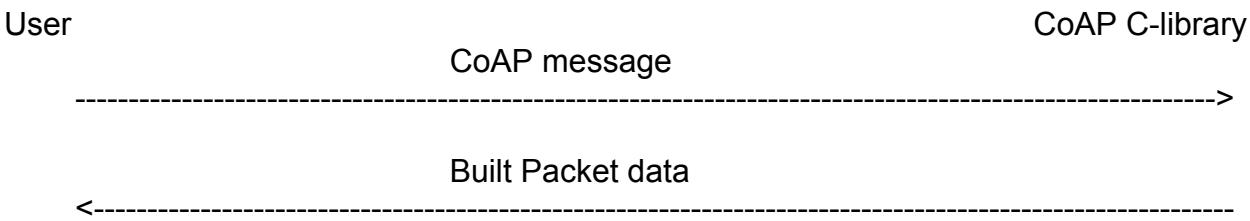
4.3 ETSI Plugtest Server

This server is structured in the same way as the other examples, and implements the ETSI Plugtest specification for CoAP interop and validation testing. This test specification defines resources and behavior that a CoAP Server under test must exhibit. This server has been used to verify the correctness of libCoap according to the currently available ETSI CoAP Plugtest tests.

5 Memory allocation and release

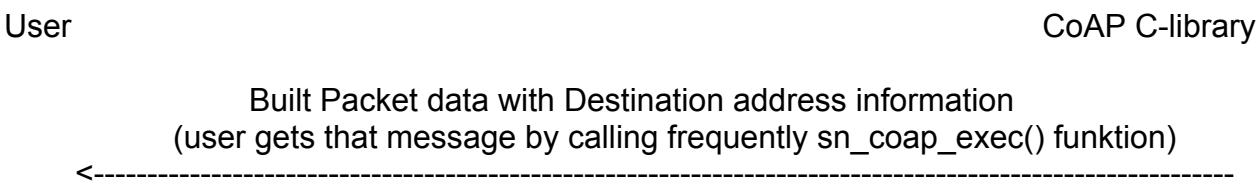
5.1 CoAP message building for sending

5.1.1 User builds message



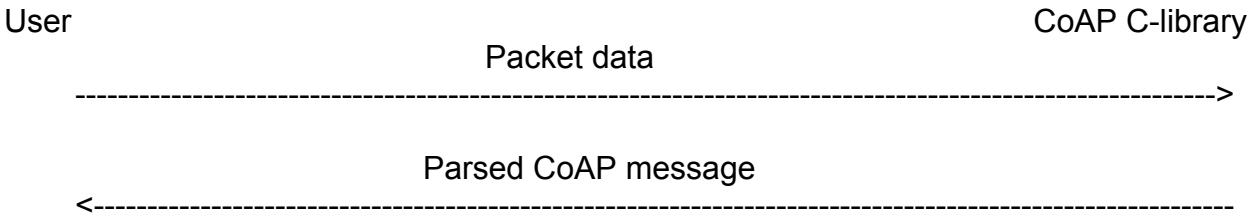
- User allocates memory for built Packet data and also takes care of releasing that memory. Needed memory count for Packet data is solved by using sn_coap_builder_calc_needed_packet_data_size() function.

5.1.2 libCoap builds message



- User also takes care of releasing memory of Packet data which is built independently by CoAP C-library. User can use for releasing sn_coap_builder_release_allocated_send_msg_mem() function.

5.2 Received Packet data parsing



- libCoap allocates memory for CoAP message structure (= sn_coap_hdr_s) where Packet data is parsed and returned to User.
    - User responsibility is to release allocated memory when does not need it any more. User can use for releasing sn_coap_parser_release_allocated_coap_msg_mem() function.

- libCoap does not allocate new memory for Payload part because CoAP message structure's Payload pointer is just pointed to Payload part in Packet data. Exception for that is Blockwise message. For that libCoap allocates memory by itself. That happens after all Blockwise messages are arrived and libCoap can gather whole Payload from received Blockwise messages.

6 Functions

This chapter describes functions which are visible to libCoap users. The sn_coap_init() and sn_coap_protocol_init() (if protocol functions are used) must be called before using any other libCoap function.

6.1 sn_coap_protocol_init()

```
void sn_coap_protocol_init(void*
(*used_malloc_func_ptr)(uint16_t),
                           void (*used_free_func_ptr)(void*))
```

Initializes CoAP Protocol part

Parameters	
used_malloc_func_ptr	Function pointer to used malloc() function. If set to NULL, CoAP Protocol part uses standard C-library malloc() function.
used_free_func_ptr	Function pointer to used free() function. If set to NULL, CoAP Protocol part uses standard C-library free() function.

Example of using:

```
sn_coap_protocol_init((void* (*)(uint16_t))&user_malloc_func,
                      &user_free_func);
```

6.2 sn_coap_build()

```
int16_t sn_coap_build(sn_nsdl_addr_s *dst_addr_ptr,
                      uint8_t *dst_packet_data_ptr,
                      sn_coap_hdr_s *src_coap_msg_ptr)
```

Builds Packet data from given CoAP header structure to be sent

Parameters	
dst_addr_ptr	Pointer to destination address where CoAP message will be sent (CoAP builder needs that information for message resending purposes)
dst_packet_data_ptr	Pointer to destination of built Packet data
src_coap_msg_ptr	Pointer to source of built Packet data
Return values	If there is not enough memory (or User given limit exceeded) for storing resending messages, situation is ignored.
>=0	Byte count of built Packet data. Note: If message is blockwised, all payload is not sent at the same time
-1	Failure in CoAP header structure
-2	Failure in given pointer (= NULL)
-3	Failure in Reset message

Example of using:

```
byte_count_built = sn_coap_build(&dst_addr,
                                 dst_packet_data_ptr,
                                 src_coap_msg_ptr);
```

6.3 sn_coap_parse()

sn_coap_hdr_s *sn_coap_parse(sn_nsdl_addr_s *src_addr_ptr, uint16_t packet_data_len, uint8_t *packet_data_ptr)	
Parses received CoAP message from given Packet data	
Parameters	
src_addr_ptr	Pointer to source address from where CoAP message was be received (CoAP parser needs that information for sending Reset message)
packet_data_len	Length of given Packet data to be parsed to CoAP message
packet_data_ptr	Pointer to source of Packet data to be parsed to CoAP message
Return values	
>0	Pointer to parsed CoAP message. This structure includes also coap_status field for following special cases: -CoAP will send Reset message to invalid message sender -CoAP will send Acknowledgement message to duplicated message sender - User will get whole message after all message blocks received. User must release messages with this status. - Acknowledgement for sent Blockwise message received -Blockwise message received but not supported by compiling switch
NULL	In following failure cases NULL is returned: -Given NULL pointer -Failure in parsed Header -Out of memory (malloc() returns NULL)

Example of using:

```
parsed_message_ptr = sn_coap_parse(&src_addr,
                                   sizeof(parsed_data_tbl),
                                   parsed_data_tbl);
```

6.4 sn_coap_exec()

User must call this function regularly because this is only way how CoAP protocol can send CoAP message. Frequency for calling depends on message traffic amount.

sn_nsdl_transmit_s *sn_coap_exec(uint32_t current_time)	
Sends one CoAP message if there is any to be sent. This function can be called e.g. once in a second but also more frequently.	
Parameters	
current_time	System time in seconds. This time is used for message resending timing.
Return values	
	Pointer to message to be sent. NULL is returned in following cases: -There is nothing to send -Out of memory (malloc() returns NULL)

Example of using:

```
coap_msg_to_be_sent_ptr = sn_coap_exec(current_system_time);
```

6.5 sn_coap_builder_and_parser_init()

void sn_coap_builder_and_parser_init(void* (*used_malloc_func_ptr)(uint16_t), void (*used_free_func_ptr)(void*))	
Initializes CoAP Builder and Parser parts	
Parameters	

used_malloc_func_ptr	Function pointer to used malloc() function. If set to NULL, CoAP Parser part uses standard C-library malloc() function.
used_free_func_ptr	Function pointer to used free() function. If set to NULL, CoAP Parser part uses standard C-library free() function.

Example of using:

```
sn_coap_builder_and_parser_init((void*(*)(uint16_t))&user_malloc_func,
                                user_free_func);
```

6.6 sn_coap_builder ()

```
int16_t sn_coap_builder(uint8_t *dst_packet_data_ptr,
                        sn_coap_hdr_s *src_coap_msg_ptr
)
```

Builds Packet data from given CoAP header structure

Parameters	
dst_packet_data_ptr	Destination for built Packet data
src_coap_msg_ptr	Source for building Packet data
Return values	
	Byte count of built Packet data. In failure cases: -1 = Failure in given CoAP header structure -2 = Failure in given pointer (= NULL)

Example of using:

```
byte_count_built = sn_coap_builder(dst_packet_data_ptr,
                                   src_coap_msg_ptr
                                  );
```

6.7 sn_coap_parser ()

```
sn_coap_hdr_s *sn_coap_parser(uint16_t packet_data_len,
                              uint8_t *packet_data_ptr,
                              coap_version_e *coap_version_ptr
)
```

Parses CoAP message from given Packet data

Parameters	
packet_data_len	Length of given Packet data to be parsed to CoAP message
packet_data_ptr	Source for Packet data to be parsed to CoAP message
coap_version_ptr	Destination for parsed CoAP specification version
Return values	
	Pointer to parsed CoAP message. In following failure cases NULL is returned: -Failure in given pointer (= NULL) -Failure in memory allocation (malloc() returns NULL)

Example of using:

```
returned_dst_coap_msg_ptr = sn_coap_parser(packet_data_len,
                                           packet_data_ptr
                                          );
```

6.8 sn_coap_builder_calc_needed_packet_data_size()

uint16_t sn_coap_builder_calc_needed_packet_data_size(sn_coap_hdr_s *src_coap_msg_ptr)	
Calculates needed Packet data memory size for given CoAP message	
Parameters	
src_coap_msg_ptr	Pointer to data which needed length is calculated
Return values	
	Return value is count of needed memory as bytes for build and returned CoAP message Packet data

Example of using:

- ```
  dst_byte_count_to_be_built =

  sn_coap_builder_calc_needed_packet_data_size(src_coap_msg_ptr
  );
  ```

6.9 sn_coap_builder_release_allocated_send_msg_mem()

void sn_coap_builder_release_allocated_send_msg_mem(sn_coap_hdr_s *freed_send_msg_ptr)	
Frees memory of given Sending message.	
Parameters	
freed_send_msg_ptr	Pointer to released Sending message

Example of using:

- ```
  sn_coap_parser_release_allocated_coap_msg_mem(freed_coap_msg_
  ptr);
  ```

6.10 sn_coap_parser_release_allocated_coap_msg_mem()

void sn_coap_parser_release_allocated_coap_msg_mem(sn_coap_hdr_s *freed_coap_msg_ptr)	
Frees memory of given CoAP message. **Note!!!** Does not release Payload part	
Parameters	
freed_coap_msg_ptr	Pointer to released CoAP message

Example of using:

- ```
  sn_coap_parser_release_allocated_coap_msg_mem(freed_coap_msg_
  ptr);
  ```

6.11 sn_coap_packet_debug()

void sn_coap_packet_debug(sn_coap_hdr_s *coap_packet_ptr)	
Provides packet debugging, using printf to display all parts of the message	
Parameters	
coap_packet_ptr	Pointer to the CoAP message structure to debug

Example of using:
  ```

- **sn_coap_packet_debug**(my_coap_packet_ptr);

6.12 sn_coap_register()

int8_t sn_coap_register(sn_coap_hdr_s *coap_hdr_ptr, const char *ep, const char *ep_type, const char *links)	
Builds a CoAP resource directory registration message from registration parameter strings	
Parameters	
coap_hdr_ptr	Pointer to the CoAP message in which the built message should be placed
ep	String with the endpoint name (ep= paramter)
ep_type	String with the node type (rt= parameter)
links	String with the link payload to be included in the registration
Return values	
	Return value 0 given on success. In failure cases: -1 = Failure

Example of using:

- ret = sn_coap_register(coap_hdr_ptr, ep, rt, links);

6.13 sn_coap_register_update()

int8_t sn_coap_register_update(sn_coap_hdr_s *coap_hdr_ptr, char *location)	
Builds a CoAP resource directory registration update message and sends it to the location returned when first registering	
Parameters	
coap_hdr_ptr	Pointer to the CoAP message in which the built message should be placed
location	Path returned when first registering
Return values	
	Return value 0 given on success. In failure cases: -1 = Failure

Example of using:

- ret = sn_coap_register_update(coap_hdr_ptr, location);

6.14 sn_coap_deregister ()

int8_t sn_coap_deregister (sn_coap_hdr_s *coap_hdr_ptr, char *location)	
Builds a CoAP resource directory deregistration message and sends it to the location returned when first registering	
Parameters	
coap_hdr_ptr	Pointer to the CoAP message in which the built message should be placed
location	Path returned when first registering
Return values	
	Return value 0 given on success. In failure cases: -1 = Failure

Example of using:

- ret = sn_coap_deregister(coap_hdr_ptr, location);

7 Data Structures

This chapter describes data structure types and fields ,which are visible for libCoap user.

7.1 Basic data types

Name	Definition
uint8_t	Unsigned 8 bit integer
int8_t	Signed 8 bit integer
uint16_t	Unsigned 16 bit integer
int16_t	Signed 16 bit integer
uint32_t	Unsigned 32 bit integer
int32_t	Signed 32 bit integer

7.2 sn_coap_hdr_s

```
typedef struct sn_coap_hdr_
{
    sn_coap_status_e         coap_status;

    sn_coap_msg_type_e       msg_type;
    sn_coap_msg_code_e       msg_code;
    uint16_t                 msg_id;

    uint16_t                 uri_path_len;
    uint8_t                 *uri_path_ptr;
    uint8_t                  token_len;
    uint8_t                 *token_ptr;
    uint8_t                  content_type_len;
    uint8_t                 *content_type_ptr;
    sn_coap_options_list_s  *options_list_ptr;

    uint8_t                  payload_len;
    uint8_t                 *payload_ptr;
} sn_coap_hdr_s;
```

Field name	Description	Notes
coap_status	CoAP status	Used for telling to User special cases when parsing message
msg_type	CoAP Message type	Possible Message types: Confirmable or non-confirmable
msg_code	CoAP Message code	Possible Message codes: Empty, request or response
uri_path_len	Uri-Path option length	Must be set to zero if not used
uri_path_ptr	Uri-Path option data	Must be set to NULL if not used
token_len	Token option length	Must be set to zero if not used
token_ptr	Token option data	Must be set to NULL if not used
content_type_len	Content type option length	Must be set to zero if not used
content_type_ptr	Content type option data	Must be set to NULL if not used
options_list_ptr	List of other options	Here are not so often used options
payload_len	Payload length	Must be set to zero if not used
payload_ptr	Payload data	Must be set to NULL if not used

7.3 sn_coap_options_list_s

```
typedef struct sn_coap_options_list_
{
    uint8_t     max_age_len;
    uint8_t    *max_age_ptr;
    uint16_t    proxy_uri_len;
    uint8_t    *proxy_uri_ptr;
    uint8_t     etag_len;
    uint8_t    *etag_ptr;
    uint16_t    uri_host_len;
    uint8_t    *uri_host_ptr;
    uint16_t    location_path_len;
    uint8_t    *location_path_ptr;
    uint8_t     uri_port_len;
    uint8_t    *uri_port_ptr;
    uint16_t    location_query_len;
    uint8_t    *location_query_ptr;
    uint8_t     observe_len;
    uint8_t    *observe_ptr;
    uint8_t     accept_len;
    uint8_t     *accept_ptr;
    uint16_t    uri_query_len;
    uint8_t    *uri_query_ptr;
    uint16_t    block1_len;
    uint8_t    *block1_ptr;
    uint16_t    block2_len;
    uint8_t    *block2_ptr;
} sn_coap_options_list_s;
```

Field name	Description	Notes
max_age_len	Max-Age option length	
max_age_ptr	Max-Age option data	Must be set to NULL if not used
proxy_uri_len	Proxy-Uri option length	
proxy_uri_ptr	Proxy-Uri option data	Must be set to NULL if not used
etag_len	Etag option length	
etag_ptr	Etag option data	Must be set to NULL if not used
uri_host_len	Uri-Host option length	
uri_host_ptr	Uri-Host option data	Must be set to NULL if not used
location_path_len	Location-Path option length	
location_path_ptr	Location-Path option data	Must be set to zero if not used
uri_port_len	Uri-Port option length	
uri_port_ptr	Uri-Port option data	Must be set to NULL if not used
location_query_len	Location-Query option length	
location_query_ptr	Location-Query option data	Must be set to NULL if not used
uri_query_len	Uri-Query option length	
uri_query_ptr	Uri-Query option data	Must be set to NULL if not used
block1_len	Block1 option length	
block1_ptr	Block1 option data	Must be set to NULL if not used
Block2_len	Block2 option length	
Block2_ptr	Block2 option data	Must be set to NULL if not used