

**NanoService Device Library C 1.2**

**Table of contents**

1 Introduction 4

2 Overview 4

3 Ports 6

3.1 x86\_gcc 6

3.2 Custom ports 6

4 Examples 6

4.1 Connected Home Example 6

4.2 Lighting Example 7

4.3 ETSI Plugtest Server 7

5 Memory allocation and release 8

5.1 CoAP message building for sending 8

5.1.1 User builds message 8

5.1.2 libCoap builds message 8

5.2 Received Packet data parsing 8

6 Functions 9

6.1 sn\_coap\_protocol\_init() 9

6.2 sn\_coap\_build() 9

6.3 sn\_coap\_parse() 9

6.4 sn\_coap\_exec() 10

6.5 sn\_coap\_builder\_and\_parser\_init() 10

6.6 sn\_coap\_builder () 11

6.7 sn\_coap\_parser () 11

6.8 sn\_coap\_builder\_calc\_needed\_packet\_data\_size() 12

6.9 sn\_coap\_builder\_release\_allocated\_send\_msg\_mem() 12

6.10 sn\_coap\_parser\_release\_allocated\_coap\_msg\_mem() 12

6.11 sn\_coap\_packet\_debug() 12

6.12 sn\_coap\_register() 13

6.13 sn\_coap\_register\_update() 13

6.14 sn\_coap\_deregister () 13

7 Data Structures 14

7.1 Basic data types 14

7.2 sn\_coap\_hdr\_s 14

7.3 sn\_coap\_options\_list\_s 15

**Changelog**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
| 0.1 | 2011-08-24 | Pekka Karjaluoto | Initial draft |
| 0.2 | 2011-09-02 | Pekka Karjaluoto | Added memory allocation and releasing part (Chapter ). Split sn\_coap\_init() function to sn\_coap\_protocol\_init() and sn\_coap\_builder\_and\_parser\_init() functions. |
| 0.3 | 2012-05-07 | Zach Shelby | Updates and improvements to libCoap |
| 1.0 | 2012-06-28 | Zach Shelby | Further documentation improvements, preparing for release |
| 1.2 | 2013-01-15 | Zach Shelby | Added NSDL C and EDTLS documentation |
|  |  |  |  |
|  |  |  |  |

# Introduction

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

# 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
     1. Handles confirmable CoAP messages resending and acknowledgement
     2. Checks validity of received CoAP message header part
     3. Resource Observation support
     4. Block Transfer support
  2. CoAP message building
     1. Builds Packet data from User’s given CoAP message structure
  3. CoAP message parsing
     1. 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:

1. sn\_coap\_builder\_and\_parser\_init()
2. sn\_coap\_builder()
3. sn\_coap\_parser()
4. sn\_coap\_builder\_calc\_needed\_packet\_data\_size()
5. sn\_coap\_builder\_release\_allocated\_send\_msg\_mem()
6. sn\_coap\_parser\_release\_allocated\_coap\_msg\_mem()

# 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.

## 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.

## 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.

# Examples

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

## 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

## 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

## 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.

# Memory allocation and release

## CoAP message building for sending

### User builds message

User CoAP C-library

CoAP message

----------------------------------------------------------------------------------------------------------->

Built Packet data

<-----------------------------------------------------------------------------------------------------------

* 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.

### libCoap builds message

User CoAP C-library

Built Packet data with Destination address information

(user gets that message by calling frequently sn\_coap\_exec() funktion)

<-----------------------------------------------------------------------------------------------------------

* 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.

## Received Packet data parsing

User CoAP C-library

Packet data

----------------------------------------------------------------------------------------------------------->

Parsed CoAP message

<-----------------------------------------------------------------------------------------------------------

* 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.

# 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.

## 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);

## 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);

## 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);

## 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);

## 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);

## 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

);

## 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

);

## 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);

## 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);

## 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);

## 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);

## 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);

## 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);

## 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);

# Data Structures

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

## 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 |

## 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 |

## 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 |