Realtek Ameba CoAP Library User Guide

This document provides a guideline to use mbed CoAP C Library APIs in Ameba SDK

Table of Contents

1 CoAP Protocol Introduction

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks. The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation.

CoAP provides a request/response interaction model between application endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types. CoAP is designed to easily interface with HTTP for integration with the Web while meeting specialized requirements such as multicast support, very low overhead, and simplicity for constrained environments.

1.1 CoAP Protocol Stack

CoAP is an application layer protocol build on the top of UDP layer, it contains two sub-layers: CoAP method layer and CoAP transaction layer.

Application Layer	Application	
	CoAP	CoAP Method
		CoAP Transaction
Transport Layer	UDP	

The function of the **CoAP transaction** layer is to control message exchanges over UDP between two endpoints. There are 4 message types: confirmable (CON), non-confirmable (NON), acknowledgement (ACK), and reset (RST). The message transactions are formed between peer to peer and identified by transaction ID. An optional token can be used to differentiate concurrent requests. The 4 message types are explained as below:

- CON: confirmable request, receiving peer must send an ACK or RST to reply the message.
- NON: non-confirmable request, receiving peer does not require to reply
- ACK: acknowledgement, send when a CON message is received, can carry payload.
- RST: rest, indicates a CON message is received but some content is missing to process it.

At the **CoAP method** layer, request and response semantics are carried in a message, and include either a method code or a response code. The message also carries optional information, such as the URI and type of payload content. There are 4 method types: GET, POST, PUT, and DELETE which are explained as below:

- GET: retrieve information from URI
- POST: create a new resource under requested URI
- PUT: update resource identified by URI
- DELETE: delete resource identified by URI

1.2 CoAP Message Format

CoAP messages are encoded in a simple binary format with extensible options. The protocol has a base header size of only 4 bytes, and a total header of 10–20 bytes for a typical request.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Token (if any, TKL bytes) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Version (Ver): 2-bit unsigned integer. Indicates the CoAP version number. Implementations of this specification MUST set this field to 1 (01 binary). Other values are reserved for future versions. Messages with unknown version numbers MUST be silently ignored.

Type (T): 2-bit unsigned integer. Indicates if this message is of type Confirmable (0), Non-confirmable (1), Acknowledgement (2), or Reset (3).

Token Length (TKL): 4-bit unsigned integer. Indicates the length of the variable-length Token field (0-8 bytes). Lengths 9-15 are reserved, MUST NOT be sent, and MUST be processed as a message format error.

Code: 8-bit unsigned integer, split into a 3-bit class (most significant bits) and a 5-bit detail (least significant bits), documented as "c.dd" where "c" is a digit from 0 to 7 for the 3-bit subfield and "dd" are two digits from 00 to 31 for the 5-bit subfield. The class can indicate a request (0), a success response (2), a client error response (4), or a server error response (5). (All other class values are reserved.) As a special case, Code 0.00 indicates an Empty message. In case of a request, the Code field indicates the Request Method; in case of a response, a Response Code.

Message ID: 16-bit unsigned integer in network byte order. Used to detect message duplication and to match messages of type Acknowledgement/Reset to messages of type Confirmable/Non-confirmable.

Token value (Token): which may be 0 to 8 bytes, as given by the Token Length field. The Token value is used to correlate requests and responses.

Options: Header and Token are followed by zero or more Options. An Option can be followed by the end of the message, by another Option, or by the Payload Marker and the payload.

Payload: Following the header, token, and options, if any, comes the optional payload. If present and of non-zero length, it is prefixed by a fixed, one-byte Payload Marker (0xFF), which indicates the end of options and the start of the payload. The payload data extends from after the marker to the end of the UDP datagram, i.e., the Payload Length is calculated from the datagram size. The absence of the Payload Marker denotes a zero-length payload. The presence of a marker followed by a zero-length payload MUST be processed as a message format error.

For detailed information about this protocol, please reference to IETF RFC7252 .

2 mbed CoAP APIs and Ameba Wrappers

CoAP messages are built/parsed by using mbed CoAP APIs and sent/received by using Ameba wrapper functions. A quick briefing is given as below:

mbed CoAP APIs	
sn_coap_header.h	
sn_coap_parser()	Use to parse an incoming message buffer to a CoAP header structure.
sn_coap_parser_release_allocated_coap_msg_mem()	This function releases any memory allocated by a CoAP message structure.
sn_coap_builder()	Use to build an outgoing message buffer from a CoAP header structure.
sn_coap_builder_calc_needed_packet_data_size()	Use to calculate the needed message buffer size from a CoAP message structure.
sn_coap_builder_2()	Builds an outgoing message buffer from a CoAP header structure.
sn_coap_builder_calc_needed_packet_data_size_2()	Calculates needed Packet data memory size for given CoAP message.

sn_coap_build_response()	Use to automate the building of a response to an incoming request.
sn_coap_parser_init_message()	Initialize a message structure to empty.
sn_coap_parser_alloc_message()	Allocate an empty message structure.
sn_coap_parser_alloc_options()	Allocates and initializes options list structure.
sn_coap_protocol.h	
sn_coap_protocol_init()	This function sets the memory allocation and deallocation functions the library will use, and must be called first.
sn_coap_protocol_destroy()	Frees all allocated memory in CoAP protocol part.
sn_coap_protocol_build()	Use to build an outgoing message buffer from a CoAP header structure.
sn_coap_protocol_parse()	Use to parse an incoming message buffer to a CoAP header structure.
sn_coap_protocol_exec()	Called periodically to allow the protocol to update retransmission timers and destroy unneeded data.
sn_coap_protocol_set_block_size()	If block transfer is enabled, this function changes the block size.
sn_coap_protocol_set_duplicate_buffer_size()	If duplicate message detection is enabled, this function changes buffer size.
sn_coap_protocol_set_retransmission_parameters()	If re-transmissions are enabled, this function changes resending count and interval.
sn_coap_protocol_set_retransmission_buffer()	If re-transmissions are enabled, this function changes message retransmission queue size
sn_coap_protocol_clear_retransmission_buffer()	If re-transmissions are enabled, this function removes all messages from the retransmission queue.
sn_coap_protocol_block_remove()	Remove saved block data. Can be used to remove the data from RAM to enable storing it to other place.
sn_coap_protocol_delete_retransmission()	If re-transmissions are enabled, this function removes message from retransmission buffer.
Ameba Wrappers	
sn_coap_ameba_port.h	
tr_debug()	Print CoAP debug message
randLIB_get_16bit()	This function returns an unsigned short, which is called in sn_coap_protocol_init() for random message ID

coap_malloc()	Ameba malloc() function wrapper
coap_free()	Ameba free() function wrapper
coap_calloc()	Ameba calloc() function wrapper
coap_sock_open()	Ameba socket() function wrapper, creates a Datagrams type (SOCK_DGRAM) socket.
coap_sock_close()	Ameba close() function wrapper, closes a socket file descriptor.
coap_protocol_init()	Ameba sn_coap_protocol_init() function wrapper, initializes CoAP Protocol part.
coap_send()	Send the constructed CoAP message to designated host address on the specific port number.
coap_recv()	Receive the constructed CoAP message from host.
coap_print_hdr()	Print CoAP message header (for debug use)

For detailed description, please reference to Ameba SDK Documentation under “Modules”-> “Network” -> “COAP” session.

3 CoAP Example

An example of using CoAP C library and correspond APIs is provided in example_coap.c

Example description:

This example demonstrates how to use mbed-CoAP C library to build and parse a CoAP message.

In the example, a confirmable GET request is send to test server "coap.me" to retrieve the resource under path "/hello". The expected return is an ACK message with payload "world".

Note:

Company Firewall may block CoAP message. You can use copper (https://addons.mozilla.org/en-US/firefox/addon/copper-270430/) to test the server's reachability.

3.1 Example Setup

1) Add CoAP library and example to SDK:

/component/common/network/coap

```
.
|-- include
|    |-- ns_list.h
|    |-- ns_types.h
|    |-- sn_coap_ameba_port.h
|    |-- sn_coap_header.h
|    |-- sn_coap_header_internal.h
|    |-- sn_coap_protocol.h
|    |-- sn_coap_protocol_internal.h
|    `-- sn_config.h
|-- sn_coap_ameba_port.c
|-- sn_coap_builder.c
|-- sn_coap_header_check.c
|-- sn_coap_parser.c
`-- sn_coap_protocol.c
```

/component/common/example/coap

```
.
|-- example_coap.c
|-- example_coap.h
`-- readme.txt
```

2) Add/Enable CONFIG_EXAMPLE_COAP in platform_opts.h

```
/* for CoAP example*/
#define CONFIG_EXAMPLE_COAP                    1
```

3) Add example_coap() to example_entry.c

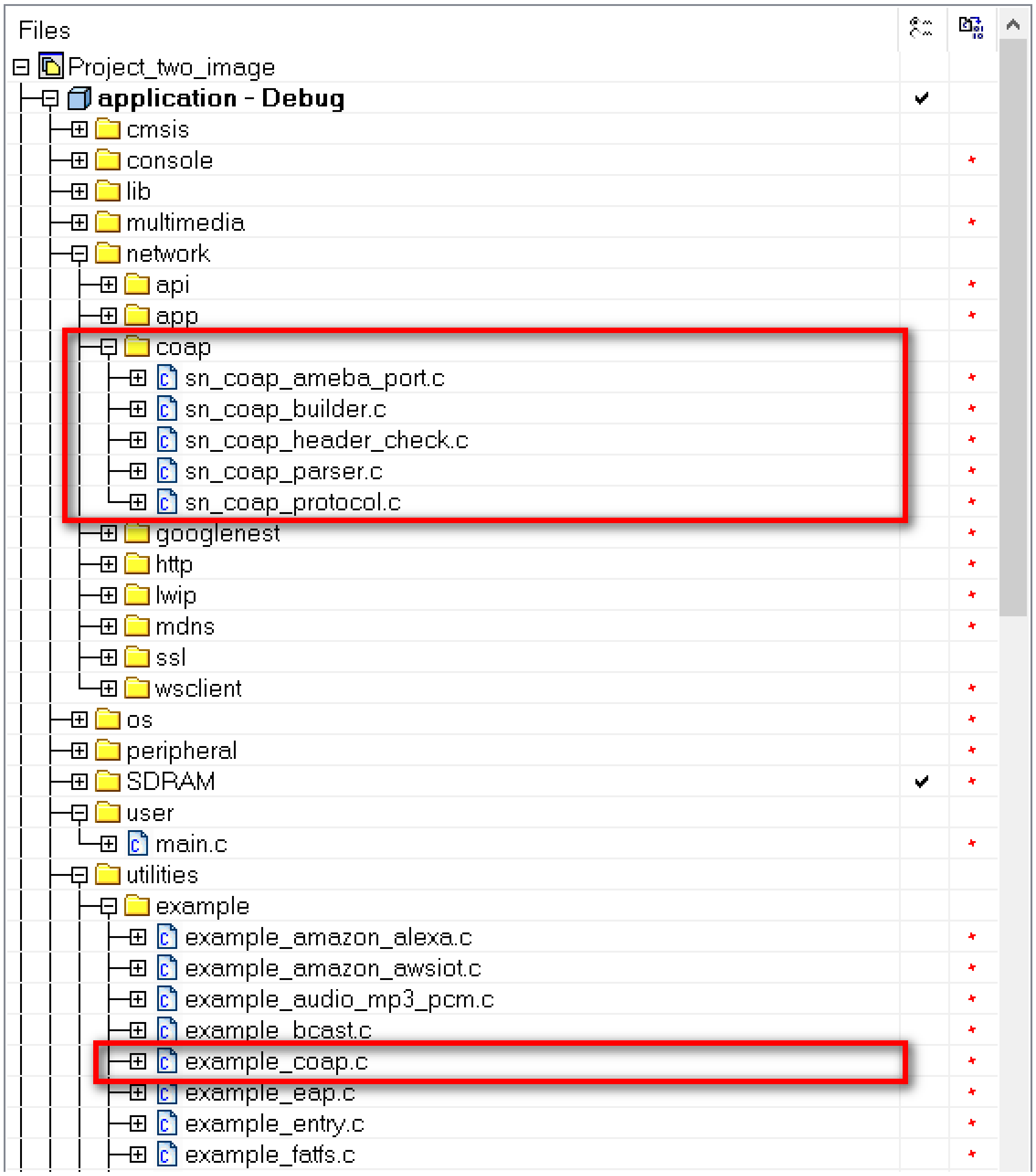
```
#if CONFIG_EXAMPLE_COAP
#include <coap/example_coap.h>
#endif

void example_entry(void)
{

#if CONFIG_EXAMPLE_COAP
    example_coap();
#endif

}
```

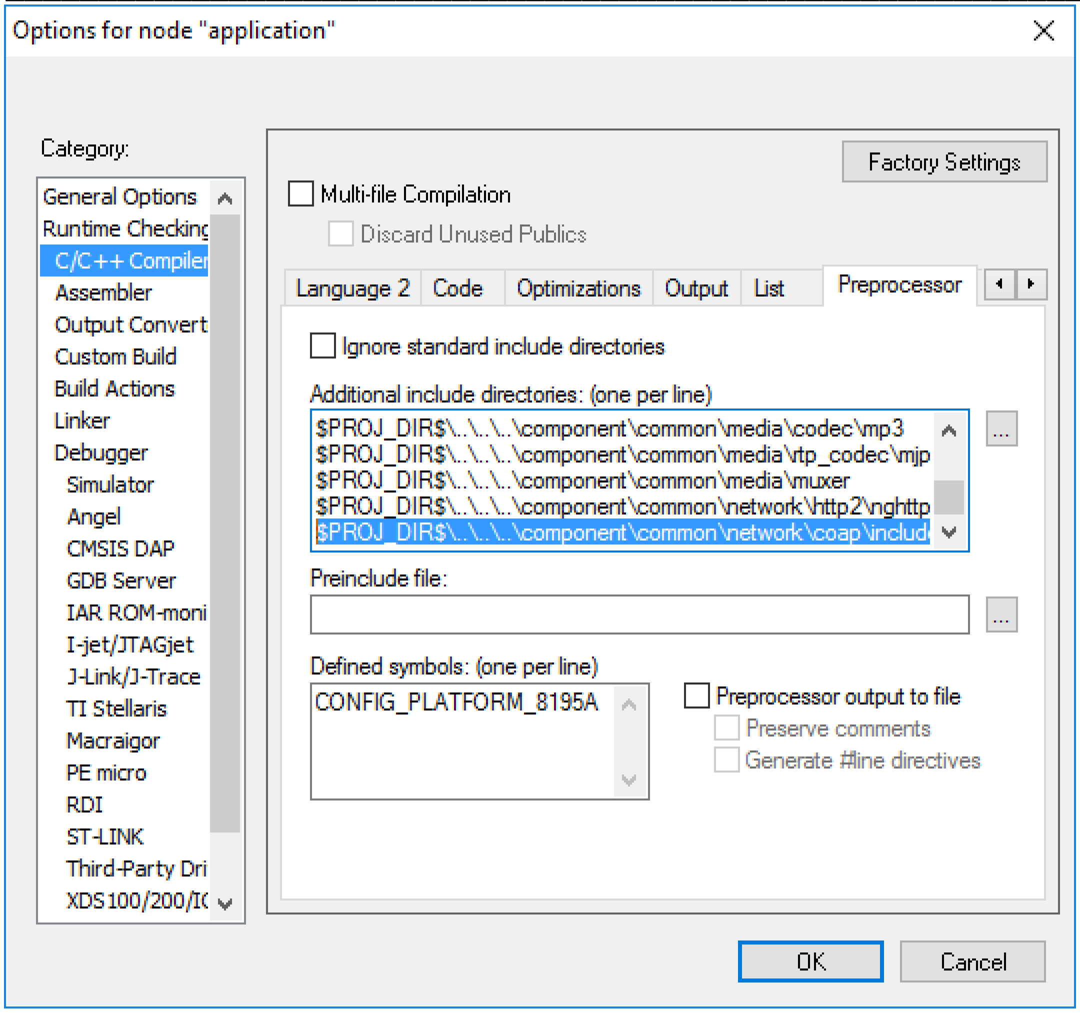
4) Add CoAP related files to IAR project



```
Files
⊟ Project_two_image
├⊟ application - Debug                          ✓
│ ├⊞ cmsis
│ ├⊞ console
│ ├⊞ lib
│ ├⊞ multimedia
│ ├⊟ network
│ │ ├⊞ api
│ │ ├⊞ app
│ │ ├⊟ coap
│ │ │ ├⊞ sn_coap_ameba_port.c
│ │ │ ├⊞ sn_coap_builder.c
│ │ │ ├⊞ sn_coap_header_check.c
│ │ │ ├⊞ sn_coap_parser.c
│ │ │ └⊞ sn_coap_protocol.c
│ │ ├⊞ googlenest
│ │ ├⊞ http
│ │ ├⊞ lwip
│ │ ├⊞ mdns
│ │ ├⊞ ssl
│ │ └⊞ wsclient
│ ├⊞ os
│ ├⊞ peripheral
│ ├⊞ SDRAM                                      ✓
│ ├⊟ user
│ │ └⊞ main.c
│ ├⊟ utilities
│ │ ├⊟ example
│ │ │ ├⊞ example_amazon_alexa.c
│ │ │ ├⊞ example_amazon_awsiot.c
│ │ │ ├⊞ example_audio_mp3_pcm.c
│ │ │ ├⊞ example_bcast.c
│ │ │ ├⊞ example_coap.c
│ │ │ ├⊞ example_eap.c
│ │ │ ├⊞ example_entry.c
│ │ │ ├⊞ example_fatfs.c
```

5) Add include directories to project

$PROJ_DIR$\..\..\..\component\common\network\coap\include



Additional include directories: (one per line)

```
$PROJ_DIR$\..\..\..\component\common\media\codec\mp3
$PROJ_DIR$\..\..\..\component\common\media\rtp_codec\mjp
$PROJ_DIR$\..\..\..\component\common\media\muxer
$PROJ_DIR$\..\..\..\component\common\network\http2\nghttp
$PROJ_DIR$\..\..\..\component\common\network\coap\includ
```

Defined symbols: (one per line)

```
CONFIG_PLATFORM_8195A
```

3.2 Example Execution

Build project and download image to Ameba. A sample of example execution result is logged as below:

```
Wait for WIFI connection ...

Wait for WIFI connection ...

Wait for WIFI connection ...



Interface 0 IP address : 192.168.1.11

Received 11 bytes from '134.102.218.18:13078'
```

```
.token_len:              0

.token_ptr:              (null)

.coap_status:            0 COAP_STATUS_OK

.msg_code:               2.05 COAP_MSG_CODE_RESPONSE_CONTENT

.msg_type:               20 COAP_MSG_TYPE_ACKNOWLEDGEMENT

.content_format:         0 COAP_CT_TEXT_PLAIN

.msg_id:                 7

.uri_path_len:           0

.uri_path_ptr:           (null)

.payload_len:            5

.payload_ptr:            world
```