Hochschule Offenburg
University of Applied Sciences
Elektrotechnik und
Informationstechnik

emb6 Documentation

Documentation of the emb6 Network Stack

(stack version V0.1.0)

May 29, 2015

Revision History

Revision	**Date**	**Author(s)**	**Description**
0.1	05.02.15	HSO	created
0.2	06.02.15	HSO	input
0.3	23.02.15	HSO	input
0.4	23.02.15	HSO	SCons input
0.5	09.03.15	HSO	minor changes
0.6	16.03.15	HSO	detailed memory consumption
0.7	31.03.15	HSO	overwork with minor changes
0.8	07.04.15	HSO	demo applications

Contents

List of Figures

List of Tables

Acronyms

BSP Board Support Package.

CoAP Constrained Application Protocol.

HAL Hardware Abstraction Layer.

IoT Internet of Things.

MTU Maximum Transmission Unit.

RPL Routing Protocol for Lossy Networks.

UDP User Datagram Protocol.

Glossary

BSP A Board Support Package ....

CoAP The Constrained Application Protocol ....

HAL A Hardware Abstraction Layer ....

IoT Internet of Things describes ....

MTU Maximum Transmission Unit describes ....

RPL The Routing Protocol for Lossy Networks ....

UDP User Datagram Protocol ....

1 Introduction

1.1 6LoWPAN

In the last decade, IPv6 over Low power Wireless Personal Area Networks, also known as 6LoWPAN, has well evolved as a primary contender for short range wireless communication and holds the promise of an Internet of Things, which is completely based on the Internet Protocol.
The IEEE 802.15.4 standard specifies a maximum frame size of 127 bytes where the IPv6 specification requires a minimum Maximum Transmission Unit (MTU) of 1280 byte. With the 6LoWPAN adaptation layer it is possible to make use of IPv6 in small and constrained wireless networks which follows the IEEE 802.15.4 standard.
In the meantime, various 6LoWPAN implementations are available, be it open source or commercial. One of the open source implementations is the C-based emb6 stack which is described in this document.

1.2 Use Cases

The emb6-stack is optimized for the use in constrained devices without an operation system. The stack operates event driven with a scalable buffer handling for optimization on different platforms.
The typical field of application is in wireless sensor networks, e.g. for home automation or industrial environments. Some use cases in a smart home environment are shown in Figure 1.1.

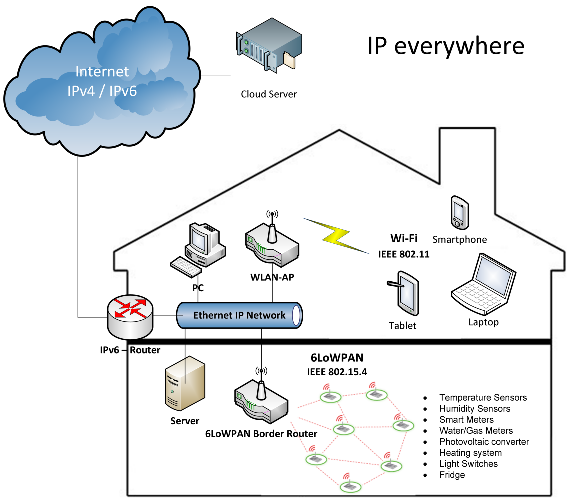


Figure 1.1: 6LoWPAN in smart home applications

2 The emb6 Network Stack

2.1 Overview

The main objective of the emb6 Networking Stack is to connect embedded devices to the Internet of Things (IoT). Therefore the emb6 Networking Stacks implements all the necessary parts of the 6LoWPAN protocol as described in chapter 1.1. Furthermore it provides additional functionalities such as different application layers and utilities. The following subchapters describe the features, basic concepts and the architecture of the emb6 Networking Stack.

2.2 Features, Concepts and Benchmarks

There are several IoT capable stacks available may it be on open source or on commercial basis. The emb6 Networking Stack provides several salient features making it a unique offering. The main features and concepts of the stack are the following:

- **Event Driven Operation** - Very small RAM overhead, one memory stack for the whole system.
- **Scalable Buffer Handling** - A common buffer module is used across layer and module boundaries. This decreases memory usage and furthermore provides scalability for usage on different hardware configurations and limitations.
- **Static memory management** - For additional stability during runtime.
- **Compile Time Options** - Usage of different compile-time settings help to make an optimum selection regarding to the anticipated use cases as well as to the hardware limitations such as memory size or computing performance.
- **Run Time Options** - Many stack parameters are accessible and changeable during runtime via remote management.
- **Full IPv6 support** - The integrated IPv6 protocol is based on the uIP-Stack [1]. This provides full IPv6 support and guarantees further support and maintenance.
- **BSP and HAL Abstraction** - Hardware dependencies are abstracted with a Board Support Package (BSP) which offers an API between the target and the applications and with a hardware abstraction layer Hardware Abstraction Layer (HAL), which allows independence from the used microcontroller IC.

- **Optimized for use in constrained devices** - The scalability of the stack enables a manifold use in highly diverse embedded systems.
- **Routing functionality** - The routing functionality is provided by the Routing Protocol for Lossy Networks (RPL) protocol [2],
- **Layered Architecture** - The design of the software stack follows a strict layered architecture (cf. Fig. 2.1).
- **Simple Setup and Configuration** - The setup and configuration can be easily executed with the help of centralized configuration files.
- **Socket Interface** - A BSD like socket API allows easy integration of customized applications.
- **Set of included Application Layer Protocols** - like Constrained Application Protocol (CoAP), ETSI, LWM2M
- **Security Support** - A DTLS1.2 protocol stack from the same protocol family can be integrated to ensure transport layer security [3].

Since the emb6 Networking Stack was mainly developed for the usage with resource constrained embedded devices, benchmarks especially regarding to memory consumption in FLASH and RAM are key points of the stack implementation. As the emb6 Networking Stack can be configured in many ways and all changes within a configuration affect the resulting memory usage, it is nearly impossible to provide a common number here. However the following table 2.1 gives a basic overview of the memory consumption caused by the different configuration based on a sample implementation for different targets with gnu-gcc compiler and activated code optimization level.

Stack Configuration	**stk3600** Flash/Ram 45,7/3,4kB	**xpro_212b** Flash/Ram 46,9/3,4kB	**atany900** Flash/Ram 46,6/2,8kB	**atany900_rfd** Flash/Ram 21,4/1,0kB
COAP:	11,6 / 1,3kB	11,9 / 1,3kB	12,5 / 1,1kB	
RPL:	13,2 / 0,3kB	14,4 / 0,3kB	11,4 / 0,2kB	10,1 / 0,1kB
IPV6:	16,4 / 1,5kB	15,5 / 1,5kB	14,7 / 1,2kB	10,5 / 0,7kB
SICSLOWPAN:	4,5 / 0,3kB	5,1 / 0,3kB	0,8 / 0,3kB	0,8 / 0,2kB

Table 2.1: Sample memory usage

2.3 Architecture

The basic architecture of most of the network stacks follows the so-called Open Systems Interconnection (ISO OSI) reference model, which splits a stack and its communication tasks into several vertical layers. The network related parts of the emb6 network stack also follow this strict layered architecture as shown in figure 2.1. Well-defined interfaces allow easy exchange, modification or removal of single layers as needed. Furthermore, development and maintenance efforts will be decreased.

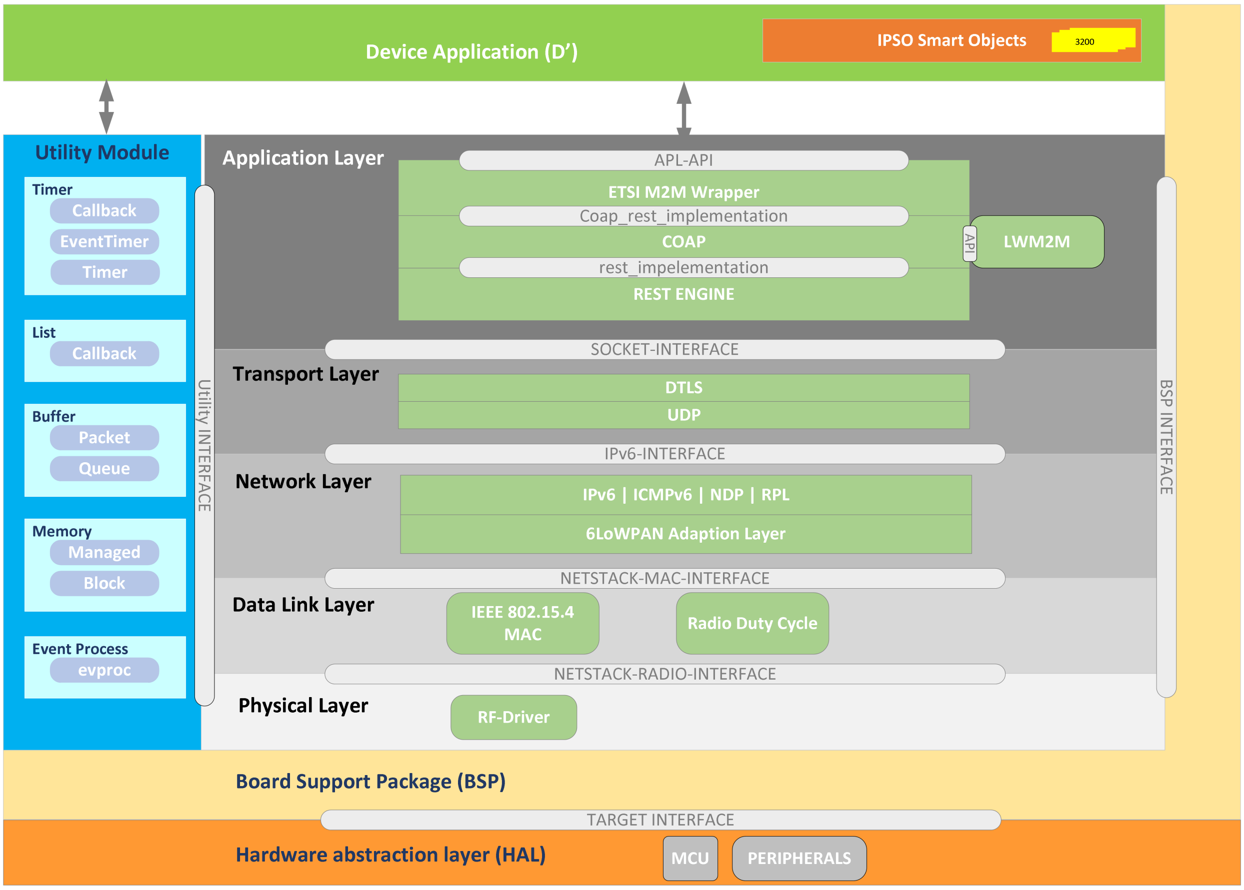


Figure 2.1: Protocol Stack of emb6

Figure 2.1 shows the basic architecture of the emb6 network stack with its networking core in the middle of the block diagram. The Networking core handles the network related tasks, mainly the communication part, whereas the different tasks have been split up into several layers. Beginning on top at the Application Layer (APL), usually serving as interface to the device application, requests will be forwarded layer by layer down to the physical layer (PHY) which is responsible for the implementation of the RF-module drivers. A detailed description of the single layers can be seen from ch. 2.4.1 to ch. 2.4.5.
Besides the networking core, a separate so called Utility Module implements all common functionalities such as timer and event handling, which are used by all other layers and modules. A detailed description of the Utility Module can be found in ch. 3.6.
To support different hardware platforms including different microcontrollers, RF modules and target boards all hardware dependent parts of the emb6 networking stack are encapsulated in a separate so-called Board Support Package (BSP) which is accessing a hardware dependent hardware abstraction layer (HAL). This allows easy porting of the emb6 Networking Stack across different hardware platforms. Detailed descriptions of the BSP/HAL architecture can be found in 3.6.4. Furthermore the emb6 Porting Guide (under construction) provides all information and instructions needed to port the emb6 Networking Stack onto a new platform.

2.4 Description of the Layers

2.4.1 Application layer

The application layers (APLs) are the highest layers of the emb6 Networking Stack and are located above the transport layer (TPL). The APL is an optional part of the emb6 Networking Stack. Depending on the application, different APLs may be used whereas the following are currently included:

- **Constrained Application Protocol (CoAP)** - The CoAP [4] protocol is HTTP like protocol adapted and optimized for the IoT. It is based on restful services and an according rest-engine which both are also part of this APL.
- **ETSI** - Under Development
- **LWM2M** - Under Development

If there is no need for an APL or a proprietary APL shall be used, it is also possible to make direct use of the underlying TPL's socket interface and to communicate using the available transport protocols.

2.4.2 Transport layer (socket interface)

The transport layer is based on the uIP embedded TCP/IP Stack [5]. The transport layer in this stack actually supports only UDP and provides a socket interface to use the data transfer over UDP.

2.4.3 Network layer

The network layer contains two sublayers, the upper IPv6 layer and the lower 6LoWPAN adaption layer. The IPv6 layer includes the routing protocol (RPL) , ICMPv6 and the neighbor discovery protocol (NDP).
The 6LoWPAN adaptation layer provides IPv6 and UDP header compression and fragmentation to transport IPv6 packets with a maximum transmission (MTU) of 1280 bytes over IEEE 802.15.4 with a MTU of 127 byte.

2.4.4 MAC layer (to be extended with a real mac protocol)

Actually just wrapper functions for stable functionality for IEEE802.15.4 (but no full feature).
Radio duty cycle as a part of IEEE 802.15.4 but not used at the moment.
At the moment CSMA functionality is done by the radio transceiver.

2.4.5 PHY layers

The physical layer is represented by the radio-interface driver and supports hardware depended functionality of the transceiver, e.g. CSMA and auto retransmission.

2.4.6 Utility Module

The Utility Module provides services to all layers of the emb6 network stack. Here the main event processing and timer handling is done. Also the queue- and packet buffer management is implemented in this part of the stack.

2.5 API architecture

The emb6 network stack makes use of a structure which is declared in the main function. This structure is named "netstack struct". The netstack structure contains a pointer for each layer which points to a structure with functions provided by the layer. With the pointers the layer specific functions for initializing and packet processing are used. The type definition of the netstack structure gives an overview of the layer structures, cf. Listing 2.1.

```
typedef struct netstack {
  const struct netstack_headerCompression*    hc;
  const struct netstack_highMac*          hmac;
  const struct netstack_lowMac*         lmac;
  const struct netstack_framer*         frame;
  const struct netstack_interface*         inif;
}s_ns_t;
```

Listing 2.1: global netstack struct definition

3 Using emb6

3.1 Introduction

This chapter describes the steps to make use of the emb6 - embedded IPv6 network stack. The parameters are listed and the initialization and configuration steps are shown.

3.2 Repository

The structure of the emb6 repository is shown in Figure 3.1.

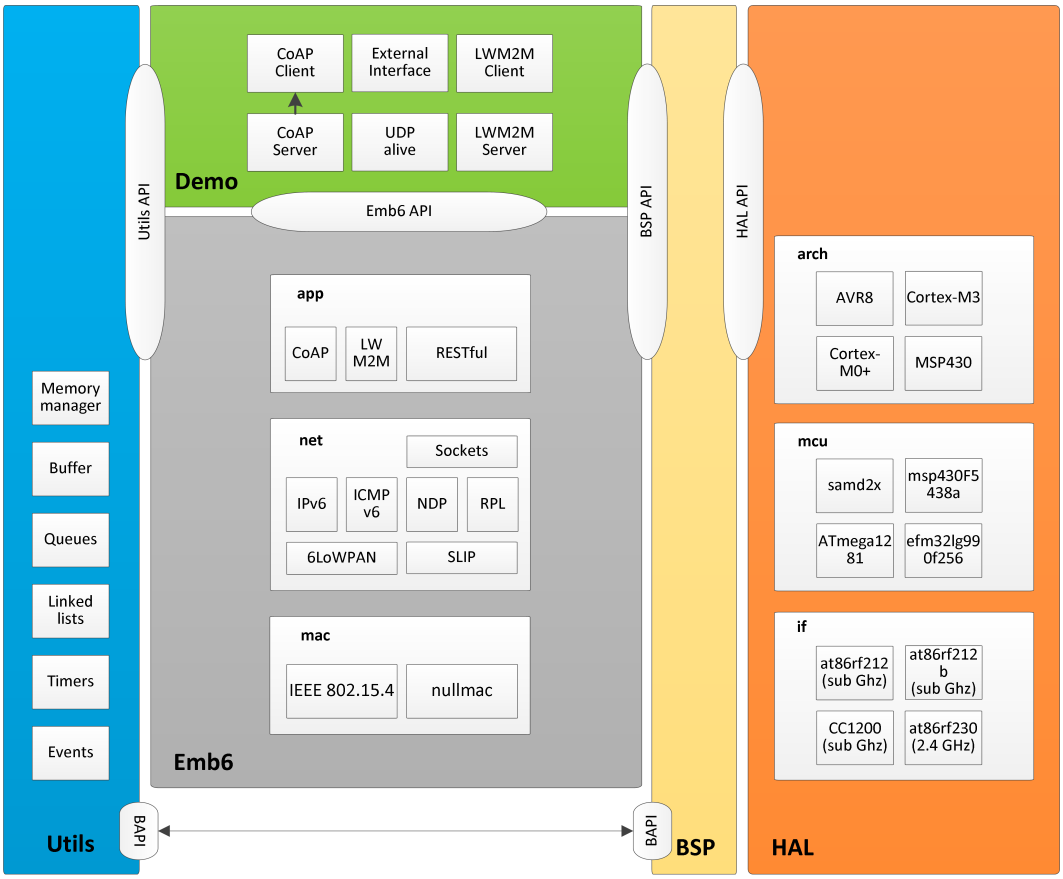


Figure 3.1: emb6 repository structure

3.3 File and Folder structure

```
\-- trunk
    |-- demo
    |   |-- coap
    |   |   |-- client
    |   |   |   |-- demo_coap_cli.c
    |   |   |   |-- demo_coap_cli.h
    |   |   |   \-- SConscript
    |   |   \-- server
    |   |       |-- demo_coap_srv.c
    |   |       |-- demo_coap_srv.h
    |   |       |-- resources
    |   |       |   |-- old res
    |   |       |   |-- res-led_toggle.c
    |   |       |   |-- res-push.c
    |   |       |   |-- res-rf_info.c
    |   |       |   |-- res-temp.c
    |   |       |   \-- res-toggle.c
    |   |       \-- SConscript
    |   |-- demo_main.c
    |   |-- extif
    |   |   |-- cmd.c
    |   |   |-- cmd.h
    |   |   |-- packetutils.c
    |   |   |-- packetutils.h
    |   |   |-- SConscript
    |   |   |-- slip.c
    |   |   |-- slip.h
    |   |   |-- slip_radio.c
    |   |   |-- slip-radio.c
    |   |   \-- slip_radio.h
    |   |-- lwm2m
    |   |   |-- client
    |   |   |   |-- lwm2mclient.c
    |   |   |   |-- object_device.c
    |   |   |   |-- object_firmware.c
    |   |   |   |-- object_security.c
    |   |   |   |-- object_server.c
    |   |   |   |-- SConscript
    |   |   |   \-- test_object.c
    |   |   |-- server
    |   |   |   |-- lwm2mserver.c
    |   |   |   \-- SConscript
    |   |   |-- TLV
    |   |   |   |-- CMakeLists.txt
    |   |   |   \-- decode.c
    |   |   \-- utils
    |   |       |-- commandline.c
    |   |       |-- commandline.h
    |   |       |-- connection.c
    |   |       \-- connection.h
    |   |-- mqtt
```

```
|   |   |-- demo_mqtt.h
|   |   |-- demo_mqtt_qos0pub_reg.c
|   |   \-- SConscript
|   |-- udp_alive
|   |   |-- demo_udp_alive.c
|   |   |-- demo_udp_alive.h
|   |   \-- SConscript
|   \-- udp-socket
|       |-- client
|       |   |-- demo_udp_cli.c
|       |   \-- demo_udp.h
|       \-- server
|           |-- demo_udp.h
|           \-- demo_udp_srv.c
|-- doxy_doc
|   |-- mainpage.dox
|   |-- mig_guide.dox
|   \-- src_org.dox
|-- emb6
|   |-- emb6.c
|   |-- emb6_conf.h
|   |-- emb6.h
|   |-- inc
|   |   |-- mac
|   |   |   |-- frame802154.h
|   |   |   |-- framer-802154.h
|   |   |   |-- mac.h
|   |   |   |-- nullmac.h
|   |   |   |-- nullrdc.h
|   |   |   |-- rdc.h
|   |   |   |-- rimeaddr.h
|   |   |   |-- rimestats.h
|   |   |   \-- sicslowmac.h
|   |   |-- net
|   |   |   |-- ipv6
|   |   |   |   |-- nbr-table.h
|   |   |   |   |-- rime.h
|   |   |   |   |-- tcpip.h
|   |   |   |   |-- uip_arch.h
|   |   |   |   |-- uip-debug.h
|   |   |   |   |-- uip-ds6.h
|   |   |   |   |-- uip-ds6-nbr.h
|   |   |   |   |-- uip-ds6-route.h
|   |   |   |   |-- uip-fw.h
|   |   |   |   |-- uip.h
|   |   |   |   |-- uip-icmp6.h
|   |   |   |   |-- uiplib.h
|   |   |   |   |-- uip-nd6.h
|   |   |   |   |-- uipopt.h
|   |   |   |   |-- uip-packetqueue.h
|   |   |   |   |-- uip-split.h
|   |   |   |   \-- uip-udp-packet.h
|   |   |   |-- rpl
```

```
|   |   |   |   |-- rpl.h
|   |   |   |   \-- rpl-private.h
|   |   |   \-- sicslowpan
|   |   |       \-- sicslowpan.h
|   |   \-- apl
|   |       |-- er-coap
|   |       |   |-- er-coap-block1.h
|   |       |   |-- er-coap-conf.h
|   |       |   |-- er-coap-constants.h
|   |       |   |-- er-coap-engine.h
|   |       |   |-- er-coap.h
|   |       |   |-- er-coap-observe.h
|   |       |   |-- er-coap-separate.h
|   |       |   \-- er-coap-transactions.h
|   |       |-- lwm2m-core
|   |       |   |-- internals.h
|   |       |   \-- liblwm2m.h
|   |       |-- mqtt
|   |       |   |-- MQTTSNConnect.h
|   |       |   |-- MQTTSNPacket.h
|   |       |   |-- MQTTSNPublish.h
|   |       |   |-- MQTTSNSearch.h
|   |       |   |-- MQTTSNSubscribe.h
|   |       |   |-- MQTTSNUnsubscribe.h
|   |       |   \-- StackTrace.h
|   |       \-- rest-engine
|   |           |-- rest-constants.h
|   |           \-- rest-engine.h
|   |-- SConscript
|   \-- src
|       |-- mac
|       |   |-- frame802154.c
|       |   |-- framer-802154.c
|       |   |-- mac.c
|       |   |-- no-framer.c
|       |   |-- nullmac.c
|       |   |-- nullrdc.c
|       |   |-- rimeaddr.c
|       |   |-- rimestats.c
|       |   \-- sicslowmac.c
|       |-- net
|       |   |-- ipv6
|       |   |   |-- nbr-table.c
|       |   |   |-- tcpip.c
|       |   |   |-- uip6.c
|       |   |   |-- uip-debug.c
|       |   |   |-- uip-ds6.c
|       |   |   |-- uip-ds6-nbr.c
|       |   |   |-- uip-ds6-route.c
|       |   |   |-- uip-icmp6.c
|       |   |   |-- uiplib.c
|       |   |   |-- uip-nd6.c
|       |   |   |-- uip-packetqueue.c
```

```
|       |   |   |-- uip-split.c
|       |   |   \-- uip-udp-packet.c
|       |   |-- rpl
|       |   |   |-- rpl.c
|       |   |   |-- rpl-dag.c
|       |   |   |-- rpl-ext-header.c
|       |   |   |-- rpl-icmp6.c
|       |   |   |-- rpl-mrhof.c
|       |   |   \-- rpl-timers.c
|       |   |-- sicslowpan
|       |   |   \-- sicslowpan.c
|       |   \-- slip
|       |       \-- slip_net.c
|       \-- apl
|           |-- er-coap
|           |   |-- er-coap-block1.c
|           |   |-- er-coap.c
|           |   |-- er-coap-engine.c
|           |   |-- er-coap-observe.c
|           |   |-- er-coap-res-well-known-core.c
|           |   |-- er-coap-separate.c
|           |   \-- er-coap-transactions.c
|           |-- lwm2m-core
|           |   |-- liblwm2m.c
|           |   |-- list.c
|           |   |-- management.c
|           |   |-- objects.c
|           |   |-- observe.c
|           |   |-- packet.c
|           |   |-- registration.c
|           |   |-- tlv.c
|           |   |-- transaction.c
|           |   |-- uri.c
|           |   \-- utils.c
|           |-- mqtt
|           |   |-- MQTTSNConnectClient.c
|           |   |-- MQTTSNConnectServer.c
|           |   |-- MQTTSNDeserializePublish.c
|           |   |-- MQTTSNPacket.c
|           |   |-- MQTTSNSearchClient.c
|           |   |-- MQTTSNSearchServer.c
|           |   |-- MQTTSNSerializePublish.c
|           |   |-- MQTTSNSubscribeClient.c
|           |   |-- MQTTSNSubscribeServer.c
|           |   |-- MQTTSNUnsubscribeClient.c
|           |   \-- MQTTSNUnsubscribeServer.c
|           \-- rest-engine
|               \-- rest-engine.c
|-- SConscript
|-- SConsTargets
|-- SConstruct
|-- target
|   |-- arch
```

```
    |   |   |-- arm
    |   |   |   |-- cm0plus
    |   |   |   |   \-- atmel
    |   |   |   \-- cm3
    |   |   |       \-- silabs
    |   |   \-- avr
    |   |       \-- avr8
    |   |           \-- atmel
    |   |-- bsp
    |   |   |-- atany900
    |   |   |   |-- board_conf.c
    |   |   |   |-- board_conf.h
    |   |   |   \-- SConscript
    |   |   |-- atany900_basic
    |   |   |   |-- board_conf.c
    |   |   |   |-- board_conf.h
    |   |   |   \-- SConscript
    |   |   |-- atany900_pro3
    |   |   |   |-- board_conf.c
    |   |   |   |-- board_conf.h
    |   |   |   \-- SConscript
    |   |   |-- atany900_pro5
    |   |   |   |-- board_conf.c
    |   |   |   |-- board_conf.h
    |   |   |   \-- SConscript
    |   |   |-- bsp.c
    |   |   |-- efm32stk3600
    |   |   |   |-- board_conf.c
    |   |   |   |-- board_conf.h
    |   |   |   \-- SConscript
    |   |   |-- samd20xpro_rf212
    |   |   |   |-- board_conf.c
    |   |   |   |-- board_conf.h
    |   |   |   \-- SConscript
    |   |   \-- samd20xpro_rf212b
    |   |       |-- board_conf.c
    |   |       |-- board_conf.h
    |   |       \-- SConscript
    |   |-- bsp.h
    |   |-- if
    |   |   |-- at86rf212
    |   |   |   |-- at86rf212.c
    |   |   |   |-- at86rf212.h
    |   |   |   \-- at86rf212_regmap.h
    |   |   |-- at86rf212b
    |   |   |   |-- at86rf212b.c
    |   |   |   |-- at86rf212b.h
    |   |   |   \-- at86rf212b_regmap.h
    |   |   |-- at86rf230
    |   |   |   |-- at86rf230.c
    |   |   |   |-- at86rf230.h
    |   |   |   \-- at86rf230_regmap.h
    |   |   \-- fake_radio
```

```
|   |       |-- fake_radio.c
|   |       \-- fake_radio.h
|   |-- mcu
|   |   |-- atmega1281
|   |   |   |-- hwinit.h
|   |   |   \-- target.c
|   |   |-- efm32lg990f256
|   |   |   |-- hwinit.h
|   |   |   \-- target.c
|   |   |-- native
|   |   |   |-- hwinit.h
|   |   |   \-- target.c
|   |   |-- samd20g18
|   |   |   |-- conf_clocks.h
|   |   |   |-- hwinit.h
|   |   |   \-- target.c
|   |   |-- samd20j18
|   |   |   |-- conf_clocks.h
|   |   |   |-- hwinit.h
|   |   |   \-- target.c
|   |   \-- samd21g18a
|   |       |-- conf_clocks.h
|   |       |-- hwinit.h
|   |       \-- target.c
|   \-- target.h
\-- utils
    |-- inc
    |   |-- cc.h
    |   |-- clist.h
    |   |-- ctimer.h
    |   |-- etimer.h
    |   |-- evproc.h
    |   |-- logger.h
    |   |-- memb.h
    |   |-- mmem.h
    |   |-- packetbuf.h
    |   |-- queuebuf.h
    |   |-- random.h
    |   |-- stimer.h
    |   \-- timer.h
    \-- src
        |-- ctimer.c
        |-- etimer.c
        |-- evproc.c
        |-- list.c
        |-- memb.c
        |-- mmem.c
        |-- packetbuf.c
        |-- queuebuf.c
        |-- random.c
        |-- stimer.c
        \-- timer.c
```

Listing 3.1: File and Folder structure

3.4 emb6 stack setup parameters

3.4.1 Architecture

The stack is controlled and configured by two central files.

- emb6_conf.h: contains all compile time parameters
- emb6.h: contains the definitions and declarations of structures with function pointers to access the specific layer functions

The emb6 network stack makes use of a global structure called netstack which is defined and declared in the emb6.h. (cf. chapter 2.5). The netstack-structure contains a pointer to each layer structure which contains likewise function pointers to each accessible function of the layer. Over this the layer specific functions for initializing and packet processing can be used. The layer specific structures are also defined and declared in the emb6.h, initialization is done in the specific source file of the layer.

3.4.2 Compile time parameter

The emb6_conf.h contains the compile time parameters. These parameters cover constants and preprocessor instructions. With them, it is possible to configure the stack for individual device specific use cases. Table 3.1 shows the most important parameters.

Section	Parameter	Default Value	Description
Application Layer	EMB_USE_DAGROOT	FALSE	If TRUE node act as DAG root
	NETWORK_PREFIX_DODAG	0xaaaa, 0x0000, 0x0000, 0x0000	IPv6 network prefix for dag root
Transport Layer	UIP_CONF_UDP	FALSE	Define using UDP
	UIP_CONF_UDP_CONNS	4	number of concurrent UDP connections
	UIP_CONF_TCP	FALSE	Define using TCP, do not change
Network Layer	UIP_CONF_IPV6	TRUE	Define IPv6 as based protocol
	UIP_CONF_ICMP6	TRUE	Define using of ICMP6
	UIP_CONF_ROUTER	TRUE	Define router functionality
	NBR_TABLE_CONF_MAX _NEIGHBORS	10	number of entries in the Neighbour table
	UIP_CONF_MAX_ROUTES	10	number of entries in the Routing table
	UIP_CONF_DS6_ADDR_NBU	3	Unicast address list
	UIP_CONF_DS6_LL_NUD	TRUE	Should we use LinkLayer acks in NUD
	SICSLOWPAN_CONF _ACK_ALL	TRUE	Force acknowledge from sender
	SICSLOWPAN_CONF_FRAG	TRUE	Support of 6lowpan fragmentation
	SICSLOWPAN_CONF_MAXAGE	3 (seconds)	Most browsers reissue GETs after 3 seconds which stops frag reassembly, longer MAXAGE does no good
	SICSLOWPAN _CONF_COMPRESSION	SICSLOWPAN _COMPRESSION _HC06	6LoWPAN Header compression
	UIP_CONF_BUFFER_SIZE	240	Default uip_aligned_buf and sicslowpan_aligned_buf
Neighbor Discovery Config	UIP_ND6_SEND_RA	FALSE	enable/disable Router Advertisement sending, not needed when using RPL
	UIP_ND6_SEND_NA	FALSE	enable/disable Neighbor Advertisement sending, not needed when using RPL
RPL Section	RPL_LEAF_ONLY	FALSE	This value decides if this node must stay as a leaf or not, leaf => RFD (reduced function device)
	UIP_CONF_IPV6_RPL	TRUE	Enable/Disable RPL
	RPL_CONF_STATS	FALSE	Enable/Disable RPL statistics
	RPL_CONF_DAO_LATENCY	bsp_get(E_BSP _GET_TRES)	Set board depended latency
	RPL_CONF_DAG_MC	RPL_DAG _MC_ETX	Routing metric

Table 3.1: Overview Compile Time Parameter

3.4.3 RFD: Reduced Function Device

There are several parameters to set up a reduced function device, for the example in chapter 2.2 the parameters shown in table 3.2 are set.

Section	Parameter	Default Value	Description
Transport Layer	UIP_CONF_UDP_CONNS	2	number of concurrent UDP connections
Network Layer	NBR_TABLE_CONF_MAX_NEIGHBORS	5	number of entries in the Neighbour table
	UIP_CONF_MAX_ROUTES	5	number of entries in the Routing table
	UIP_CONF_BUFFER_SIZE	130	Default uip_aligned_buf and sicslowpan_aligned_buf
Neighbor Discovery Configuration	UIP_ND6_SEND_RA	FALSE	enable/disable Router Advertisement sending, not needed when using RPL
	UIP_ND6_SEND_NA	FALSE	enable/disable Neighbor Advertisement sending, not needed when using RPL
RPL Section	RPL_LEAF_ONLY	TRUE	This value decides if this node must stay as a leaf or not, leaf => RFD (reduced function device)

Table 3.2: Overview RFD Parameter

3.4.4 Run time parameter

<u>RPL Parameters</u>
The dynamic RPL parameters are relevant only if the node acts as a DAG-root. If the node acts as a normal router or leaf the RPL parameters are taken from the broadcast DIO messages.
The rpl_config structure is initialized in the emb6.c with the default values, shown in table 3.3.

structure	Element	Default Value	Description
s_rpl_conf_t rpl_config	DIO_interval_min	8	The RPL-DIO interval (n) represents 2^n ms
	DIO_interval_doublings	12	Maximum amount of timer doublings
	default_instance	0x1e	This value decides which DAG instance we should participate in by default
	init_link_metric	2	Initial metric attributed to a link when the ETX is unknown
	default_route_lifetime_unit	0xffff	Default route lifetime unit. This is the granularity of time used in RPL lifetime values, in seconds
	default_route_lifetime	0xff	Default route lifetime as a multiple of the lifetime unit

Table 3.3: Overview RPL dynamic configuration

The MAC and PHY parameters

Within this structure, initialized in the emb6.c, the initial RF-Transceiver settings are configured. By default the receive sensitivity and transmit power is set to maximum. The MAC address and PAN-ID are set to fixed values. These parameters can be changed during the initialization process. It is also possible to change these parameters during runtime but after this the RF-interface has to be reinitialized.
All the MAC and PHY parameters are shown in Tab. 3.4.

structure	Element	Default Value	Description
s_mac_phy_conf_t mac_phy_config	mac_address	0x00,0x50,0xc2,0xff, 0xfe,0xa8,0xdd,0xdd	64bit MAC address, used also as IPv6 address
	pan_id	0xabcd	PAN-ID
	init_power	11	Initial transmit power in dBm
	init_sensitivity	-100	Initial receive sensitivity dBm
	modulation	MODULATION_BPSK20	RF modulation type, possible values: MODULATION_QPSK100, MODULATION_BPSK20

Table 3.4: Overview MAC and PHY dynamic configuration

3.4.5 Board configuration

"Board" represents always a complete embedded device with connected and used periphery. Transceiver and other target board specific configurations and parameters can be found in the board_conf.h. This file contains e.g. the pinning of LEDs and RF-Transceiver.

3.5 Getting started with emb6

3.5.1 SCons build system

To build the software the tool SCons is used [6]. With the configuration file SconsTargets the general and initial setup is done. Here are the application, the board and the main parameters set. The complete setup procedure is summarized in the flow chart in figure 3.2:

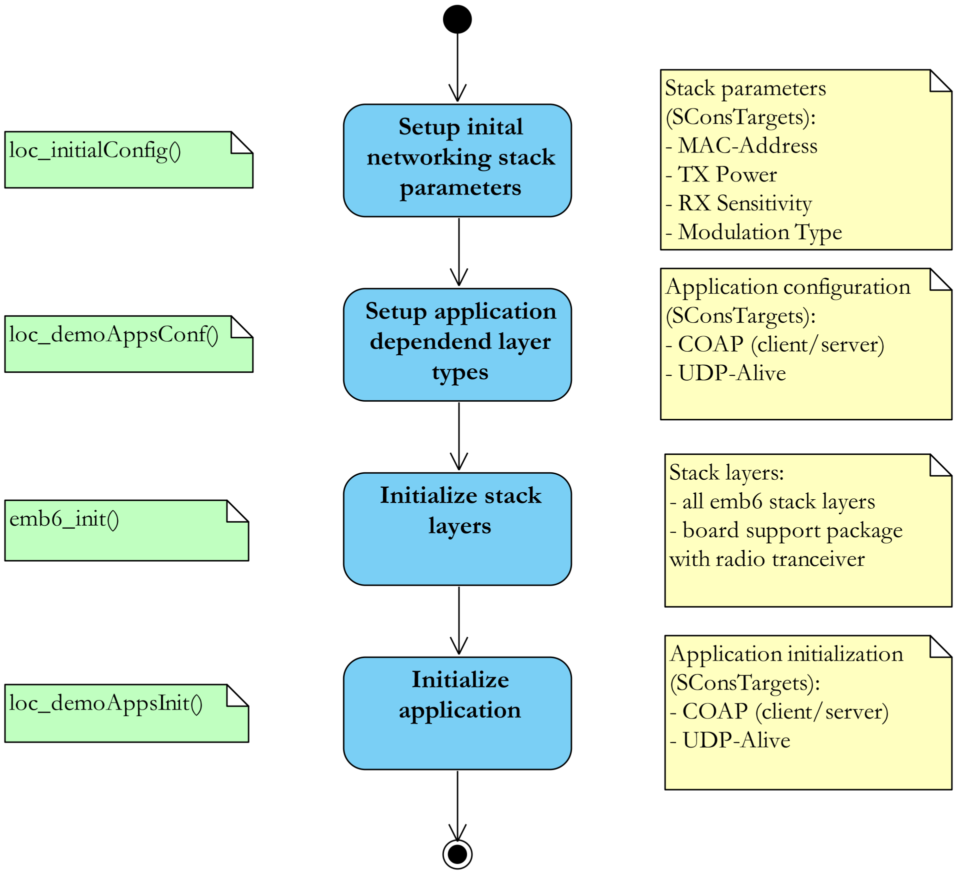


Figure 3.2: emb6 initialization diagram

3.5.2 Main initialization

The initialization of mandatory parameters is done in the demo_main.c file and is ordered as follows: First in the loc_initialconfig() optional customer settings are set using the mac_phy configuration struct. Without this initialization the default values are used. An example implementation of the initialization can be found in the demo_main.c file. The used application set up the proper stack pointers to the network stack structure in the loc_demoAppsConf() function. After that the emb6 stack can be initialized with the emb6_init(). This function initializes the full network stack and the radio interface. Finally the selected applications are initialized with the loc_demoAppsInit() function, see Figure 3.2.

```
// set initial stack parameters
  loc_initialConfig();
// set up stack configuration
  if (!loc_demoAppsConf(&st_netstack)) {
    return 0;
  }
// initialize layers
  if (emb6_init(&st_netstack)) {
    // initialize demo apps
    if (!loc_demoAppsInit()) {
      return 0;
  }
```

Listing 3.2: emb6 initialization sourcecode

3.5.3 Emb6 process

After successful initialization the function emb6_process(delay) is called (cf. Listing 3.3). The delay parameter sets an additional delay in μs. The default value is set to 500.
Inside the emb6_process() an endless loop is performed where the emb6 event management is handled, e.g. UDP transmissions, timeouts. It's now allowed to add additional code inside the loop, because of the strict event driven operation. If there are any customer applications, the event management of the emb6 stack has to be used.

```
// call process function with delay in us
emb6_process(500);
```

Listing 3.3: emb6 process call

3.5.4 Using emb6-stack within an operation system

The emb6 stack is often used in applications without an operating system. However, the use of an operating system is possible, when the emb6 initialization and the main are running in one task.

3.6 emb6 stack functional description

3.6.1 Event handling

The event driver engine functioned as a lib between the contiki-based timer management and the emb6 architecture without protothreads. The event handling is done in the evproc sources and headers. In the evproc.h all used event types are defined and the API functions are described for an easy use of the event handling. For each event a callback function is registered and invoked in the emb6 main function.

3.6.2 Timer management

The emb6 timer management is derived from the contiki sources and just adapted with small changes to fit in the emb6 stack. The API of the different timers is described in the related header files. More detailed information can be found under [7]

3.6.3 Packet buffer

A packet buffer is a structure that is used to create an outbound packet or store an inbound packet. A packet buffer is also used to operate on a packet and it can store only one packet at a time. More detailed information can be found under [8].

3.6.4 Board Support Package (BSP)

The BSP depicts a sublayer between network stack and hardware abstraction layer 3.6.5. With the BSP it is possible to use hardware functionality in a hardware independent way. On the one hand the BSP just acts as a wrapper of the HAL and one the other hat the BSP provides some added value with extra features e.g. to toggle a LED. These functions are constructed with the help of the utils section (cf. ch. 2.4.6)
The platform independent Board Support Package (BSP) provides its API description in the bsp.h, located in the bsp folder including the related source file.

3.6.5 Hardware Abstraction Layer

The hardware abstraction layer represents the hardware dependent functions for every target.
The API of the HAL is described in the target.h, located in the target folder. The target-specific sources are implemented in the target.c which can be found in the related MCU folder.

3.7 Example implementations

To make use of emb6 stack an example implementation which demonstrates the use of the socket interface for server and client functionality can be found in the /demo folder and there is also a CoAP server and client demo implementation included in the source code. It is also recommended to have a look at the er-rest-examples of the Contiki OS GitHub repository [9].

4 Demo Applications

4.1 Introduction to Demos

The demos included in this software release show the basic usage of the components of the emb6 stack.

4.2 CoAP Applications

CoAP is an application protocol suitable for exchanging messages on constrained nodes. It allows a client/server interaction model between endpoints. Two demo applications, showing a client and server role, are included in the emb6 stack.

4.2.1 CoAP Client

A CoAP client sends requests to a CoAP server whenever necessary. The CoAP client application sends periodic POST requests to a remote CoAP server. The address of the server can be configured in the demo_coap_cli.c file, using the SERVER_NODE(ipaddr) macro. The CoAP port of the server is configured by the REMOTE_PORT macro.

The aim of periodically sending POST requests is to toggle an LED resource located on a server. This LED toggle resource is included in the CoAP server demo(cf. ch. 4.2.2). The toggle interval in configured with the TOGGLE_INTERVAL macro. Table 4.1 summarizes useful attributes to configure the CoAP client demo as required.

Parameter	**Configuration mechanism**	**Description**
Server address	SERVER_NODE() macro	Found in demo_coap_cli.c
Server port	REMOTE_PORT macro	Found in demo_coap_cli.c
Resource URI on the destination server	service_urls[] char array	Found in demo_coap_cli.c
Request period	TOGGLE_INTERVAL macro	Found in demo_coap_cli.c

Table 4.1: CoAP client demo

4.2.2 CoAP Server

A CoAP server hosts resources which can be accessed through specified methods. When the server receives a request to a specific resource, the CoAP resource engine calls the associated method handler, if available.
Multiple resources are included in the CoAP server demo. The resource shall provide handlers for CoAP methods, i.e GET, POST, PUT, and DELETE. The signature of the handlers is as shown in Listing 4.1. Each resource is created in a separate file and is initialized in the CoAP resource engine when the application starts. This initialization is done using the RESOURCE(name, attributes, get_handler, post_handler, put_handler, delete_handler) macro.

```
/* signatures of handler functions */
typedef void (*restful_handler)(void *request, void *response, uint8_t *buffer,
uint16_t preferred_size, int32_t *offset);
```

Listing 4.1: Resource handler function signature

Table 4.2 describes the resources available in the demo and the function of each CoAP handler.

Resource name	**Handler description**			
	GET	POST	PUT	DELETE
res-led_toggle	Returns informational message	Toggles an LED on the board (if available)	-	-
res-push	Returns a counter value of this periodic resource	-	-	-
res-rf_info	Returns RSSI, Power, and Sensitivity value in dB	-	-	-
res-temp	Returns a dummy temperature value	-	-	-
res-toggle	Returns informational message	Switches ON an LED on the board (if available)	Switches OFF an LED on the board (if available)	-

Table 4.2: CoAP server demo resources

4.3 UDP Sockets

The User Datagram Protocol (UDP) demo applications show the use of standard UDP sockets. The client and server demo application work together to exchange message back-and-forth.

4.3.1 UDP Client

The UDP client periodically sends a sequence number that is incremented on the next use. The frequency of sending and the destination server address and port can be easily configured on the demo application (cf. Table 4.3). The _demo_udp_callback() function is called periodically to send message. This function also handles the receiving of messages from lower layers, i.e. UDP .

Parameter	Configuration mechanism	Description
Destination Server address	SERVER_IP_ADDR macro	Composed of NETWORK_PREFIX and SERVER_IP_ADDR_8_0 macros. It is found in demo_udp_cli.c
Destination Server port	__SERVER_PORT macro	Found in demo_udp_cli.c
Client UDP port	__CLIENT_PORT macro	Found in demo_udp_cli.c
Sending period	SEND_INTERVAL macro	Found in demo_udp_cli.c

Table 4.3: UDP client demo

4.3.2 UDP Server

The UDP server demo always waits for a data to receive. This demo works together with a UDP client demo application which is already included in this software release (cf. ch. 4.3.1). When the UDP server receives a random sequence number it increments the received value by one and sends it back to the client. Configurable parameters are described in Table 4.4.

Parameter	Configuration mechanism	Description
UDP Server port	__SERVER_PORT macro	Found in demo_udp_srv.c
Client UDP port	__CLIENT_PORT macro	Found in demo_udp_srv.c

Table 4.4: UDP server demo

4.4 UDP Keep Alive

This demo application provides DODAG visualization on a CETIC 6LBR router. The application sends UDP keep alive messages to the 6LBR border router. Sending of the keep alive messages is done periodically based on the time value configured in the SEND_INTERVAL macro. The SEND_INTERVAL can be found in the demo_udp_alive.c file.

5 Installation Guide

5.1 Operating System

The installation guide describes the installation of the development environment for Windows and Linux.

5.1.1 Windows

Windows 7 (x64)

5.1.2 Linux

Linux XUbuntu Version 4.4.1.
Attention: All example settings are set for this version. But other Linux distributions are possible, but have to be adapted for the installation.

5.2 Supported Targets

In the following table you will have an overview of the available targets with related information.

TARGET	MCU	RF	TOOLCHAIN	DEBUGGER
Atany900	ATMEGA1281	AT86RF212	avr-gcc	—
Efm32stk3600	EFM32LG990F256	AT86RF212b	arm-none-eabi-gcc	J-Link
SamD20XplainedPro	SAMD20J18	AT86RF212b	arm-none-eabi-gcc	OpenOCD

Table 5.1: Supported Targets

5.3 Setting up the development environment

5.3.1 Eclipse-IDE

Eclipse is an open source development environment and is used to develop the C/C++ based applications. Eclipse offers the possibilities to expand the functionality with plugins to the users need. For the selected

targets a cross-compiler and linker has to be installed. Therefore a basic environment for building C/C++ programs is necessary. To complete the environment a debugger has to be installed. Download the latest version of eclipse from:
https://eclipse.org/downloads/
with the compatible IDE extension for C/C++ Developer (CDT) and install the IDE. The version used for this installation guide was Kepler:
https://eclipse.org/downloads/packages/release/Kepler/SR2
but the version Luna is also possible:
https://eclipse.org/downloads/packages/release/Luna/R.
Additional installation instructions can be found on:
http://www.eclipse.org/cdt/downloads.php. Once the Eclipse packet is downloaded and installed several feature and plugins has to be added.

5.3.2 GCC Toolchain

GCC Toolchain for ARM and AVR architecture

Download and install the GCC tool chains for the ARM and AVR architecture from
https://launchpad.net/gcc-arm-embedded/+download
(Windows: gcc-arm-none-eabi-4_9-2014q4-20141203-win32.exe) and
http://www.atmel.com/microsite/atmel_studio6
In this Installation Guide Atmel Studio version 6.2 is used. The ARM tool chain is version 4.9.

GNU ARM Eclipse plugin

With the GNU ARM Eclipse plugin a add-on is provided for the developer to compile, link and debug the C/C++ applications in the eclipse environment.
The GNU ARM Eclipse plugin can be installed directly using the Eclipse IDE with the included software update procedure at "Help/Install New Software...".
Add following software repository
http://gnuarmeclipse.sourceforge.net/updates to have the plug in available.

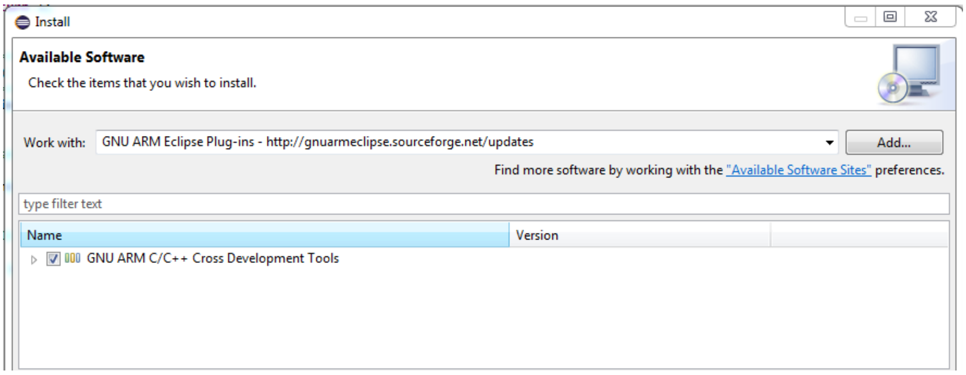


Figure 5.1: Installation of GNU ARM Eclipse plugin

Open the property dialog at "Project->Properties->C/C++-Build->Environment" and add the path of GNU ARM tool chain or the AVR Atmel Studio to the PATH variable of the belonging configuration.

ARM	\GNU Tools ARM Embedded\4.9 2014q4\bin
AVR	\Atmel\Atmel Toolchain\AVR8 GCC\Native\3.4.1061\avr8-gnu-toolchain\bin

Table 5.2: Path for GNU ARM and AVR Toolchain

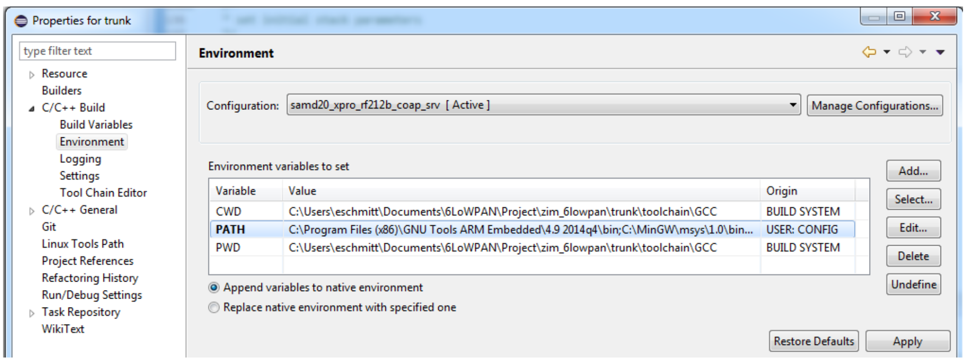


Figure 5.2: Path for GNU ARM Toolchain

5.3.3 SCons

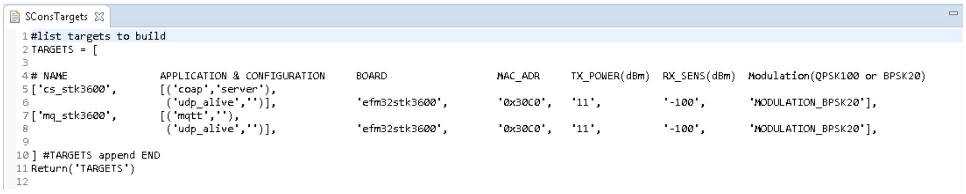
SCons is a replacement for make with improved features. SCons is based on Python what makes the tool very powerful for the build process. To install SCons an installation of Phyton and the Phyton-Win32-Extensions for Windows is required. Download the from following URL:
Phyton version 2.7.9 32bit: https://www.python.org/downloads/release/python-279/ (python-2.7.9.msi)
Attention: SCons requires a Phyton version below 3.0.
Phyton-Win32-Extensions for version 2.7.9 32bit: http://sourceforge.net/projects/pywin32/files/ (pywin32-219.win32-py2.7.exe)
SCons version 2.3.4: http://www.scons.org/download.php (Windows installer: scons-2.3.4-setup.exe)

Setup the target configurations

SCons provide the possibility to set up the several required target configuration in the SConsTarget file. This configuration can be build as described in the chapter below.
Example setup:

Application	coap as server
Board	efm32stk3600
MAC Address	0x30C0
Transmission Power	11dBm
Receiving Sensitivity	-100dBm
Type of Modulation	MODULATION_BPSK20

Table 5.3: Example for target configuration


SConsTargets

```
#list targets to build
TARGETS = [

# NAME              APPLICATION & CONFIGURATION     BOARD              MAC_ADR     TX_POWER(dBm)  RX_SENS(dBm)  Modulation(QPSK100 or BPSK20)
['cs_stk3600',      [('coap','server'),
                     ('udp_alive','')],              'efm32stk3600',    '0x30C0',   '11',          '-100',       'MODULATION_BPSK20'],
['mq_stk3600',      [('mqtt',''),
                     ('udp_alive','')],              'efm32stk3600',    '0x30C0',   '11',          '-100',       'MODULATION_BPSK20'],

] #TARGETS append END
Return('TARGETS')
```

Figure 5.3: Target configuration with SCons

SCons setup for the configurations

To build the configurations using SCons the build command and the path has to be adapted as follow:

For Windows: The default 'Build command' has to be modified exactly as shown in Figure 5.4 with the '.bat' extension, i.e. Build command: 'scons.bat target=cs_atany900'.

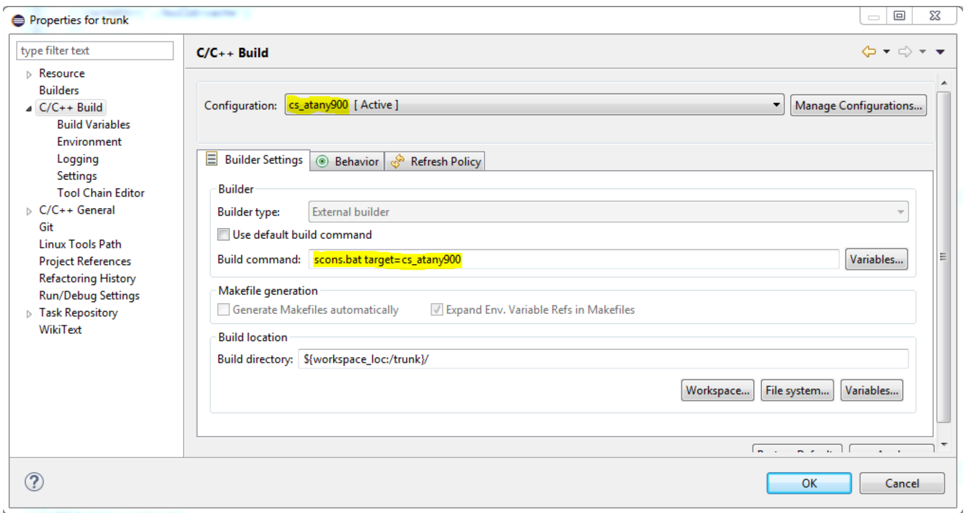


Figure 5.4: Setup for SCons build command for Windows

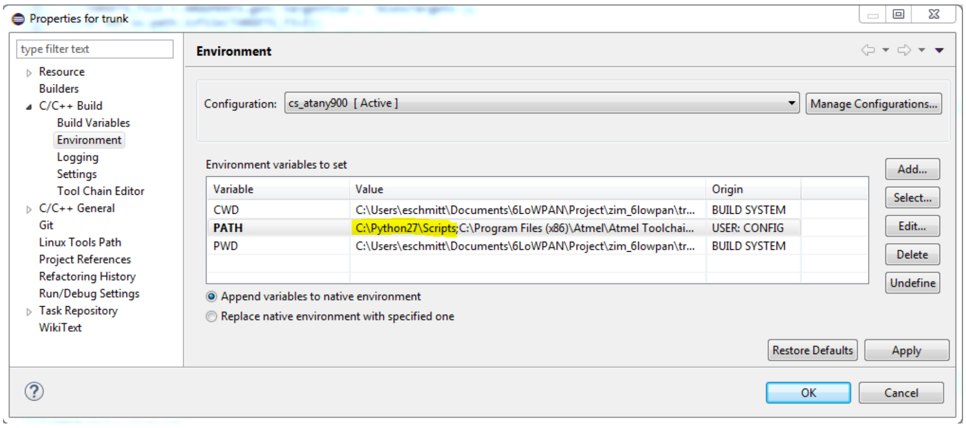


Figure 5.5: Path for SCons build procedure for Windows

Attention: Do not add the path to the global Windows PATH environment variable. Use the Eclipse specific PATH variable as shown in figure 5.5 to get access to the tool chain. Adapt the settings described above to your local settings and add this path setting to all used configurations.

For Linux:

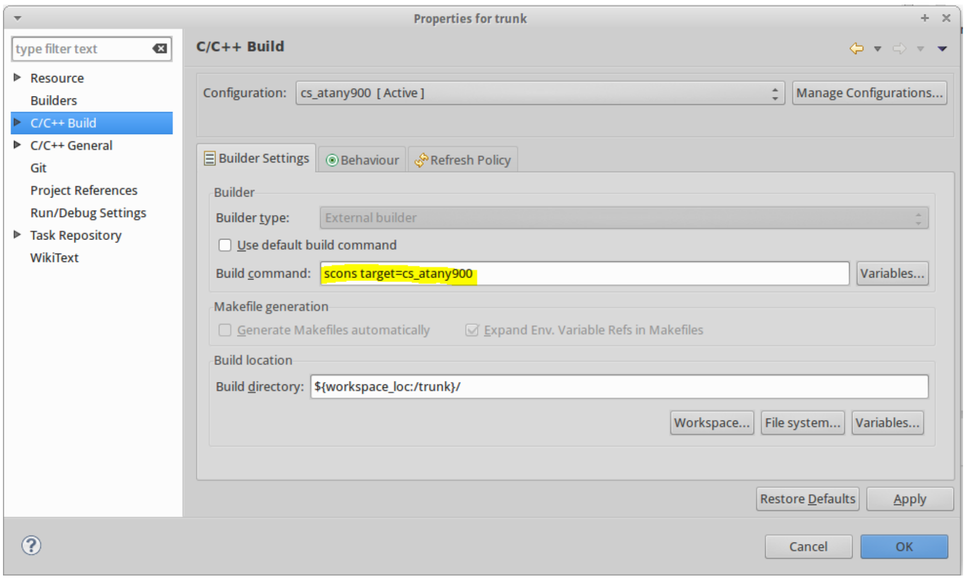


Figure 5.6: Setup for SCons build command for Linux

A special path setting is not necessary in the Linux environment.

5.4 Debug configuration

5.4.1 Target Atany900

JTAGICE mkII

The JTAGICE mkII is included in the AVR Studio. With "Tools->Device Programming" the Tool, Device and Interface can be set, as well as the output file.
Download the installation files from:
http://www.atmel.com/microsite/atmelstudio6/ and install it.

Attention: be sure to have built the .hex or .elf output file before programming the application

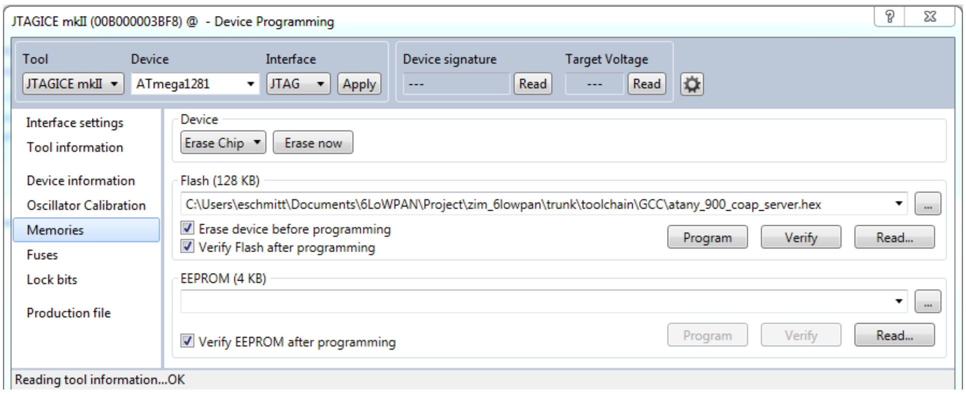


Figure 5.7: Device Programming with AVR-Studio

For Linux:
Install the avrdude package using synaptic or the apt-get command. The board can be flashed with the following command from the command line using the AVRJTAGICE:
"sudo avrdude -c jtag2 -P usb -p atmega1281 -e -U flash:w:cs_atany900.bin"

Segger J-Link

Segger J-Link is a debugger probe to access the hardware for debugging and programming purpose. Actually it is used for Atmel Cortex-M3 based MCU and the ATMEGA1281 MCU on the ATANY900 target. These boards do not have a J-Link debug hardware on board. Download the drivers from:
https://www.segger.com/jlink-software.html and install it. Currently no debug environment for the ATANY900 board is available. Therefore the Atmel-Studio is used to program the board. Following settings are necessary to setup up the programming.

5.4.2 Target Efm32Stk3600

5.4.3 Target SamD20

OpenOCD

OpenOCD is a debugger interface that is used from the GNU ARM debugger to access the hardware for debugging purpose via GDB command. Actually it is used for the Atmel SAMD20 Cortex-M0+ based MCU on the "SAM D20 Xplained Pro Evaluation Kit" which have the J-Link debug chip on board.

Windows (x32 or x64 version is):
Download the installation file from http://www.freddiechopin.info/en/download and unpack it at desired location. Be sure to use the Windows binaries for x32 or x64, both versions are possible. Open the preferences dialog at "Window->Preferences->Run/Debug->String Substitution" and add following name- and path- substitutions as the debug configuration information matching to your system.

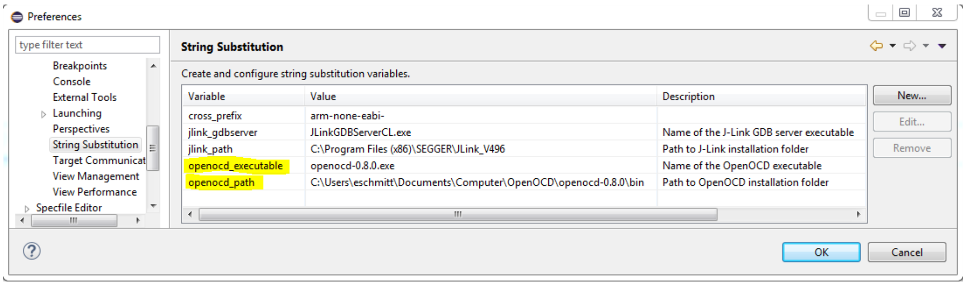


Figure 5.8: Path for OpenOCD Debugger

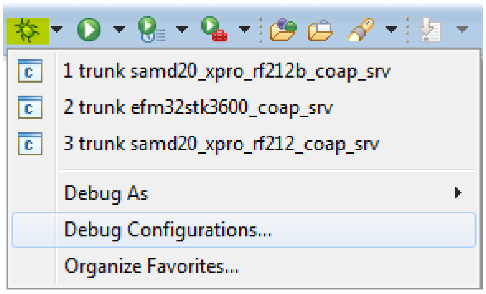


Figure 5.9: OpenOCD configuration - open dialog

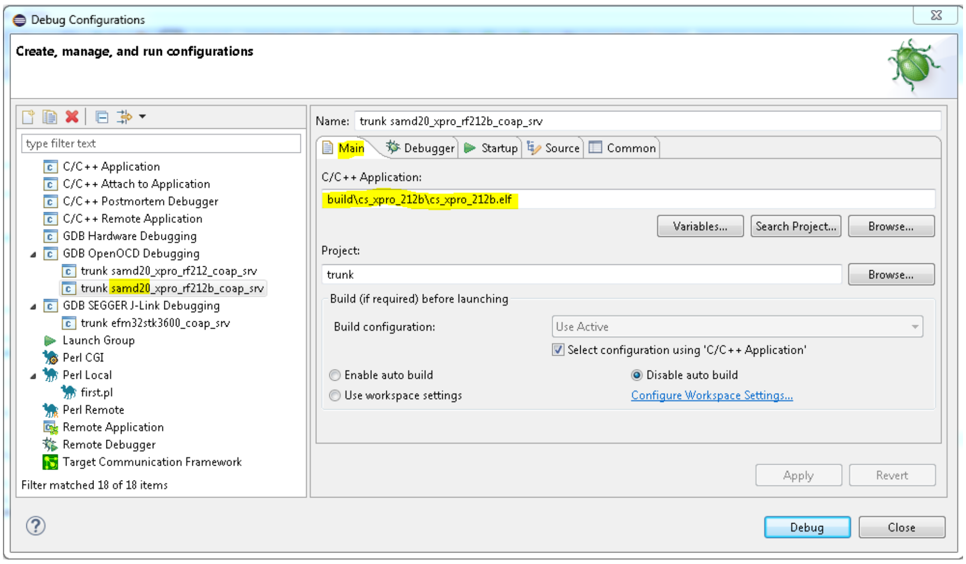


Figure 5.10: OpenOCD configuration - dialog main

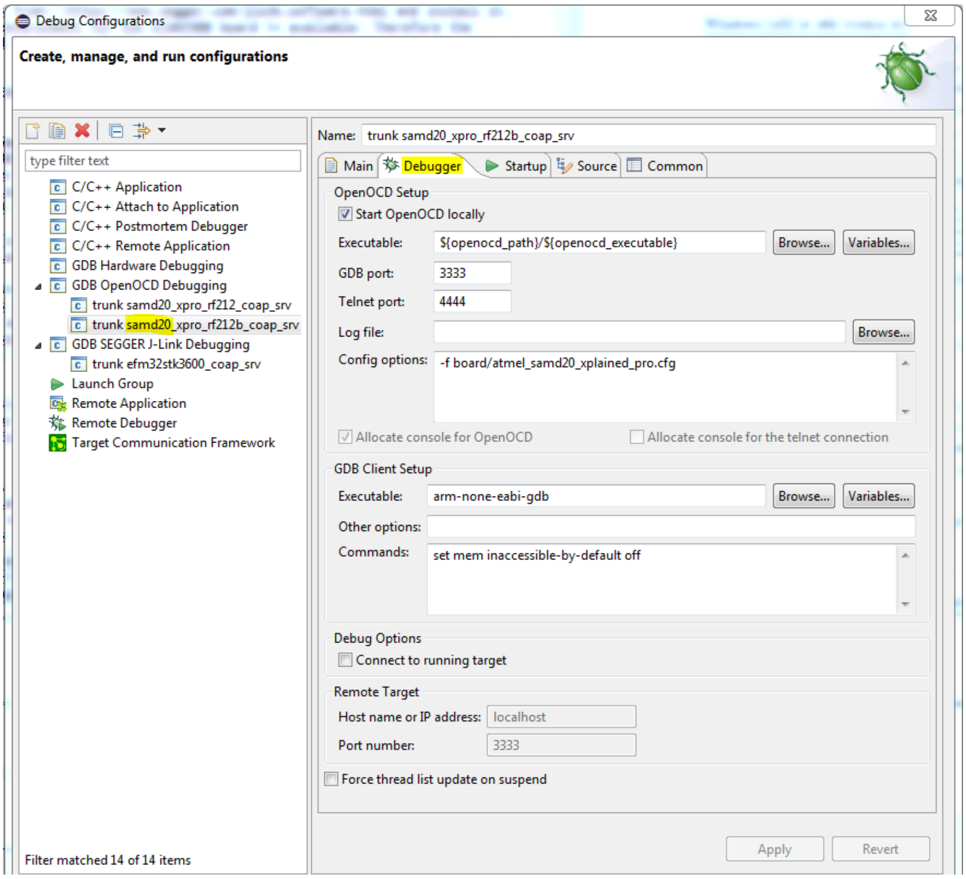


Figure 5.11: OpenOCD configuration - dialog debugger

Linux:

Download the installation file from http://sourceforge.net/projects/openocd/files/openocd/ and install it. Use version 0.8.0 to have the actual debug configuration files available. Open the preferences dialog at "Window->Preferences->Run/Debug->String Substitution" and add following name- and path-substitutions as the debug configuration information matching to your system.

Be sure that the USB port is available by using the HDI-API.

Release Notes

V0.1.0

Release date: 29.05.2015
Initial release for commit. This release does not have a feature or bug list yet since its just serves as reference for furhter releases.

Bibliography

[1] "Contiki-os," 2015. [Online]. Available: http://www.contiki-os.org/

[2] "Rfc6550," March 2012, rPL: IPv6 Routing Protocol for Low-Power and Lossy Networks.

[3] "Rfc6347," January 2012, datagram Transport Layer Security Version 1.2.

[4] "Rfc7252," June 2014, the Constrained Application Protocol (CoAP).

[5] "uip stack," 2015. [Online]. Available: http://en.wikipedia.org/wiki/UIP_(micro_IP)

[6] "Scons build system," 2015. [Online]. Available: http://www.scons.org/

[7] "Contiki timer management," 2015. [Online]. Available: https://github.com/contiki-os/contiki/wiki/Timers

[8] "Contiki packet buffer," 2015. [Online]. Available: http://anrg.usc.edu/contiki/index.php/Packetbuffer_Basic

[9] "Contiki-os, github repository," 2015. [Online]. Available: https://github.com/contiki-os/contiki