ATx

Smart Connect 6LoWPAN

USER GUIDE

Introduction

Smart Connect – 6LoWPAN is an Atmel version of IPv6 over Lossy and Low Power WPAN network stack for Internet of Things. It enables Atmel Smart power Microcontrollers to communicate to internet.

This user guide provides,

- Smart Connect 6LoWPAN architecture, package directory structure, understanding of low-power MAC and stack configurations.
- Example application provided in the release, supported platforms and references for help.
- Project files for Atmel Studio and gcc support.

Features

- Portable, low-power 6LoWPAN solution
- Contiki MAC support – Low power MAC
- RPL, IPHC, IPv6, IPv4 protocol support.
- UDP, TCP, HTTP, CoAP, MQTT and Web Sockets Application support.
- OTAU support with internal flash.
- Small foot-print size for entire 6LoWPAN solution.
- Support for ATMEL Cortex-M0 microcontrollers – SAMR21 and SAMD21.
- Support for ATMEL transceivers - AT86RF233 and AT86RF212B.
- Example applications description.

Table of Contents

Atmel Confidential: For Release Only Under Non-Disclosure Agreement (NDA)

Table of Figures

1 Smart Connect 6LoWPAN

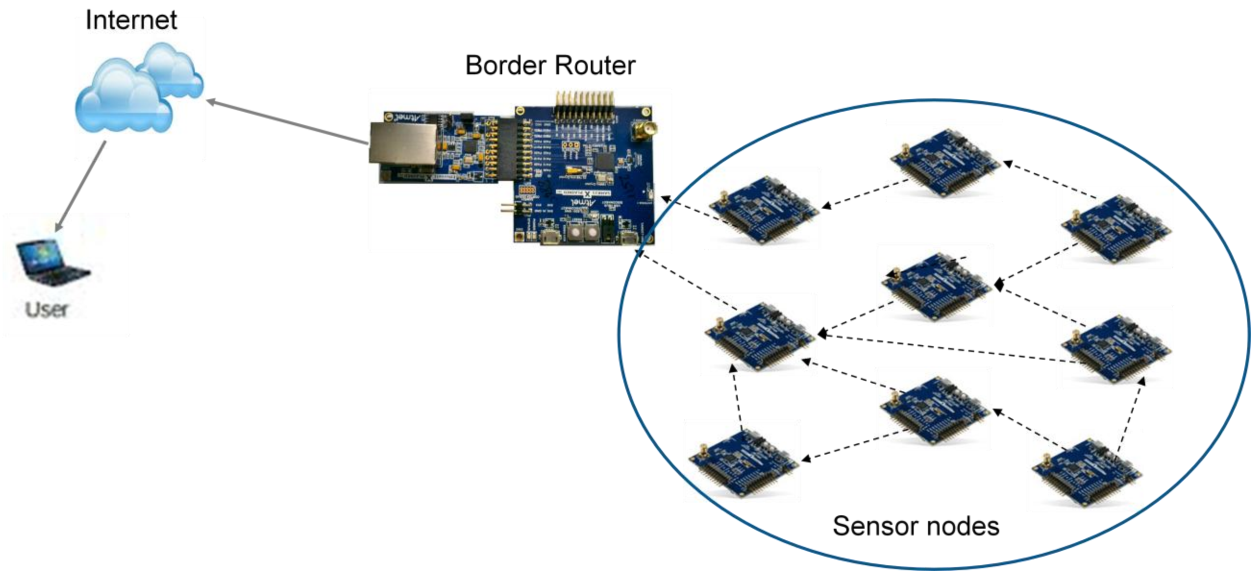
Smart Connect 6LoWPAN is Contiki [1] version with support for Atmel Microcontrollers and with additional features. Smart Connect 6LoWPAN is intended to support Internet of Things devices.

This 6LoWPAN stack has layered approach of Application to Physical layer and concentrates mainly on lightweight operation, low memory foot-print and connectivity to internet.

The wireless connection between nodes and from to edge node happens via IEEE 802.15.4 and IP layer brings in the connectivity to the outside world.

Typical Smart Connect 6LoWPAN network with sensor nodes and border router looks like Figure 1.

Figure 1: Typical Smart Connect 6LoWPAN network

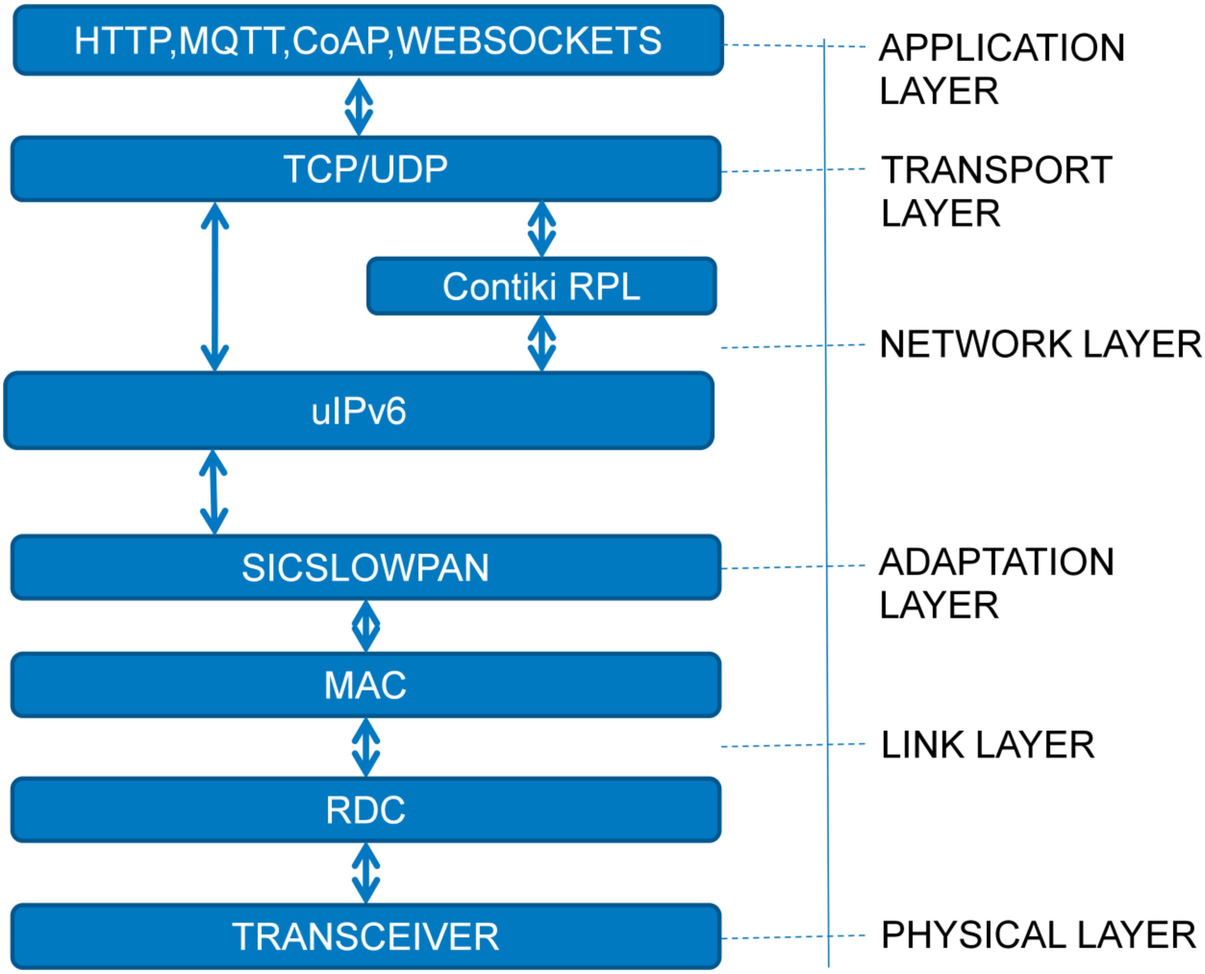


2 General Architecture

Smart Connect 6LoWPAN layer follows modular approach and has defined model to communicate between layers.

General architecture of Smart Connect 6LoWPAN is given below.

Figure 2 : Smart Connect 6LoWPAN architecture



INFO **The Smart Connect 6LoWPAN architecture is derived from standard Contiki OS architecture.**

TIP **To know more about Contiki OS architecture, please refer *www.contiki-os.org***

2.1 Stack Layers

Stack layers are described starting from bottom up:

Smart Connect 6LoWPAn offers different flavors' of modules to satisfy the scenario. All modules are configurable in nature. So based on the need in hand, any module can be added or removed.

Table 1 : Smart Connect 6LoWPAN layers

Layer	Protocol
Application	HTTP, MQTT, IETF CoAP/REST Engine
Transport	UDP, TCP
Network	IPv6 / RPL
Adaptation	Sicslowpan
MAC	CSMA, NULLMAC
RDC	CONTIKIMAC, NULLRDC
Physical	IEEE 802.15.4
Security	Link layer security

INFO **Reference to the description of each layers are given in the succeeding section of the document.**

2.1.1 Transceiver layer

This layer contains the transceiver specific functionality used for IEEE 802.15.4 MAC packets and also provides interface to RDC layer. Currently, the transceiver layer is implemented for individual transceivers.

Both 2.4GHz and sub-GHz transceiver is supported by the stack.

2.1.2 RDC Layer

Smart Connect 6LoWPAN stack supports two RDC layer,

2.1.2.1 NULL RDC layer

NULL RDC layer is a "null" RDC layer that always keeps the radio ON and used for testing purposes. No duty cycling is performed by this layer.

The NULL RDC driver is called:

```
nullrdc_driver
```

2.1.2.2 CONTIKIMAC RDC layer

CONTIKIMAC RDC layer is a low power radio duty cycling layer which keeps the radio most of the time OFF to achieve minimum power consumption, meanwhile synchronized with the nodes in the network in way to receive packets. This is achieved by the use of real timer module.

The CONTIKIMAC RDC driver is called:

```
contikimac_driver
```

2.1.3 MAC Layer

Smart connect 6LoWPAN support two MAC layers,

2.1.3.1 NULL MAC Layer

This layer takes packets from adaptation layer and transmits via RDC layers. This layer will not re-transmit any packets in case of any MAC failures.

The NULL MAC driver is called:

```
nullmac_driver
```

2.1.3.2 CSMA Layer

The CSMA layer receives and transmits packets using RDC layer. When MAC failure occurs, this layer will re-transmit the packets.

The CSMA layer driver is called:

```
csma_driver
```

2.1.4 SICSLOWPAN Layer

The SICSLOWPAN layer is the adaptation layer which interacts with Network layer and MAC layer.

SICSLOWPAN layer module handles the conversion of IP packets to IEEE 802.15.4 packets and vice versa.

The functions of the layers are,

- Receive IP/RPL packets from network layer.
- Fragmentation of IP packets whose size is greater than IEEE 802.15.4 MAC frames MAX size (128).
- IPv6 header compression/decompression.
- Re-assembly of received IPv6 fragmented packets.
- Provide interface for MAC layer to send the received MAC frame to IP layer.

This module is does not have a separate process as such. Instead it will be called by MAC layer, when new data is received by the node and similarly called by uIP layer to send a IP packet over IEEE 802.15.4 network.

When a packet is received by this module from uIP layer, this module does a header compression and it checks for size of IP packet data length. If the IP packet data length is greater than length of IEEE 802.15.4 payload size, it will fragment the IP packet and send in sequence.

In same fashion, if the data is received from MAC layer, this module will check whether the packet is fragmented or complete packet. If the packet is fragmented, it will keep the received packet in re-assembly buffer and wait for next fragment to receive. If the packet is not fragmented, it will do the header de-compression and forward the packet to uIP layer for further processing.

This module also does Header compression of IP packets.

INFO **Please refer *RFC4944* and *IETF 6LoWPAN Header Compression draft***

The SICSLOWPAN layer driver is called:

```
sicslowpan_driver
```

2.1.5 RPL – Routing protocol for Lossy and Low power Wireless PAN

RPL is routing protocol IETF ROLL standard for low power and lossy networks.

RPL is a tree oriented topology which has one or more root and it trickles downward to the leaf nodes.

Contiki RPL is a lightweight and power efficient version of standard IETF RPL.

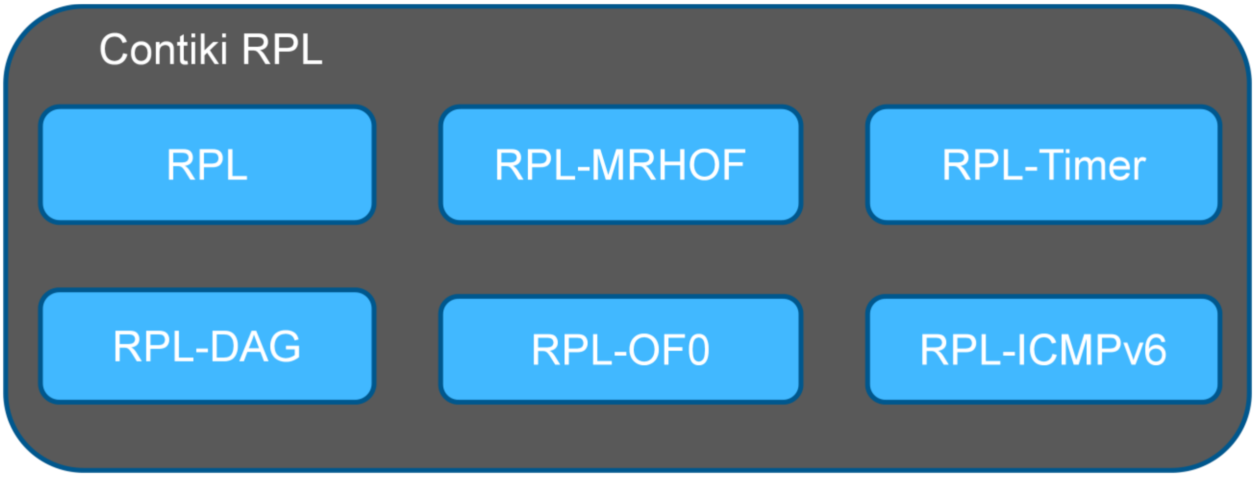
Contiki RPL is implemented alongside the uIP layer and RPL layer uses ICMPv6 for sending RPL packets.

Figure 3 displays the sub-modules inside the Contiki RPL module and description of each sub-module is followed subsequently.

2.1.5.1 RPL

This sub-module handles the initialization of the Contiki RPL module. The main functions of this sub-module are add/remove routes, initialization and purging routes.

Figure 3 : Contiki RPL Block Diagram



2.1.5.2 RPL-MRHOF

The Minimum Rank Hysteresis Objective Function sub-module uses estimated number of transmission (ETX) as a additive routing metric and also provides stubs for energy metric.

2.1.5.3 RPL-Timer

This sub-module handles the timer management of Contiki RPL and provides timer support to other sub-modules within Contiki RPL module.

2.1.5.4 RPL-DAG

The RPL-DAG sub-module has the logic for Directed Acyclic Graphs in RPL.

The following functions are handled by this sub-module.

1. Allocation and freeing of new DAG.
2. Selection and joining the available DAGs.

3. Neighbor handling.
4. Parent selection and Parent table handling.
5. New RPL instance creation and de-allocation.
6. Selection and joining existing RPL instance.
7. Global and local DAG repair.
8. Re-calculation of ranks.
9. Send and process DAO and DIO messages.

2.1.5.5 RPL-OF0

This sub-module contains Objective Function 0 logic. It provides the logic for selecting best DAG available and calculates the RANK of the node.

2.1.5.6 RPL-ICMPv6

This sub-module is used to prepare and process RPL control messages like DAO, DIO and DIS.

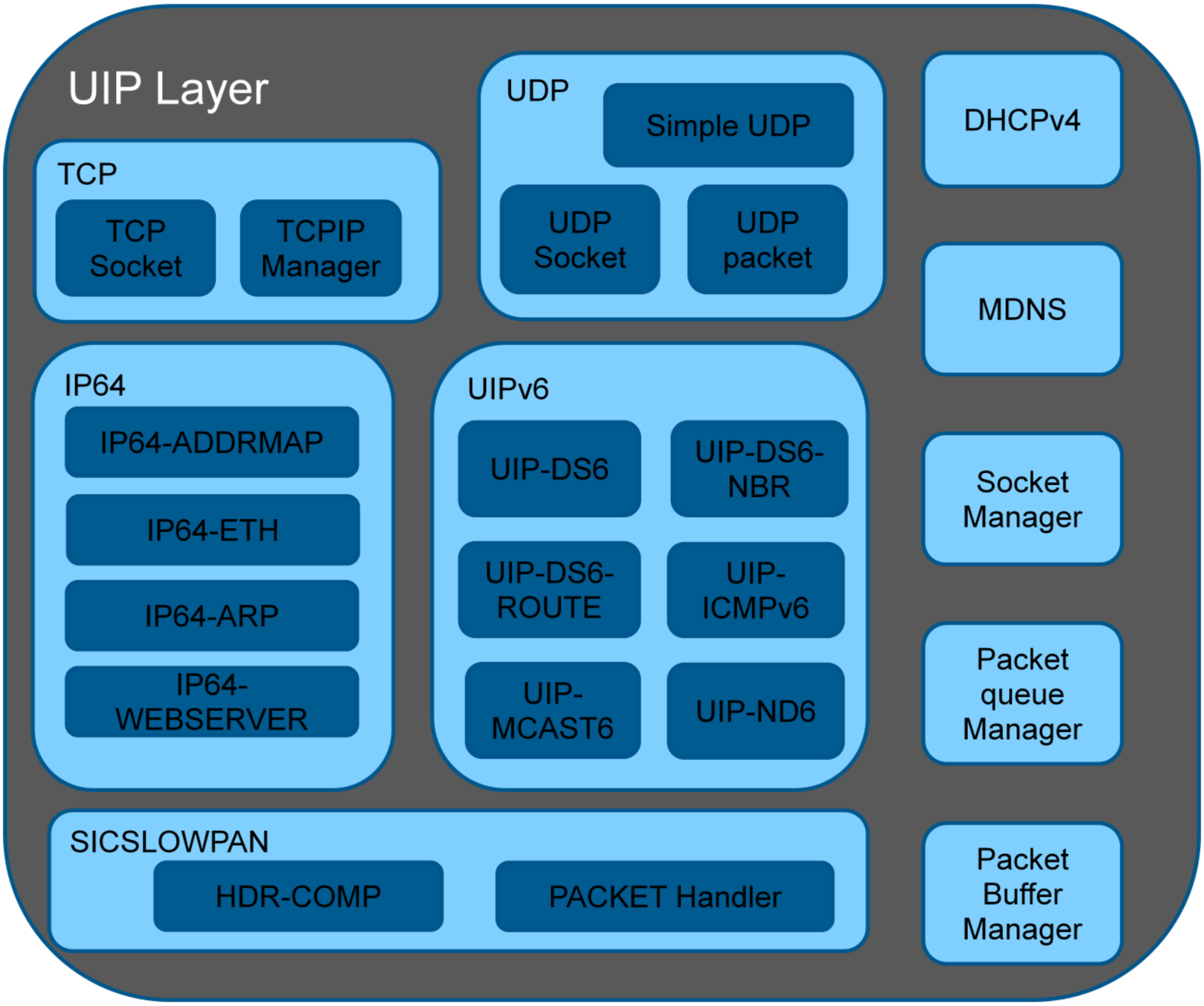
INFO **Refer to draft specifications released by the ROLL working group for detailed study of RPL – *RFC6550***

2.1.6 uIP layer

In Smart Connect 6LoWPAN, the uIP layer handles the transport and network layer functionalities.

The following Figure 4 displays the functional modules and respective sub-modules of uIP layer.

Figure 4 : uIP Layer Block Diagram



2.1.6.1 Packet Queue Manager

In Smart Connect 6LoWPAN stack, the Packet Queue sub-module contains the handling of queuing and de-queuing of IP packets buffers.

2.1.6.2 Packet Buffer Manager

The packet buffer manager handles the buffer management of uIP stack module.

2.1.6.3 TCPIP manager

This process handles the TCP connections and TCP sync and other timer modules.

The TCPIP process handles all the IP packets delivery to respective processes.

2.1.6.4 TCP socket process

This process initializes the TCP sockets and handles the socket connections.

This process exposes the TCP socket APIs for application layer.

2.1.6.5 UDP packet sub-module

This sub-module is used to send UDP packets through uIP layer.

2.1.6.6 UDP socket

UDP socket process initializes the UDP sockets requested by Upper layers.

2.1.6.7 Simple UDP layer

The UDP socket layer is complex and so simple UDP layer exposes APIs to application in simplified manner and handles the UDP sockets in conjunction with UDP socket layer.

Simple UDP layer can be found in file simple-udp.c under core/net/ip/ directory.

2.1.6.8 MDNS

The functions of the layer are,

1. The uIP DNS resolver functions are used to lookup a hostname and map it to a numerical IP address.
2. It maintains a list of resolved hostnames that can be queried with the resolv_lookup() function.
3. New hostnames can be resolved using the resolv_query() function.
4. The event resolv_event_found is posted when a hostname has been resolved. It is up to the receiving process to determine if the correct hostname has been found by calling the resolv_lookup() function with the hostname.

MDNS module files can be found in apps/mdns directory.

2.1.6.9 DHCPv4

DHCPv4 process is for Dynamic Host Resolution Protocol for IP version 4 connections.

The functions of this layer are,

1. This process will search for DHCP server in the internet by sending discover message using send_discover() function and establish a DHCP connection by sending a DHCP request message using send_request() function.
2. Once the connection is established, the DHCP process will obtain a Dynamic IP address for the Device.

Smart Connect 6LoWPAn uses dhcpv4.c in apps/ip64 directory as DHCPv4 module.

2.1.6.10 uIP-DS6

This sub-module handles functions of the IPv6 data structures. This comprises part of Neighbor discovery and auto configuration state machines.

uIP-DS6 sub-module can be found in file uip-ds6.c under core\net\ipv6 directory.

ℹ **INFO** **Please refer RFC 4862 for Neighbor discovery and RFC 4361 for auto configuration state machines.**

2.1.6.11 uIP-DS6-NBR

This sub-module functions includes as follows,

1. Maintaining Neighbor table.
2. Add, Remove Neighbors in the table.
3. Address mapping table of link layer address and IPv6 address of devices in the Neighbor table

UIP-DS6-NBR sub-module can be found in file uip-ds6-nbr.c under core\net\ipv6 directory.

2.1.6.12 UIP-DS6-ROUTE

This sub-module functions includes as follows,

1. Maintains Neighbor route table in conjunction with central Neighbor table.
2. This table contains an entry for each route which specifies that which neighbor the given route as to go through.
3. Add/Remove the route entry, update of next hop neighbor and other route parameters.
4. Initialization and Add/Remove route list based on necessity.

UIP-DS6-ROUTE sub-module can be found in file uip-ds6-route.c under core\net\ipv6 directory.

2.1.6.13 uIP-ICMPv6

uIP-ICMPv6 sub-module handles the ICMPv6 echo messages and error messages for IP layer.

UIP-ICMPv6 sub-module can be found in file uip-icmpv6.c under core\net\ipv6 directory.

2.1.6.14 uIP-MCAST6

This module includes the support for IPv6 Multicast to uIPv6 layer.

Currently, two engines are supported for uIPv6 layer,

1. Stateless Multicast RPL forwarding (SMRF) – This is lightweight engine and handles the datagram forwarding.
2. Multicast Forwarding with Trickle – This engine contains the algorithm described in IETF ROLL specification *draft-ietf-roll-trickle-mcast-11.* This ROLL specifies Multicast protocol for Lossy and Low-Power Networks (MPL).

INFO

Please refer *draft-ietf-roll-trickle-mcast-11* for more information on MPL and for SMRF *here*.

2.1.6.15 uIP-ND6

This sub-module contains the implementation of Neighbor Discovery algorithm described in RFC 4861.

The functions of this sub-module are,

1. Neighbor Advertisement Preparing and Processing messages.
2. Router Advertisement Preparing and Processing messages.

uIP-ND6 sub-module can be found in file uip-nd6.c under core/ipv6 directory.

2.1.6.16 IP64

The ip64 module is a translator between IPv6 and IPv4 packets. The IPv6 packets come from an IPv6 network and are translated into a single IPv4 host. The IPv6 network typically is a low-power RF network and the IPv4 network typically is an Ethernet.

IP64 maps all IPv6 addresses from inside the IPv6 network to its own IPv4 address. This IPv4 address would typically have been obtained with DHCP from the IPv4 network, but the exact way this has been obtained is outside the scope of the ip64 module. The IPv4 address is given to the ip64 module through the ip64_set_ipv4_address () function.

2.1.6.17 IP64-ADDRMAP

IP64-ADDRMAP sub-module handles the adding, storing, recycling and removing the IPv6 to IPv4 address conversion in ip64-addrmap table.

2.1.6.18 IP64-ETH

IP64-ETH sub-module consists of both ip64-eth and ip64-eth-interface sub-module.

The functionality of this sub-module is to interact with configured Ethernet interface and send/receive data via the Ethernet interface.

2.1.6.19 IP64-ARP

This sub-module handles Address Resolution Protocol message parsing and maintaining an ARP table for the Devices in the internet network.

Once the IP address obtained for a particular device, ARP table will be updated with a new entry. And if not an ARP request will be sent in the IP network.

If the Destination IP address is not found in the connected network, the IP packet will be forwarded to the default router IP address.

2.1.6.20 IP64-WEBSERVER

This sub-module has default HTML page to be displayed by the device.

For more information, please refer br-webserver-node example.

2.1.6.21 SICSLOWPAN layer

Please refer ***section 2.1.4***

2.2 Other stack components

Other stack components include HAL services such as timer, sleep module,

2.2.1 Timer module

Stack Module uses three types of timers

2.2.1.1 etimer (Event Timer):

Event timers provide a way to generate timed events.

An event timer will post an event to the process that set the timer when the event timer expires. An event timer is declared as a struct etimer and all access to the event timer is made by a pointer to the declared event timer.

Etimer uses a base hardware timer which runs in 8MHz.It generates interrupt every 8ms from which the timer value is calculated.

2.2.1.2 ctimer (Callback Timer):

The ctimer module provides a timer mechanism that calls a specified callback function when a ctimer expires.

Callback timer module also uses the same hardware timer for etimer and this will call the callback function on timer expiry.

2.2.1.3 rtimer (Real Timer Module):

The real-time module handles the scheduling and execution of real-time tasks (with predictable execution times).

The Real timer module is used for time critical tasks where the event cannot wait for other processes to finish. This uses a base hardware timer running at very lower resolution of 31.25 KHz.(rtimer priority).This Rtimer Should run in highest priority as this is for time critical events and to resume when in power down modes.

Hardware dependent timer module files can be found in the directory services\timer\sam0

Stack dependent timer files can be found in the directory core\sys

2.2.2 Sleep module

For Battery powered end nodes contikimac implements low power wakeup mechanism. In the Power saving cycle, Both the transceiver and MCU goes to power down mode.Sleep module is an add-on for the Low power MAC. It Uses the Internal RTC which is in synchronization with the radio duty cycling as a wakeup source and puts the MCU in a lowest Power Mode (Standby Mode).

In the contiki MAC module It can be configured whether MCU also enters into deep sleep Mode.It can be configured using the Macro CONF_MCU_SLEEP

Sleep Module functions can be found in the directory services\timer\rtimer-arch.c

2.2.3 Resource management

Core stack has a module for Memory block allocation and Management

The memory block allocation routines provide a simple yet powerful set of functions for managing a set of memory blocks of fixed size. A set of memory blocks is statically declared with the MEMB () macro. Memory blocks are allocated from the declared memory by the memb_alloc () function, and are deallocated with the memb_free () function. This Module is inside the core\sys.

The managed memory allocator is a fragmentation-free memory manager.

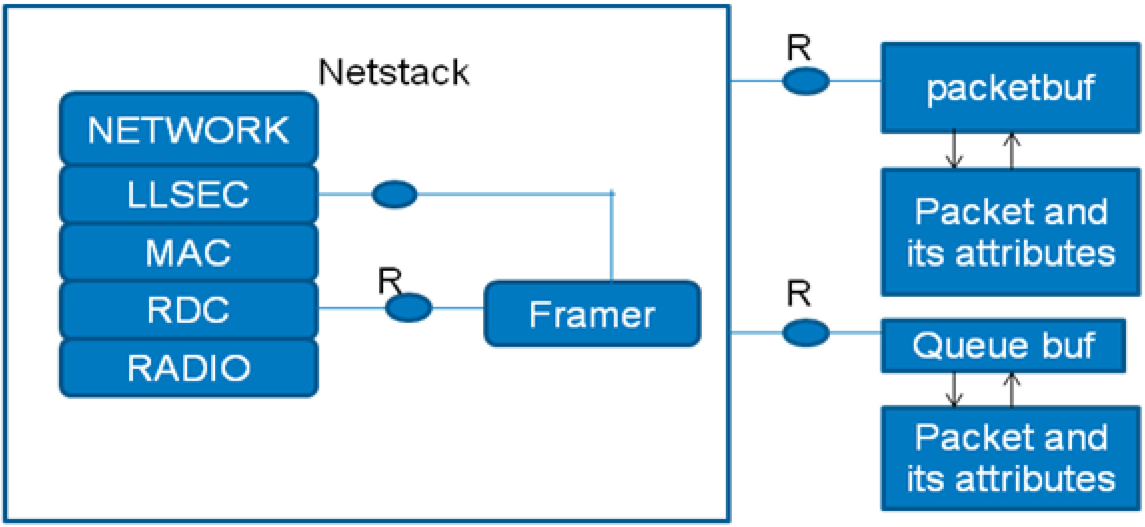
It keeps the allocated memory free from fragmentation by compacting the memory when blocks are freed. A program that uses the managed memory module cannot be sure that allocated memory stays in place. Therefore, a level of indirection is used: access to allocated memory must always be done using a special macro.

Memory module files can be found in the directory core\lib\ memb.c

2.2.4 Link layer security

Link layer security is implemented as a netstack layer, which is located in between the MAC and the NETWORK layer:

Figure 5: Link layer security



noncoresec is a noncompromise-resilient 802.15.4 security implementation, which uses a network-wide key. `coresec` is a compromise-resilient LLSEC driver. `coresec` implements the 802.15.4 security sublayer, the Adaptable Pairwise Key Establishment Scheme, as well as the Easy Broadcast Encryption and Authentication Protocol. Currently noncoresec support is available based on software aes.

ENABLE_AES Build switch configures the Link layer security in stack.

LLsec files can be found in the directory core\net\llsec

3 Understanding software package

3.1 Stack folder structure

Smart connect 6LoWPAN uses Contiki folder structure and uses general application modules from Contiki OS stack. Below given the Smart connect 6LoWPAN folder structure and this software package strategically placed under the thirdparty/wireless folder in the ASF directory.

```
├──addons
├───Smart Connect 6LoWPAN
│    ├───apps
│    │    ├───aes
│    │    ├───http-socket
│    │    ├───ip64-addr
│    │    ├───mdns
│    │    ├───simple-rpl
│    │    ├───websocket
│    │    ├───mqtt
│    │    ├───dhcp
│    │    ├───ip64
│    │    ├───ip64-webserver
│    │    ├───network-reboot
│    │    ├───rest-coap
│    │    ├───….
│    ├───core
│    │    ├───dev
│    │    ├───lib
│    │    ├───net
│    │    │    ├───ip
│    │    │    ├───ipv4
│    │    │    ├───ipv6
│    │    │    │    ├───multicast
│    │    │    ├───mac
│    │    │    │    ├───contikimac
│    │    │    ├───rpl
│    │    ├───sys
│    ├───dev
│    │    ├───at86rf212b
│    │    ├───at86rf233
│    ├───examples
│    │    ├──br-node
│    │    │    ├──samd21_xpro_reb233
│    │    │    ├──samr21_xpro
│    │    │    ├──samd21_xpro_reb212b
│    │    ├──br-webserver-node
│    │    │    ├──samd21_xpro_reb233
│    │    │    ├──samr21_xpro
│    │    │    ├──samd21_xpro_reb212b
│    │    ├──hello-world
│    │    │    ├──samd21_xpro_reb233
│    │    │    ├──samr21_xpro
│    │    │    ├──samd21_xpro_reb212b
│    │    ├──mesh-node
│    │    │    ├──samd21_xpro_reb233
│    │    │    ├──samr21_xpro
│    │    │    ├──samd21_xpro_reb212b
│    │    ├──udp-broadcast-contikimac
│    │    │    ├──samd21_xpro_reb233
│    │    │    ├──samr21_xpro
│    │    │    ├──samd21_xpro_reb212b
│    │    ├──udp-broadcast-example
│    │    │    ├──samd21_xpro_reb233
│    │    │    ├──samr21_xpro
│    │    │    ├──samd21_xpro_reb212b
│    │    ├──udp-unicast-sender
│    │    │    ├──samd21_xpro_reb233
│    │    │    ├──samr21_xpro
│    │    │    ├──samd21_xpro_reb212b
│    │    ├──udp-unicast-receiver
│    │    │    ├──samd21_xpro_reb233
│    │    │    ├──samr21_xpro
│    │    │    ├──samd21_xpro_reb212b
│    ├──services
│    │    ├──at24mac602
│    │    ├──ksz8852snl
│    │    ├──sam0
│    │    ├──timer
│    ├──doc
│    │    ├──Smart Connect 6LoWPAN user guide 1.0
```

These directories contain the following items.

- Apps:

  The thirdparty/wireless/SmartConnect_6LoWPAN/apps directory contains the application modules that are used in conjunction with core stack.

- Core:

  Core directory contains the main stack modules.

  - Lib – thirdparty/wireless/SmartConnect_6LoWPAN/core/lib module contains management modules that are required by stack to use like memory management, list management, etc.…
  - Sys – thirdparty/wireless/SmartConnect_6LoWPAN/core/sys module contains system modules like timers, process, auto-start module, etc.…
  - Net – net module contains the main protocol stack
    - Ip – thirdparty/wireless/SmartConnect_6LoWPAN/core/ip layer directory contains the general IP packet handling functional modules.
    - Ipv4 – thirdparty/wireless/SmartConnect_6LoWPAN/core/ipv4 directory contains IPv4 packet processing, forwarding modules.
    - Ipv6 – thirdparty/wireless/SmartConnect_6LoWPAN/core/ipv6 directory contains sicslowpan, icmpv6, neighbor discovery and multicast module.
    - Mac – thirdparty/wireless/SmartConnect_6LoWPAN/core/mac directory contains contikimac, nullmac, csma and other IEEE 802.15.4 related modules.
    - Rpl – thirdparty/wireless/SmartConnect_6LoWPAN/core/rpl directory contains the IETF RPL functional modules.

- Dev –

  thirdparty/wireless/SmartConnect_6LoWPAN/dev directory contains the transceiver functional modules.

- Doc – thirdparty/wireless/SmartConnect_6LoWPAN/doc directory contains the Smart Connect 6LoWPAN User guide document for reference.
- Examples –
  - thirdparty/wireless/SmartConnect_6LoWPAN/examples directory contains variety of examples for SmartConnect_6LoWPAN stack.
  - The provided makefiles and atmel studio projects can be used as quick start.

3.2 Application and Stack configuration

3.2.1 Stack Configuration

The stack and application based on the stack can be highly configured according to the end user application needs.

This requires a variety of build switches to be set appropriately.

The stack configuration macros can be referred from Contiki-conf.h

The following section describes that build switches may be used during the build process.

3.2.1.1 NETSTACK_CONF_MAC

This Build switch defines the Mac layer driver to be used in the stack either csma or null mac.

Please refer **section 2.1.3** for more information

```
#define NETSTACK_CONF_MAC                    nullmac_driver
```

3.2.1.2 NETSTACK_CONF_FRAMER

This Build switch defines the framer and used here is IEEE 802.15.4 framer according to the IEEE MAC 802 15.4 Specification.

```
#define NETSTACK_CONF_FRAMER                  framer_802154
```

3.2.1.3 NETSTACK_CONF_NETWORK

This Build switch defines the 6lowpan layer which format packets between the 802.15.4 and the IPv6 layers. Here it is configured to use the default siclowpan driver

```
#define NETSTACK_CONF_NETWORK                 sicslowpan_driver
```

3.2.1.4 NBR_TABLE_CONF_MAX_NEIGHBORS

This build switch configures the Neighbor table size for the Maximum number of neighbors.

```
#define NBR_TABLE_CONF_MAX_NEIGHBORS          20
```

3.2.1.5 UIP_CONF_IPV6

This build switch defines whether IPv6 network enabled or not.

```
#define UIP_CONF_IPV6                         1
```

INFO **Please note, even though the option is given, Stack is works for IPv6 network only.**

3.2.1.6 SICSLOWPAN_CONF_FRAG

This build switch defines whether to enable fragmentation of IP packets if the size of the IP packets is greater than IEEE 802.15.4 MAC frame payload size.

```
#define SICSLOWPAN_CONF_FRAG                  1
```

3.2.1.7 UIP_CONF_BUFFER_SIZE

This build switch configures the UIP buffer size.

This MACRO is has to be increased, if the application payload is expected to be greater than the configured size. Default value is 400.

```
#define UIP_CONF_BUFFER_SIZE            400
```

3.2.1.8 UIP_CONF_MAX_ROUTES

This build switch configures the Maximum number route entries that can be stored in the Route table.

```
#define UIP_CONF_MAX_ROUTES             400
```

3.2.1.9 ENABLE_AES

This build switch configures the software AES module used for link layer security. This switch has to be configured from project properties of Atmel studio project.

ENABLE_AES = 1: enables the link layer security

ENABLE_AES = 0: disables the link layer security

3.2.1.10 DATA_RATE

This Build switch is for sub-GHz band. This is to configure the band/channel page in sub-GHz.

Supported bands:

```
#define DATA_RATE BPSK_20

#define DATA_RATE BPSK_40

#define DATA_RATE OQPSK_SIN_RC_100

#define DATA_RATE OQPSK_SIN_250
```

3.2.2 Application configurations

Please refer Section 6 for individual Application configuration.

4 Brief about ASF

The following chapter gives a brief explanation about the Atmel Software Framework.

The Atmel Software Framework (ASF) is a MCU software library providing a large collection of embedded software for Atmel flash MCUs: megaAVR, AVR XMEGA, AVR UC3 and SAM devices.

4.1 ASF directory structure

1. For more details on ASF directory structure please refer to Atmel Software Framework http://www.atmel.com/tools/avrsoftwareframework.aspx http://asf.atmel.com/docs/latest/

Atmel Software Framework documentation [5].

5 Contiki MAC power optimization

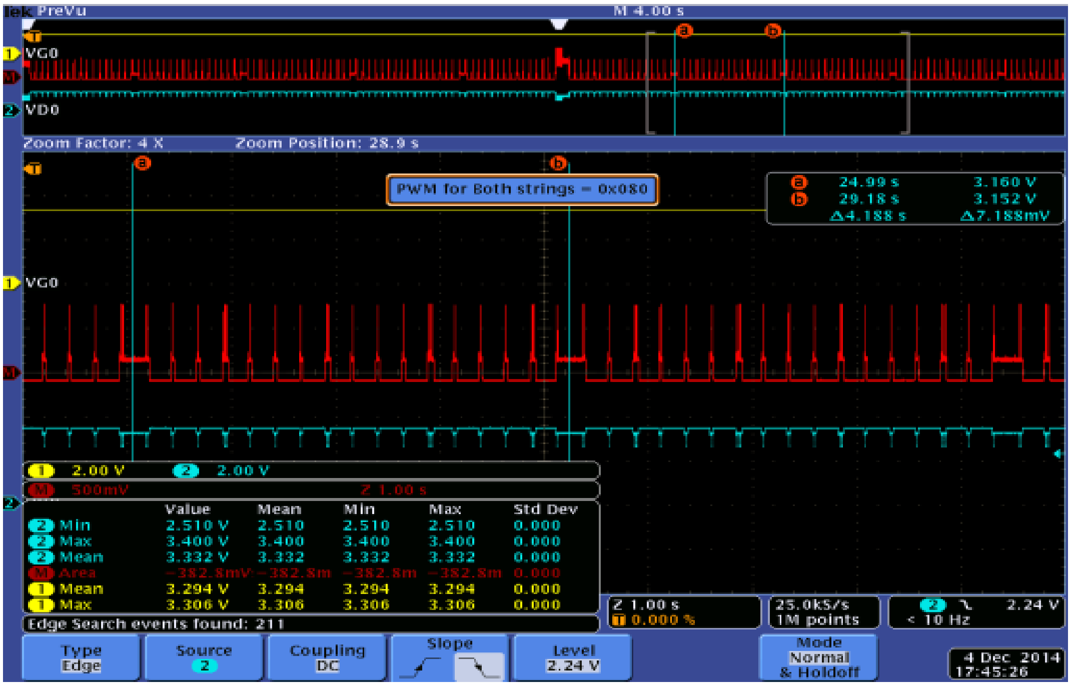
ContikiMAC is a simple low power radio duty cycling layer (RDC). ContikiMAC uses only asynchronous mechanisms, no signaling messages, and no additional packet headers. ContikiMAC is a radio duty cycling protocol that uses periodical wake-ups to listen for packet transmissions from neighbors. If a packet transmission is detected during a wake-up, the receiver is kept on to be able to receive the packet. When the packet is successfully received, the receiver sends a link layer acknowledgment. To transmit a packet, a sender repeatedly sends its packet until it receives a link layer acknowledgment from the receiver. Packets that are sent broadcasts do not result in link-layer acknowledgments. Instead, the sender repeatedly sends the packet during the full wake-up interval to ensure that all neighbors have received it.

Radio duty cycle can be configured according to the application by the Macro in contiki-conf.h.

```
#define NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 4
```

The below scope shot shows the periodic wakeups and CCA cycle of the MCU.

Figure 6: Contiki MAC power profiling data



	24.99 s	3.160 V
b	29.18 s	3.152 V
	Δ4.188 s	Δ7.188mV

1 2.00 V 2 2.00 V

M 500mV Z 1.00 s

	Value	Mean	Min	Max	Std Dev
2 Min	2.510 V	2.510	2.510	2.510	0.000
2 Max	3.400 V	3.400	3.400	3.400	0.000
2 Mean	3.332 V	3.332	3.332	3.332	0.000
M Area	−382.8mV	−382.8m	−382.8m	−382.8m	0.000
1 Mean	3.294 V	3.294	3.294	3.294	0.000
1 Max	3.306 V	3.306	3.306	3.306	0.000

Edge Search events found: 211

Z 1.00 s
T 0.000 %

25.0kS/s
1M points

2 2.24 V
< 10 Hz

Type Edge
Source 2
Coupling DC
Level 2.24 V
Mode Normal & Holdoff

4 Dec 2014
17:45:26

Results from above power measurements with `NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 4` configuration are,

Average Current Consumption: 137uA (for one duty cycle)

Maximum Current Consumption: 9.44mA

Minimum Current Consumption: 3.35uA

INFO **These power measurements are done with no transmission packets in the channel and in lab scenario.**

IMPORTANT **Please note this measurement may vary. These results are shown here are just for the reference.**

6 Example applications

The Smart Connect 6LoWPAN software package includes a variety of example applications which can be flashed on the supported hardware platforms and be executed immediately. On the other hand the complete source code is provided to help the application developer to more easily understand the proper utilization of the stack and to be able to build its own applications as fast as possible.

These applications will be explained in more detail in the subsequent sections. If the example application makes use of the UART interface, the UART is set to 115200.

6.1 Walking through a basic application

This section describes the example application released with the Smart Connect 6LoWPAN software package. The examples applications given in the package gives the insight of Smart Connect 6LoWPAN stack functionalities. These examples can be used by a developer as a starting point for further designs.

Detailed description of APIs can be viewed from doxygen documents.

6.2 hello-world example

This example is given just to understand the initialization procedure of the stack.

Once, Hello-world example is started, it will initialize all the configured required modules, initiate the RPL connection and starts a process thread to print “hello-world” in COM port of Host PC.

6.3 udp-broadcast-example

6.3.1 Introduction

This example uses Simple UDP module to send broadcast data using UDP socket.

This example demonstrates the stack ability to create a UDP socket and successfully send and receive UDP broadcast data over the network.

6.3.2 Configuration

Following are configuration required to do in example application, in conjunction with Stack configuration mentioned in Section 3.2

1. UDP port has to be mentioned for which Socket will be created.

```
#define UDP_PORT                      1234
```

2. Time interval for Periodic timer has to be configured.

```
#define SEND_INTERVAL                (5 * CLOCK_SECOND)
```

6.3.3 Application sequence

The sequence with which the example application executes, are given below,

Step 1: Stack initialization (done in broadcast-example-main.c)

Step 2: Start broadcast_example_process.

Step 3: Get link address and register UDP socket with simple_udp_sendto API.

Step 4: Start periodic timer with SEND_INTERVAL as periodic value.

Step 5: Once periodic timer expired, send broadcast data using the registered UDP socket.

Step 6: Repeats step 5 for every Periodic timer expiry.

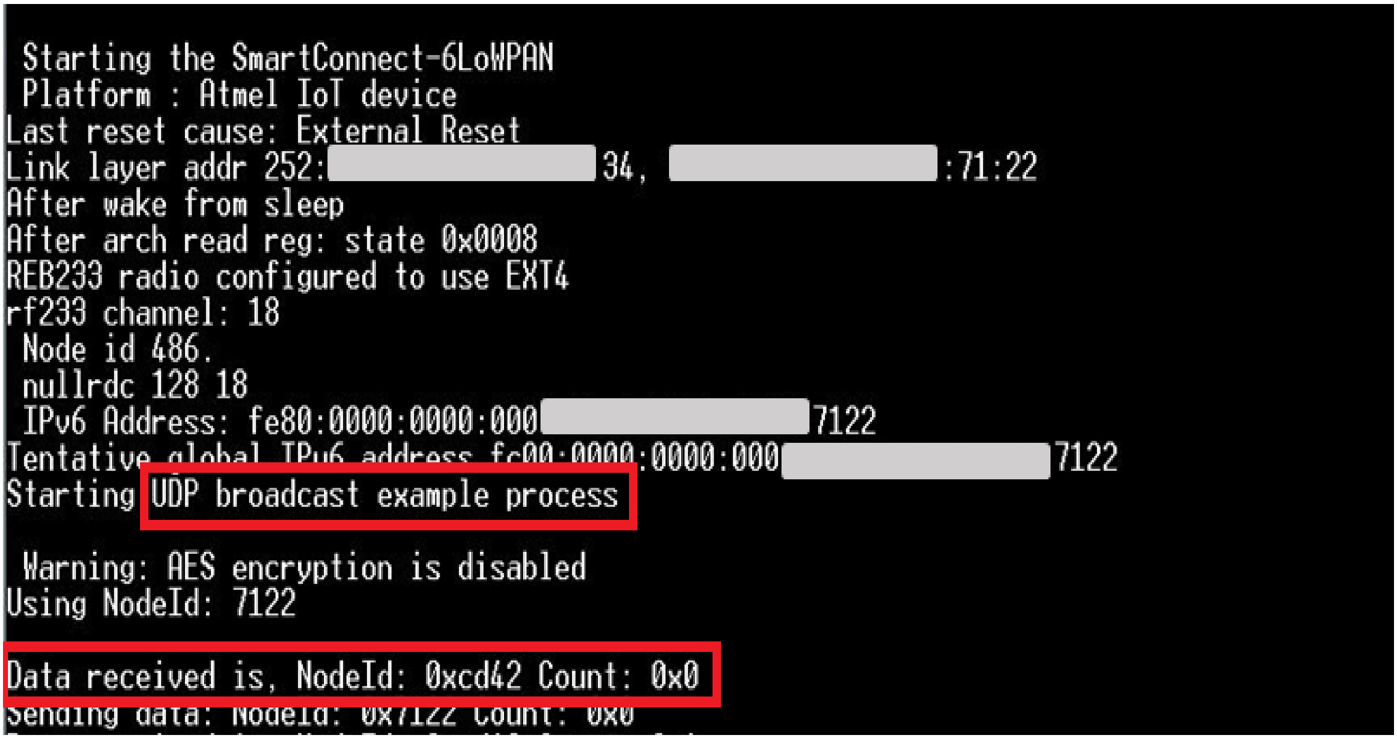
INFO

These steps are given for reference for the user. The user may vary the example application according to own needs.

6.3.4 Application execution snapshot

The following snapshot gives the view of Stack initialization, Application process getting started and working as expected.

Figure 7: UDP broadcast node serial log



```
 Starting the SmartConnect-6LoWPAN
 Platform : Atmel IoT device
Last reset cause: External Reset
Link layer addr 252:[redacted]34, [redacted]:71:22
After wake from sleep
After arch read reg: state 0x0008
REB233 radio configured to use EXT4
rf233 channel: 18
 Node id 486.
 nullrdc 128 18
 IPv6 Address: fe80:0000:0000:000[redacted]7122
Tentative global IPv6 address fc00:0000:0000:000[redacted]7122
Starting UDP broadcast example process

 Warning: AES encryption is disabled
Using NodeId: 7122

Data received is, NodeId: 0xcd42 Count: 0x0
Sending data: NodeId: 0x7122 Count: 0x0
```

6.4 udp-unicast-receiver

6.4.1 Introduction

This example application uses Simple udp module and servreg-hack module to register a UDP socket and wait on it to receive UDP uni-cast data from one or more sender devices.

This example demonstrates the ability of stack to establish UDP connection with multiple devices and successfully receive data from network.

6.4.2 Configuration

Following are configuration required to do in example application, in conjunction with Stack configuration mentioned in Section 3.2

1. UDP port has to be mentioned for which Socket will be created.

```
#define UDP_PORT                        1234
```

2. Time interval for Periodic timer has to be configured.

```
#define SEND_INTERVAL                  (5 * CLOCK_SECOND)
```

3. Server register ID

```
#define SERVICE_ID                      190
```

6.4.3 Application sequence

The sequence with which the example application executes, are given below,

Step 1: Stack initialization (done in udp-unicast-receiver-main.c)

Step 2: Start unicast_receiver_process.

Step 3: Initialize the servreg_hack module.

Step 4: Set global address using function set_global_address ().

Step 5: Make the device as root of the DAG network using function create_rpl_dag ().

Step 6: Register in servreg_hack module with defined `SERVICE_ID.`

Step7: Register UDP socket with simple_udp_sendto API.

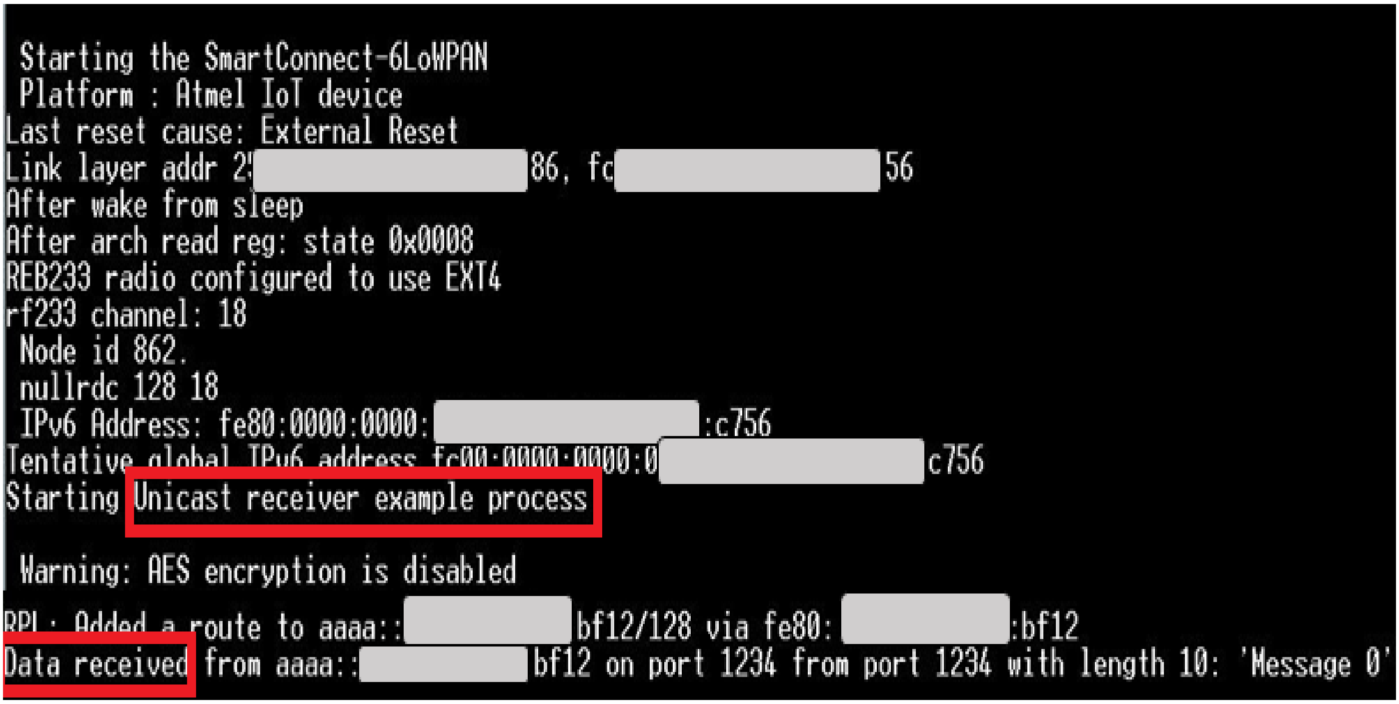
Step8: Wait for packets.

INFO **These steps are given for reference for the user. The user may vary the example application according to own needs.**

6.4.4 Application execution snapshot

The following snapshot gives the view of Stack initialization, Application process getting started and working as expected.

Figure 8: UDP Unicast receiver serial log



```
 Starting the SmartConnect-6LoWPAN
 Platform : Atmel IoT device
Last reset cause: External Reset
Link layer addr 2[redacted]86, fc[redacted]56
After wake from sleep
After arch read reg: state 0x0008
REB233 radio configured to use EXT4
rf233 channel: 18
 Node id 862.
 nullrdc 128 18
 IPv6 Address: fe80:0000:0000:[redacted]:c756
Tentative global IPv6 address fc00:0000:0000:0[redacted]c756
Starting Unicast receiver example process

 Warning: AES encryption is disabled
RPL: Added a route to aaaa::[redacted]bf12/128 via fe80:[redacted]:bf12
Data received from aaaa::[redacted]bf12 on port 1234 from port 1234 with length 10: 'Message 0'
```

6.5 udp-unicast-sender

6.5.1 Introduction

This example application uses Simple udp module and servreg-hack module to register a UDP socket and send UDP uni-cast data to UDP receiver device.

This example demonstrates the ability of stack to establish UDP connection and send UDP uni-cast data to a device.

6.5.2 Configuration

Following are configuration required to do in example application, in conjunction with Stack configuration mentioned in **Section 3.2**

1. UDP port has to be mentioned for which Socket will be created.

```
#define UDP_PORT                    1234
```

2. Time interval for Periodic timer has to be configured.

```
#define SEND_INTERVAL               (5 * CLOCK_SECOND)
```

3. Server register ID

```
#define SERVICE_ID                  190
```

6.5.3 Application sequence

The sequence with which the example application executes, are given below,

Step 1: Stack initialization (done in udp-unicast-sender-main.c)

Step 2: Start unicast_sender_process.

Step 3: Initialize the servreg_hack module.

Step 4: Set global address using function set_global_address ().

Step 5: Register UDP socket with simple_udp_sendto API.

Step 7: Start periodic timer with SEND_INTERVAL as periodic value.

Step 7: Once periodic timer expired, lookup for `SERVICE_ID` in the servreg_hack module and check any receiver device has registered with the `SERVICE_ID`. If found, get the address of the receiver device.

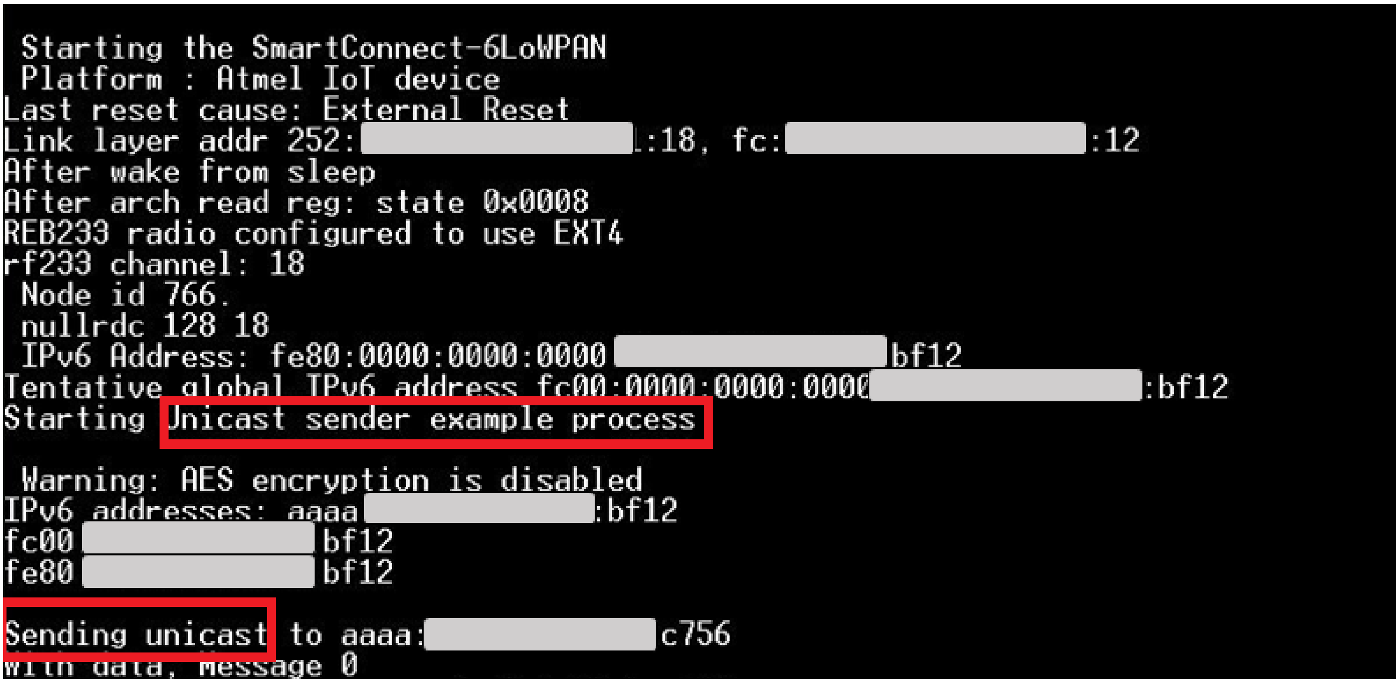
Step 8: After looked-up of global address of the receiver device, send UDP uni-cast message to the device.

ℹ **INFO** **These steps are given for reference for the user. The user may vary the example application according to own needs.**

6.5.4 Application execution snapshot

The following snapshot gives the view of Stack initialization, Application process getting started and working as expected.

Figure 9: UDP unicast sender node serial log



```
 Starting the SmartConnect-6LoWPAN
 Platform : Atmel IoT device
Last reset cause: External Reset
Link layer addr 252:[redacted]:18, fc:[redacted]:12
After wake from sleep
After arch read reg: state 0x0008
REB233 radio configured to use EXT4
rf233 channel: 18
 Node id 766.
 nullrdc 128 18
 IPv6 Address: fe80:0000:0000:0000[redacted]bf12
Tentative global IPv6 address fc00:0000:0000:000[redacted]:bf12
Starting Unicast sender example process

 Warning: AES encryption is disabled
IPv6 addresses: aaaa[redacted]:bf12
fc00[redacted]bf12
fe80[redacted]bf12

Sending unicast to aaaa:[redacted]c756
with data, message 0
```

6.6 mesh-node

6.6.1 Introduction

This example application has two options.

1. Mesh-node – Join an existing DAG network. Wait till DAG network is found.
2. Mesh-root – Create a DAG network and becomes root of the DAG network.

This example uses UIP-DS6 to get notification from core stack, when route is added for the node.

This example demonstrates the ability of stack to create DAG network, join the existing network and get notifications from UIP module.

6.6.2 Configuration

Following are configuration required to do in example application, in conjunction with Stack configuration mentioned in Section 3.2

1. Define MESH_ NODE = 1 if the application has to behave as a node in the network.
2. Define MESH_NODE = 0 if the application has to be the root of the network.

TIP **To define MESH_NODE in go to Project>> Properties >> Toolchain >> Symbols and change the value of the MESH_NODE. To know more refer Atmel_Studio >> help**

6.6.3 Application sequence

The sequence with which the example application executes, are given below,

6.6.3.1 Mesh-node

Step 1: Stack initialization (done mesh-node-main.c)

Step 2: Start mesh-node process

Step 3: Switch ON the LED.

Step 4: Add the call back function in the UIP-DS6 notification list using uip_ds6_notification_add (). So when route added for the node, UIP module will indicate the application by the calling the registered callback function.

Step 5: Once the callback function is called after adding a route, switch off the LED and print the route details.

6.6.3.2 Mesh-root

Step 1: Stack initialization (done in mesh-node-main.c)

Step 2: Start mesh-root process

Step 3: Create the DAG root using simple_rpl_init_dag () function.

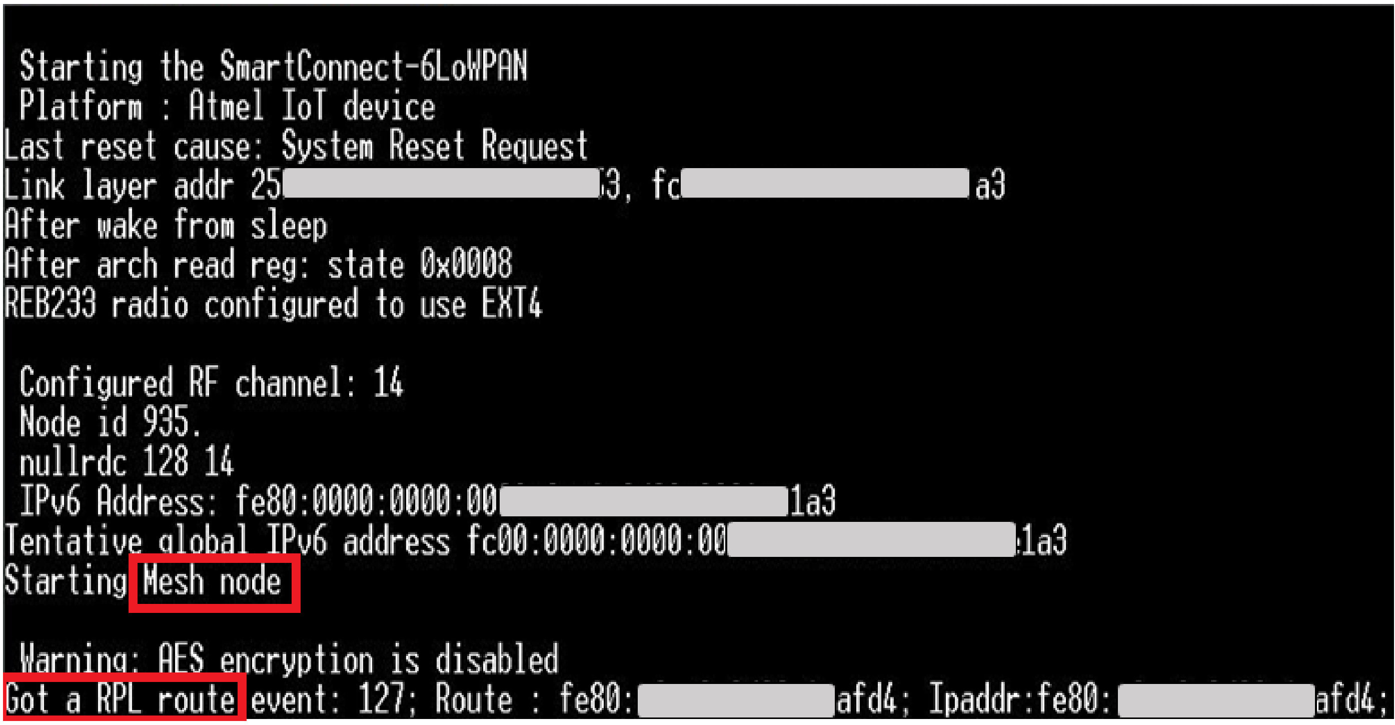
INFO

These steps are given for reference for the user. The user may vary the example application according to own needs.

6.6.4 Application execution snapshot

The following snapshot gives the view of Stack initialization, Application process getting started and working as expected.

Figure 10: Mesh node serial log



```
 Starting the SmartConnect-6LoWPAN
 Platform : Atmel IoT device
Last reset cause: System Reset Request
Link layer addr 25[redacted]3, fc[redacted]a3
After wake from sleep
After arch read reg: state 0x0008
REB233 radio configured to use EXT4

 Configured RF channel: 14
 Node id 935.
 nullrdc 128 14
 IPv6 Address: fe80:0000:0000:00[redacted]1a3
Tentative global IPv6 address fc00:0000:0000:00[redacted]1a3
Starting Mesh node

 Warning: AES encryption is disabled
Got a RPL route event: 127; Route : fe80:[redacted]afd4; Ipaddr:fe80:[redacted]afd4;
```

6.7 br-node

6.7.1 Introduction

Border Router example application uses IP64 module to communicate outside 6LoWPAN network, DHCP module to get an IPv4 address and simple RPL module to create RPL DAG root

This example demonstrates the ability of stack to create a DAG root and act as Border gateway to the nodes which wants to communicate outside the network.

This application uses IP64 module which converts IPv6 address of a node to IPv4 address and creates an entry in IP64 table and maintains it.

This Application uses DHCPv4 module to get an IPv4 address from outside the network.

ℹ INFO **To know more about IP64 and DHCPv4, please refer Section 2.1.6**

6.7.2 Application setup

6.7.2.1 SAMR21 setup

SAMR21_xpro boards – 4

Ethernet1_xpro board - 1

Step 1: Compile and flash br-webserver-node to one of SAMR21_xpro + Ethernet1_xpro board.

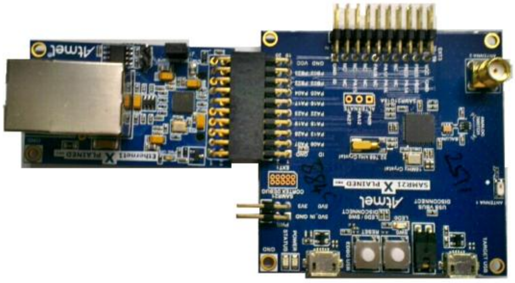
Step 2: Compile and flash MQTT examples to 3 SAMR21_xpro boards.

Step 3: Once DHCP is configured, MQTT device can communicate to configured MQTT server. Refer **section 6.9** to know about MQTT device configuration.

6.7.2.2 Usage of Ethernet1_xpro board

Connect Ethernet1_xpro board in EXT1 of SAMR21 board as shown below and connect Ethernet cable to Ethernet1_xpro board.

Figure 11: Border router node (SAMR21_xpro + Ethernet1_xpro)



6.7.3 Configuration

Following are configuration required to do in example application, in conjunction with Stack configuration mentioned in **Section 3.2**

6.7.4 Application sequence

The sequence with which the example application executes, are given below,

Step 1: Stack initialization (done in br-node-main.c)

Step 2: Start router_node_process.

Step 3: Call simple_rpl_init_dag () function to setup DAG network and become root of it.

Step4: Call ip64_init () function, to initialize IP64 module.

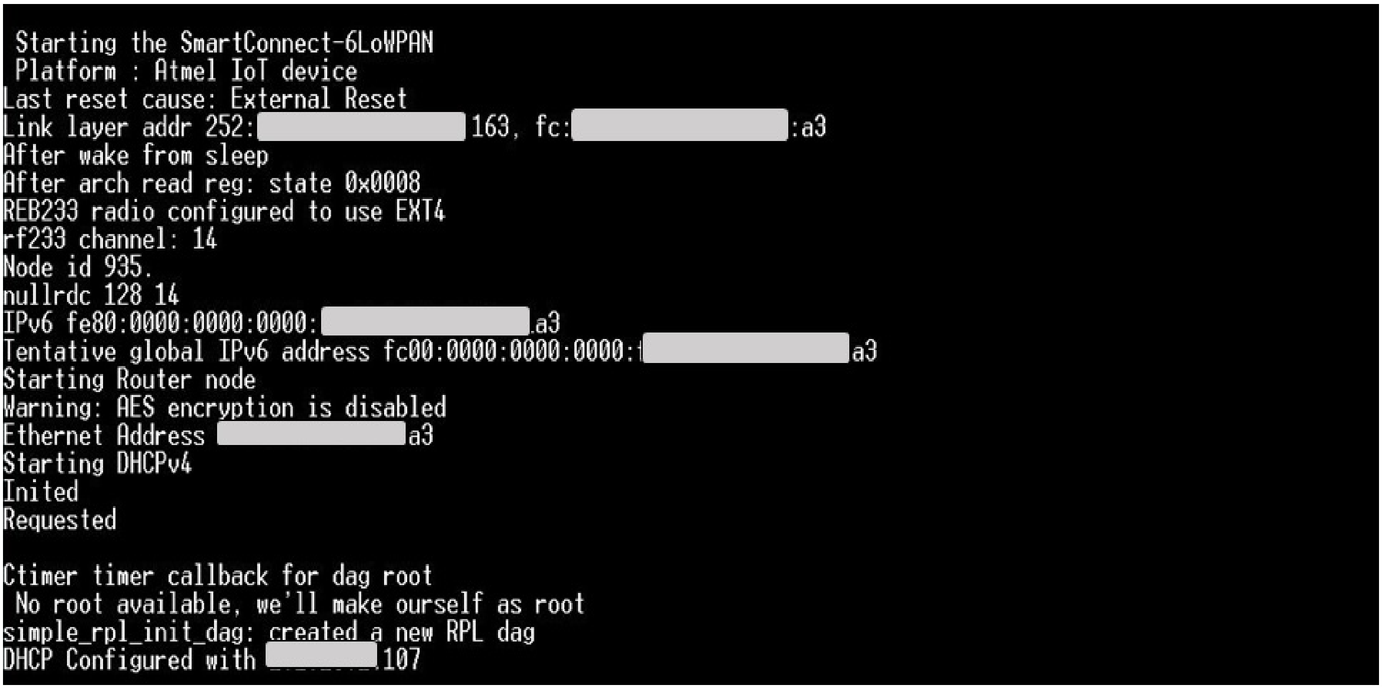
Step5: The IP64 module in turn will call DHCPv4 process and get dynamic IPv4 address.

INFO **These steps are given for reference for the user. The user may vary the example application according to own needs.**

6.7.5 Application execution snapshot

The following snapshot gives the view of Stack initialization, Application process getting started and working as expected.

Figure 12: Border Router serial log



```
 Starting the SmartConnect-6LoWPAN
 Platform : Atmel IoT device
Last reset cause: External Reset
Link layer addr 252:[redacted]163, fc:[redacted]:a3
After wake from sleep
After arch read reg: state 0x0008
REB233 radio configured to use EXT4
rf233 channel: 14
Node id 935.
nullrdc 128 14
IPv6 fe80:0000:0000:0000:[redacted].a3
Tentative global IPv6 address fc00:0000:0000:0000:[redacted]a3
Starting Router node
Warning: AES encryption is disabled
Ethernet Address [redacted]a3
Starting DHCPv4
Inited
Requested

Ctimer timer callback for dag root
 No root available, we'll make ourself as root
simple_rpl_init_dag: created a new RPL dag
DHCP Configured with [redacted].107
```

6.8 br-webserver-node

6.8.1 Introduction

Border Router Web-server example application uses IP64 module to communicate outside 6LoWPAN network, DHCP module to get an IPv4 address and simple RPL module to create RPL DAG root.

This example also uses IP64 Web serve module, to create HTML pages views of configured information which can be viewed from a browser outside the network. Such an example of browser is shown in Application snapshot section.

This example demonstrates the ability of stack to create a DAG root and act as Border gateway to the nodes which wants to communicate outside the network.

This application uses IP64 module which converts IPv6 address of a node to IPv4 address and creates an entry in IP64 table and maintains it.

This Application uses DHCPv4 module to get an IPv4 address from outside the network.

INFO **To know more about IP64 and DHCPv4, please refer Section 2.1.6**

6.8.2 Application setup

6.8.2.1 SAMR21 setup

SAMR21_xpro boards – 4

Ethernet1_xpro board - 1

Step 1: Compile and flash br-webserver-node to one of SAMR21_xpro + Ethernet1_xpro board.

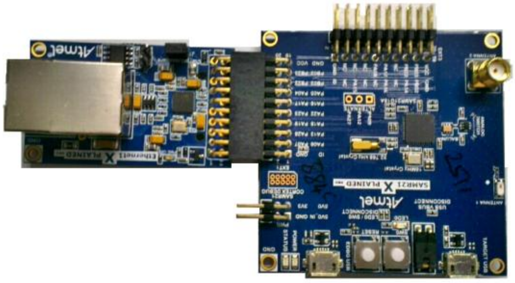
Step 2: Compile and flash mesh-node examples to 3 SAMR21_xpro boards.

Step 3: Once DHCP is configured, view the mesh-node information from browser.

6.8.2.2 Usage of Ethernet1_xpro board

Connect Ethernet1_xpro board in EXT1 of SAMR21 board as shown below and connect Ethernet cable to Ethernet1_xpro board.

Figure 13: Border router web-server node (SAMR21_xpro + Ethernet1_xpro)



6.8.3 Configuration

Following are configuration required to do in example application, in conjunction with Stack configuration mentioned in Section 3.2

6.8.4 Application sequence

The sequence with which the example application executes, are given below,

Step 1: Stack initialization (done in br-node-main.c)

Step 2: Start router_node_process.

Step 3: Call simple_rpl_init_dag () function to setup DAG network and become root of it.

Step 4: Call ip64_init () function, to initialize IP64 module.

Step 5: Call ip64_webserver_init () function to initialize the IP64-WEBSEVER module and create default HTML pages.

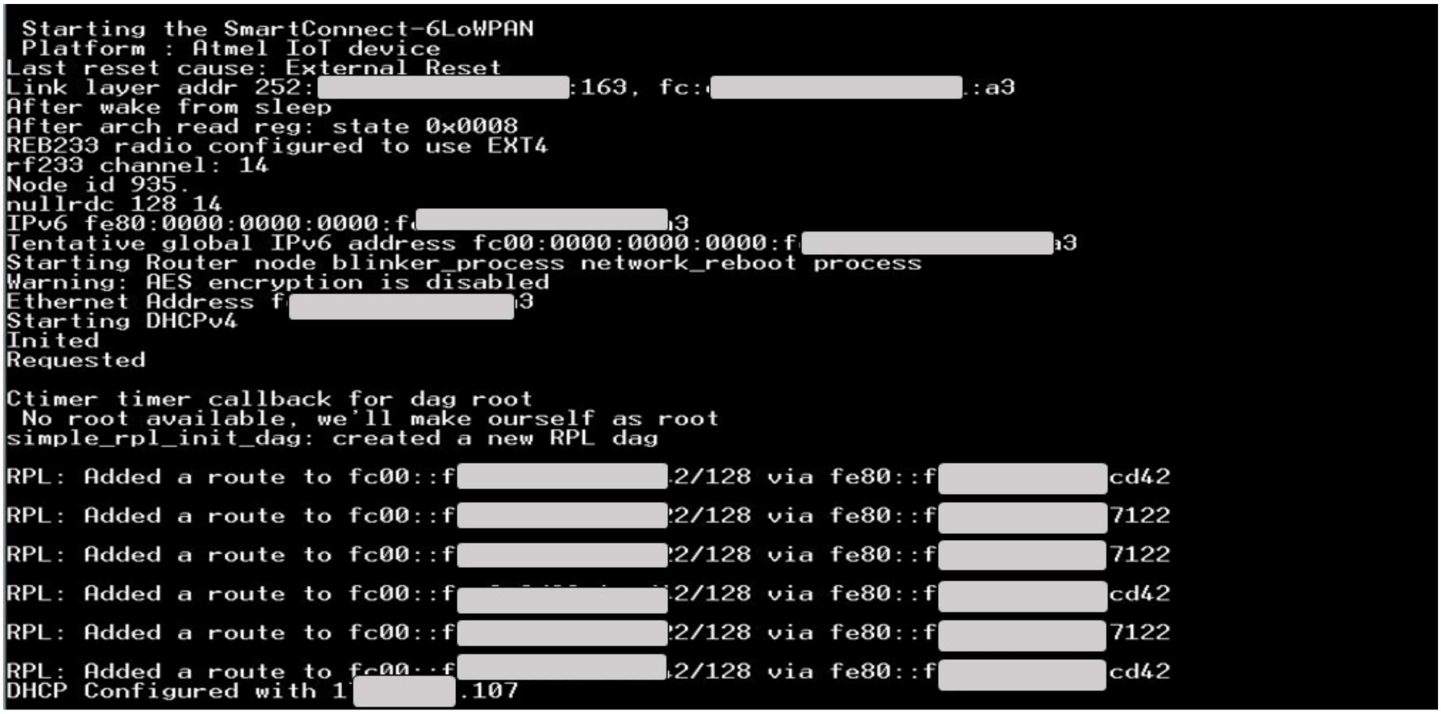
Step 6: The IP64 module in turn will call DHCPv4 process and get dynamic IPv4 address.

INFO **These steps are given for reference for the user. The user may vary the example application according to own needs.**

6.8.5 Application execution snapshot

The following snapshot gives the view of Stack initialization, Application process getting started and working as expected.

Figure 14: Border router web server serial log



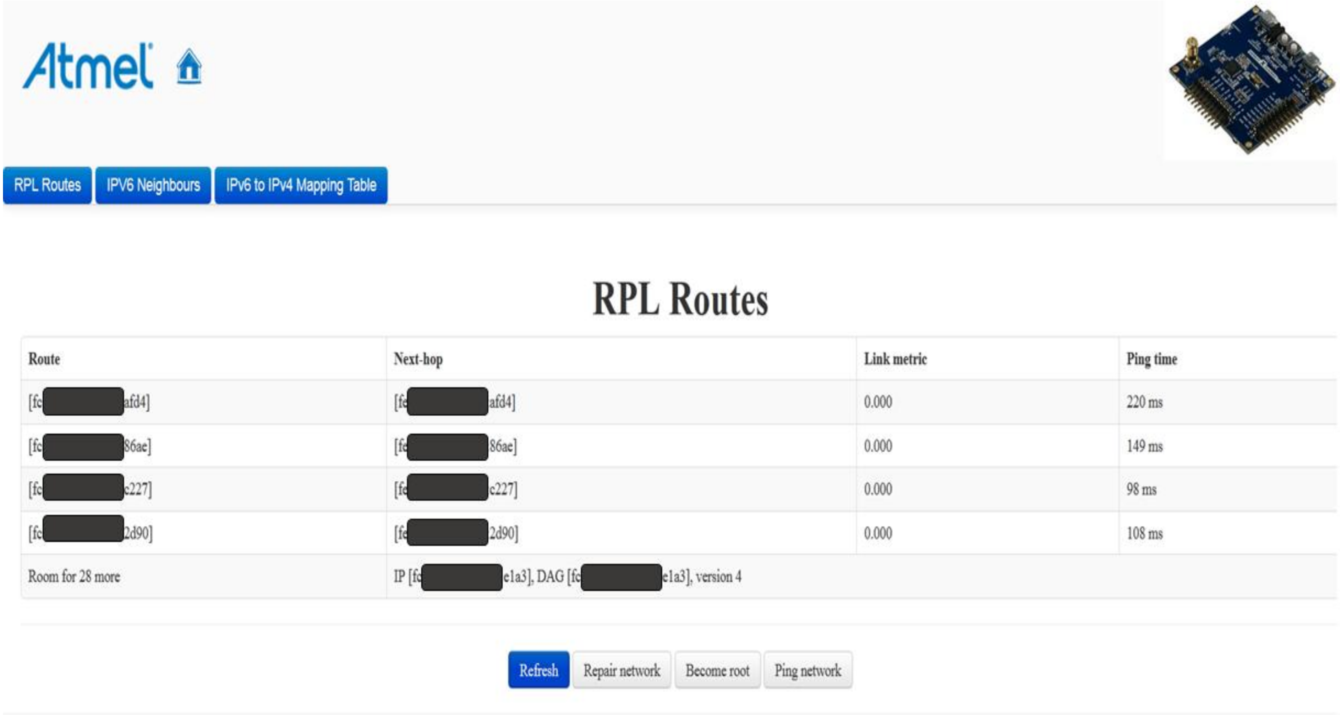
```
 Starting the SmartConnect-6LoWPAN
 Platform : Atmel IoT device
Last reset cause: External Reset
Link layer addr 252:[redacted]:163, fc:[redacted].:a3
After wake from sleep
After arch read reg: state 0x0008
REB233 radio configured to use EXT4
rf233 channel: 14
Node id 935.
nullrdc 128 14
IPv6 fe80:0000:0000:0000:f[redacted]3
Tentative global IPv6 address fc00:0000:0000:0000:f[redacted]3
Starting Router node blinker_process network_reboot process
Warning: AES encryption is disabled
Ethernet Address f[redacted]3
Starting DHCPv4
Inited
Requested

Ctimer timer callback for dag root
 No root available, we'll make ourself as root
simple_rpl_init_dag: created a new RPL dag

RPL: Added a route to fc00::f[redacted]2/128 via fe80::f[redacted]cd42

RPL: Added a route to fc00::f[redacted]2/128 via fe80::f[redacted]7122

RPL: Added a route to fc00::f[redacted]2/128 via fe80::f[redacted]7122

RPL: Added a route to fc00::f[redacted]2/128 via fe80::f[redacted]cd42

RPL: Added a route to fc00::f[redacted]2/128 via fe80::f[redacted]7122

RPL: Added a route to fc00::f[redacted]2/128 via fe80::f[redacted]cd42
DHCP Configured with 1[redacted].107
```

Once the DHCP is configured with IPv4 address, user can view the HTML pages from web server from a PC connected inside the LAN network. Type the IPv4 address shown in the Tera Term in the Web browser to view the information about the nodes.

This web-server has a feature to view the devices inside the 6LoWPAN network and ping them individually.

Additionally, Neighbor table and IP64 table also can be viewed from Web-Server.

Figure 15: Border Router Web Page



RPL Routes

Route	Next-hop	Link metric	Ping time
[fc[redacted]afd4]	[fe[redacted]afd4]	0.000	220 ms
[fc[redacted]86ae]	[fe[redacted]86ae]	0.000	149 ms
[fc[redacted]c227]	[fe[redacted]c227]	0.000	98 ms
[fc[redacted]2d90]	[fe[redacted]2d90]	0.000	108 ms
Room for 28 more	IP [fc[redacted]e1a3], DAG [fc[redacted]e1a3], version 4		

6.9 mqtt-example

6.9.1 Introduction

MQTT example application uses MQTT and MDNS modules to communicate to MQTT host configured and establish a MQTT connection.

This example,

1. Will publish data to the MQTT host broker configured with configured topic.
2. Uses IO1_xpro board, to get Temperature and Light sensor details of the current environment and send them to MQTT broker in JSON format.
3. Uses Border router to connect outside the 6LoWPAN network.
4. Uses MDNS module, to get IP address of MQTT host broker configured. MDNS server used here, is Google DNS server (::ffff:8.8.8.8).

INFO **To know more about MDNS, please refer Section 2.1.6**

6.9.2 Application setup

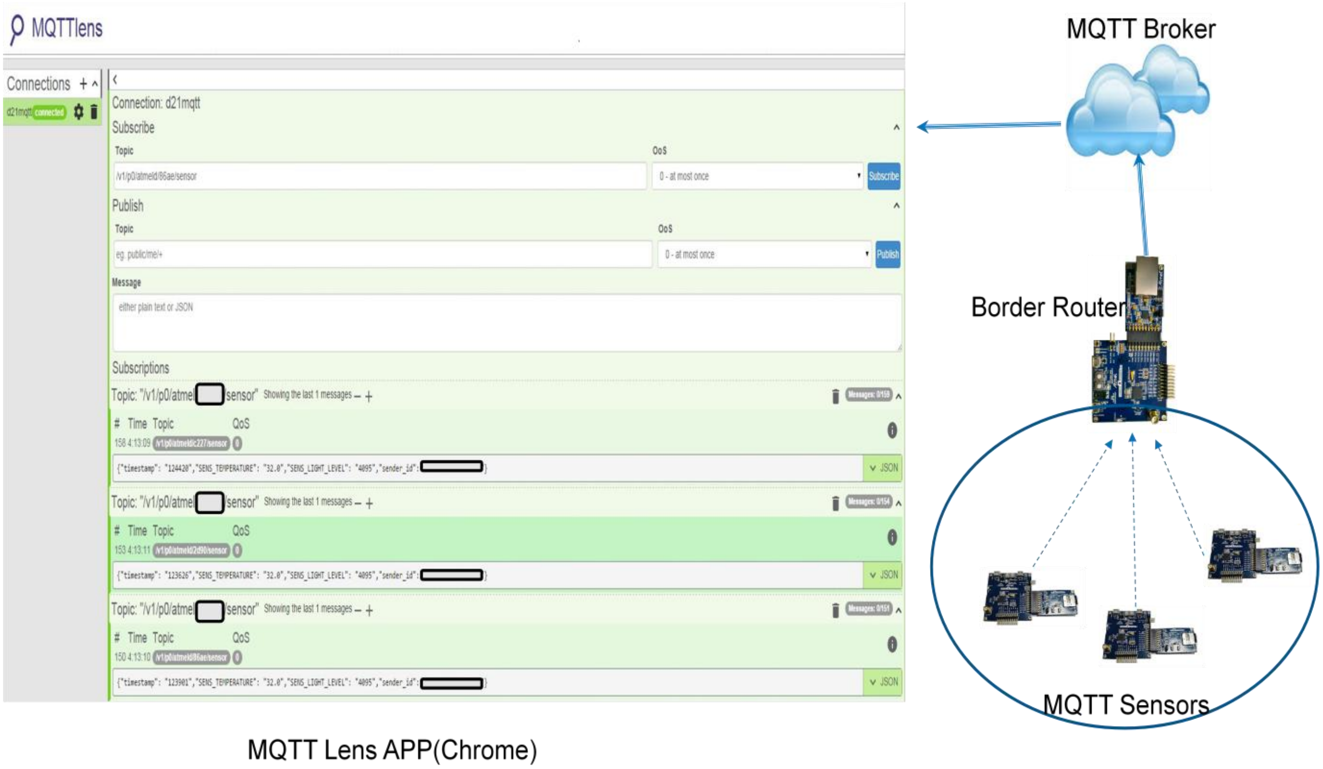
6.9.2.1 SAMR21 setup

Refer **section 6.7** for the SAMR21 Border Router setup details.

Connect IO1_xpro board to SAMR21 board. Compile and flash MQTT example code in SMR21 board.

Example setup looks like given below,

Figure 16: MQTT setup



MQTTlens

Connections + ^

d21mqtt connected

Connection: d21mqtt

Subscribe

Topic: /v1/p0/atmeld/86ae/sensor QoS: 0 - at most once Subscribe

Publish

Topic: eg. public/me/+ QoS: 0 - at most once Publish

Message: either plain text or JSON

Subscriptions

Topic: "/v1/p0/atmel[redacted]sensor" Showing the last 1 messages – + Messages: 0/159

```
#   Time    Topic                       QoS
158 4:13:09 /v1/p0/atmeld/c227/sensor   0
{"timestamp": "124420","SENS_TEMPERATURE": "32.0","SENS_LIGHT_LEVEL": "4095","sender_id":[redacted]}
```

Topic: "/v1/p0/atmel[redacted]/sensor" Showing the last 1 messages – + Messages: 0/154

```
#   Time    Topic                       QoS
153 4:13:11 /v1/p0/atmeld/2d90/sensor   0
{"timestamp": "123626","SENS_TEMPERATURE": "32.0","SENS_LIGHT_LEVEL": "4095","sender_id":[redacted]}
```

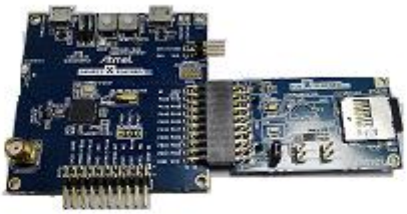
Topic: "/v1/p0/atmel[redacted]/sensor" Showing the last 1 messages – + Messages: 0/151

```
#   Time    Topic                       QoS
150 4:13:10 /v1/p0/atmeld/86ae/sensor   0
{"timestamp": "123901","SENS_TEMPERATURE": "32.0","SENS_LIGHT_LEVEL": "4095","sender_id":[redacted]}
```

6.9.2.2 Usage of IO1_xpro board

Connect IO1_xpro board in EXT1 of SAMR21 board as shown.

Figure 17: MQTT Sensor (SAMR21_xpro + IO1_xpro)



6.9.3 Configuration

Following are configuration required to do in example application, in conjunction with Stack configuration mentioned in Section 3.2

In mqtt-example.c, following macros has to be modified in each device for a MQTT Application to run in that specific device.

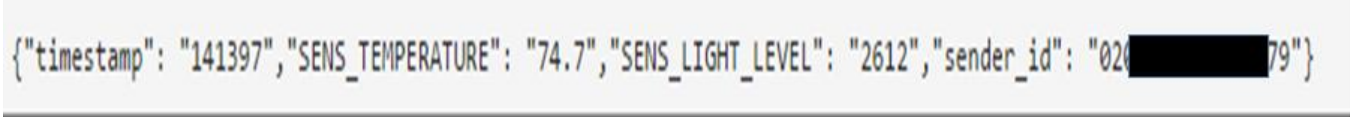
```
/* MQTT Configuration details */
#define HOST           "m2m.eclipse.org"
#define VERSION        "v1"
#define PRIORITY       "p0"
#define UUID           "atmeld"
```

INFO **The UUID will be the MQTT Client name and it'll register with MQTT Host Broker with same name.**

6.9.3.1 MQTT Client Configuration

The topics of MQTT Client are currently defined as follows.

1. `/<version>/<priority>/<UUID>/sensor` - topic for subscribe. This is the topic, we need specify in MQTT app (MQTT Lens for browser or MY MQTT android app) to get sensor data from each MQTT device.
2. Currently both Light and Temperature sensor data are integrated (IO1 Xpro board).
3. All sensor data are sent in JSON format as seen below.



```
{"timestamp": "141397","SENS_TEMPERATURE": "74.7","SENS_LIGHT_LEVEL": "2612","sender_id": "02[redacted]79"}
```

4. `/<version>/<priority>/<UUID>/led` – topic for publishing to client. Currently to publish data to MQTT device, use the above topic and send data as “on” or “off” to make LED ON/OFF.

6.9.4 Application sequence

The sequence with which the example application executes, are given below,

Step 1: Stack initialization (done in mqtt-example-main.c)

Step 2: Start mqtt_example_process.

Step 3: Call uip_ds6_get_link_local () to get link local address, which sent in Publish data.

Step 4: Configure topics based on configured details and set periodic timer to publish MQTT data.

Step 5: Resolve the MQTT host broker address using MDNS module.

Step 6: Register MQTT connection with given `UUID`.

Step 7: Initialize IO1_xpro board to get Temperature and Light sensor details.

Step 8: Find RPL parent and add route to it using simple_rpl_parent ().

Step 9: Connect to MQTT broker using mqtt_connect () API.

Step 10: Once MQTT connection is established, Publish Status Topic stating the device is “online” and Subscribe to the LED topic.

Step 11: Once periodic timer is expired, Collect Temperature and Light sensor details from IO1_xpro board.

Step 12: Put the collected details, timestamp and MAC address of the device in JSON format.

Step 13: Send the JSON formatted data to MQTT broker.

Step 14: Repeat from **Step 11** for every periodic timer expiry.

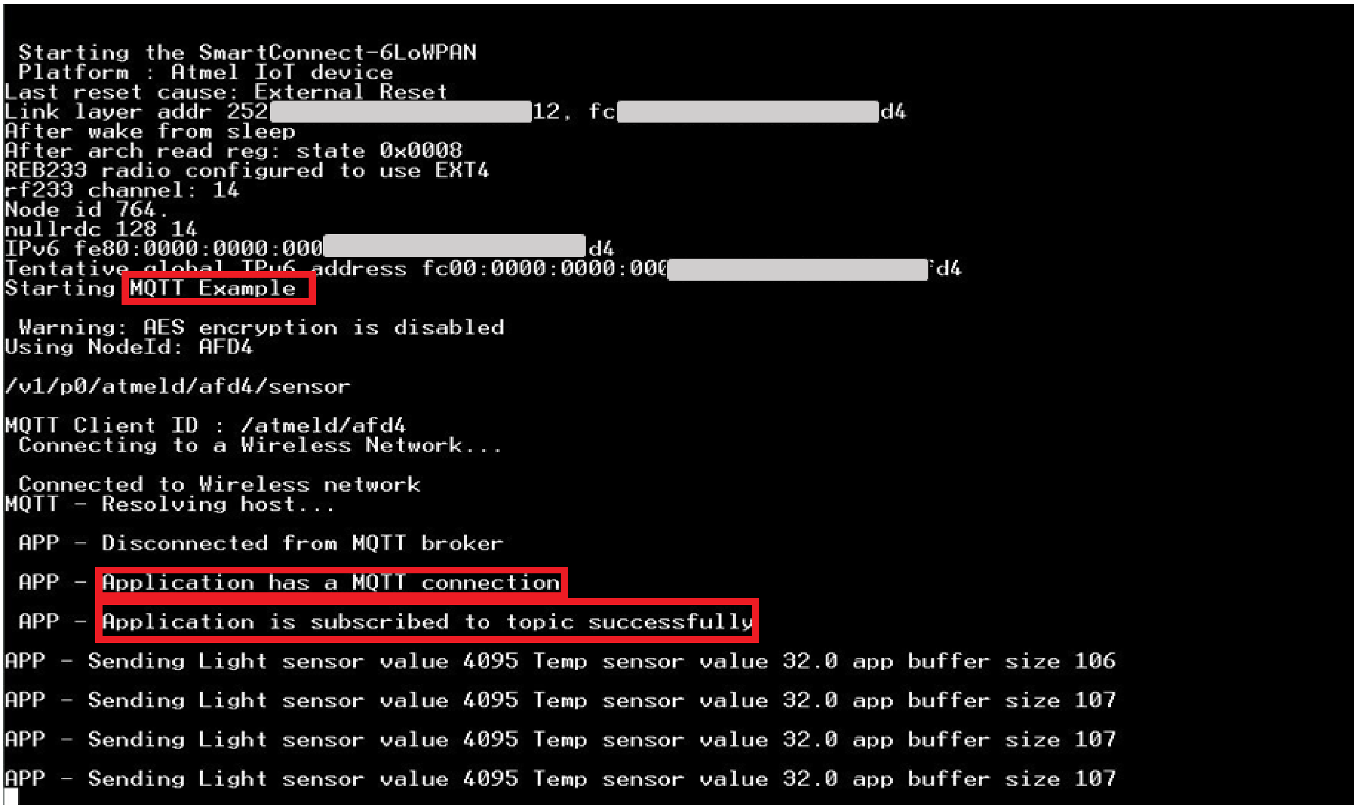
INFO **These steps are given for reference for the user. The user may vary the example application according to own needs.**

6.9.5 Application execution snapshot

6.9.5.1 Device serial snapshot

The following snapshot gives the view of Stack initialization, Application process getting started and working as expected.

Figure 18: MQTT Sensor node serial log screen shot



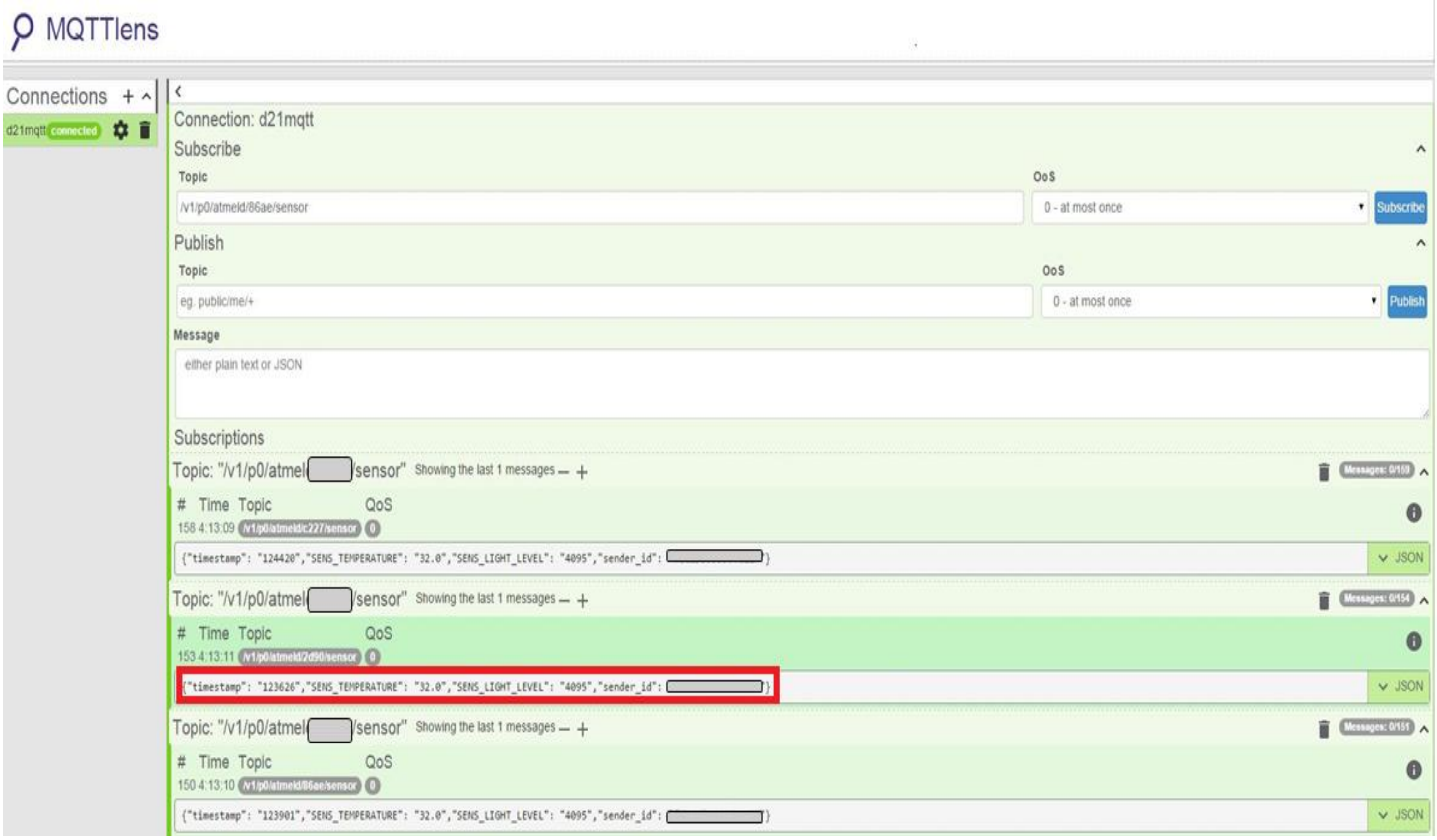
```
 Starting the SmartConnect-6LoWPAN
 Platform : Atmel IoT device
Last reset cause: External Reset
Link layer addr 252[redacted]12, fc[redacted]d4
After wake from sleep
After arch read reg: state 0x0008
REB233 radio configured to use EXT4
rf233 channel: 14
Node id 764.
nullrdc 128 14
IPv6 fe80:0000:0000:000[redacted]d4
Tentative global IPv6 address fc00:0000:0000:000[redacted]d4
Starting MQTT Example

 Warning: AES encryption is disabled
Using NodeId: AFD4

/v1/p0/atmeld/afd4/sensor

MQTT Client ID : /atmeld/afd4
 Connecting to a Wireless Network...

 Connected to Wireless network
MQTT - Resolving host...

 APP - Disconnected from MQTT broker

 APP - Application has a MQTT connection

 APP - Application is subscribed to topic successfully

APP - Sending Light sensor value 4095 Temp sensor value 32.0 app buffer size 106

APP - Sending Light sensor value 4095 Temp sensor value 32.0 app buffer size 107

APP - Sending Light sensor value 4095 Temp sensor value 32.0 app buffer size 107

APP - Sending Light sensor value 4095 Temp sensor value 32.0 app buffer size 107
```

6.9.5.2 MQTT lens screenshot.

1. Install MQTT lens in Chrome browser.

2. Establish a connection to the MQTT host broker.
3. After connected to MQTT broker, subscribe for the topic configured in each device in the setup. For eg: in above screen shot, the topic is `/v1/p0/atmeld/afd4/sensor.`
4. From the MQTT lens Application, you can each device published data in JSON format.

Figure 19: MQTT subscription logs using MQTT lens



Subscribe — Topic: /v1/p0/atmeld/86ae/sensor — QoS: 0 - at most once

Publish — Topic: eg. public/me/+ — QoS: 0 - at most once

Message: either plain text or JSON

Topic: "/v1/p0/atmel[redacted]/sensor" Showing the last 1 messages — + Messages: 0/159

#	Time	Topic	QoS
158	4:13:09	/v1/p0/atmeld/c227/sensor	0

```
{"timestamp": "124420","SENS_TEMPERATURE": "32.0","SENS_LIGHT_LEVEL": "4095","sender_id": [redacted]}
```

Topic: "/v1/p0/atmel[redacted]/sensor" Showing the last 1 messages — + Messages: 0/154

#	Time	Topic	QoS
153	4:13:11	/v1/p0/atmeld/2d90/sensor	0

```
{"timestamp": "123626","SENS_TEMPERATURE": "32.0","SENS_LIGHT_LEVEL": "4095","sender_id": [redacted]}
```

Topic: "/v1/p0/atmel[redacted]/sensor" Showing the last 1 messages — + Messages: 0/151

#	Time	Topic	QoS
150	4:13:10	/v1/p0/atmeld/86ae/sensor	0

```
{"timestamp": "123901","SENS_TEMPERATURE": "32.0","SENS_LIGHT_LEVEL": "4095","sender_id": [redacted]}
```

7 Tool chain

7.1 General prerequisites

The following tools and tool-chains are used for building the applications from this MAC package:

- Atmel Studio 6.2
  (see http://www.atmel.com/tools/ATMELSTUDIO.aspx)
- ARM Code Sorcery GCC Toolchain for Windows(IA32 Windows Installer)
  (see http://www.codesourcery.com/sgpp/lite/arm/portal/release642)

7.2 Building applications

7.2.1 Using GCC makefiles

Each application should be built using the provided Makefiles. Please follow the procedure as described:

- Change to the directory where the Makefile for the desired platform of the corresponding application is located, for example:

```
cd D:\ASF\thirdparty\wireless\SmartConnect_6LoWPAN\examples\hello-world
cd samr21_xpro\gcc
```

- Run the desired Makefile, for example:

```
make –f Makefile
```

Makefile builds a binary optimized for code size without Serial I/O support, whereas Makefile_Debug builds a version for better debug support without optimization but with additional Serial I/O support

- After running one of the Makefiles the same directory contains both a hex-file and an elf-file which can be downloaded onto the hardware (see Section **Error! Reference source not found.**)
- The above procedure for building the Makefiles is common for all platforms.

8 Supported Platforms

This chapter describes which hardware platforms are currently supported with the Atmel AVR2025 MAC software package. A platform usually comprises of three major components:

- An MCU,
- A transceiver chip (this may be integrated into the MCU for Single Chips)
- A specific Board or even several boards that contain the MCU or the transceiver chip

8.1.1 Supported MCU families

Currently the following generic MCU families are supported:

- SAM0: ARM Cortex-M0 platforms

8.1.2 Supported transceivers

Currently the following transceivers are supported:

- AT86RF233 – 2.4GHz transceiver
- AT86RF212B – sub-GHZ transceiver

8.1.3 Supported boards

Few of the currently supported boards and combinations are given below.

- samr21_xpro
- samd21_xpro

9 Platform porting

For details and description about platform porting please refer to Atmel Software Framework documentation [5] and Atmel Studio-help [6].

10 Abbreviations

6LoWPAN – IPv6 packets over Wireless Personal Area Network

ASF – Atmel Software Framework

AES – Advanced Encryption Standard

CoAP – Constrained Application Protocol

CSMA – Carrier Sense Multiple Access

DHCP – Dynamic Host Control Protocol

HTTP – Hypertext Transfer Protocol

IPHC – IP packet Header Compression

IPv6 – IP version 6

IP – Internet Protocol

MAC – Medium Access Control

MCU – Micro Controller

MDNS – Multicast Domain Name System

MQTT – Message Queue Telemetry Transport

RDC – Radio Duty Cycle

RPL – Routing Protocol for Lossy and Low power Networks

UDP – User Datagram Protocol

TCP – Transmission Control Protocol

11 Reference

2. Atmel Wireless MCU Software Website
   http://www.atmel.com/products/microcontrollers/wireless/default.aspx?tab=tools
3. Atmel Wireless Support avr@atmel.com
4. IEEE Std 802.15.4™-2006 Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)
5. Atmel documents for supported families and boards : asf.atmel.com/docs/latest
6. Atmel Software Framework http://www.atmel.com/tools/avrsoftwareframework.aspx
   http://asf.atmel.com/docs/latest/
7. Atmel Studio - http://www.atmel.com/tools/ATMELSTUDIO.aspx
8. IETF RPL RFC 6550 - https://tools.ietf.org/html/rfc6550
9. IETF Transmission of IPv6 packets over IEEE 802.15.4 networks RFC 4944 -
   https://tools.ietf.org/html/rfc4944

ATMEL EVALUATION BOARD/KIT IMPORTANT NOTICE AND DISCLAIMER

This evaluation board/kit is intended for user's internal development and evaluation purposes only. It is not a finished product and may not comply with technical or legal requirements that are applicable to finished products, including, without limitation, directives or regulations relating to electromagnetic compatibility, recycling (WEEE), FCC, CE or UL. Atmel is providing this evaluation board/kit “AS IS” without any warranties or indemnities. The user assumes all responsibility and liability for handling and use of the evaluation board/kit including, without limitation, the responsibility to take any and all appropriate precautions with regard to electrostatic discharge and other technical issues. User indemnifies Atmel from any claim arising from user's handling or use of this evaluation board/kit. Except for the limited purpose of internal development and evaluation as specified above, no license, express or implied, by estoppel or otherwise, to any Atmel intellectual property right is granted hereunder. ATMEL SHALL NOT BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMGES RELATING TO USE OF THIS EVALUATION BOARD/KIT.

ATMEL CORPORATION
1600 Technology Drive
San Jose, CA 95110
USA

Revision History

Doc Rev.	Date	Comments
0.1	02/2015	Initial document release.

Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**

Atmel Confidential: For Release Only Under Non-Disclosure Agreement (NDA)

Atmel®, Atmel logo and combinations thereof, Enabling Unlimited Possibilities®, and others are registered trademarks or trademarks of Atmel Corporation in U.S. and other countries. ARM®, ARM Connected® logo, and others are the registered trademarks or trademarks of ARM Ltd. Other terms and product names may be trademarks of others.

DISCLAIMER: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

SAFETY-CRITICAL, MILITARY, AND AUTOMOTIVE APPLICATIONS DISCLAIMER: Atmel products are not designed for and will not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death ("Safety-Critical Applications") without an Atmel officer's specific written consent. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Atmel products are not designed nor intended for use in military or aerospace applications or environments unless specifically designated by Atmel as military-grade. Atmel products are not designed nor intended for use in automotive applications unless specifically designated by Atmel as automotive-grade.