Radiocrafts
Embedded Wireless Solutions

User Manual

RIIM SDK
Radiocrafts Industrial IP Mesh - Software Development Kit
Rev. 3.1.0

Table of Tables

Table of Figures

Table of Examples

Abbreviations

Abbreviation	Description
ADC	Analog-to-Digital Converter
API	Application Programming Interface
CoAP	Constrained Application Protocol
DTLS	Datagram Transport Layer Security
GPIO	General Purpose Input/Output
I^2C	Inter-Integrated Circuit
ICI	Intelligent C-Programmable I/O
LQI	Link Quality Indicator
MAC	Media Access Control
OSI	Open Systems Interconnection
PAN	Personal Area Network
PHY	Physical Layer of the OSI model
RF	Radio Frequency
RIIM	Radiocrafts Industrial IP Mesh
RSSI	Received Signal Strength Indicator
SDK	Software Development Kit
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol
WSN	Wireless Sensor Network
TSCH	Time-Slotted Channel Hopping
SC	Single Channel

1 Introduction RIIM

The RIIM network consists of these key elements

- The RIIM SDK
  - Software development kit with ICI application frameworks and tools for creating and uploading end ICI applications to the RC1882-IPM
- The RC1882-IPM module
  - The RC1882-IPM module can be configured as Border Router node, Mesh Router node or Leaf node.
    - As a Border Router it acts as the base of the mesh network. It can connect to an external network via ethernet or custom user ICI application on other interfaces such as UART
    - As a Mesh Router, it will be able to transport packets in the RIIM mesh network
    - As a Leaf, it is not able to transport packets to other nodes except its parent. This mode uses the least amount of energy.
  - All node configurations require an ICI application for RF and interface configuration and the user application. The same RIIM Software Development Kit (SDK) is used to create the ICI application for all node configurations.

Below is an illustration of the different elements and the documentation available

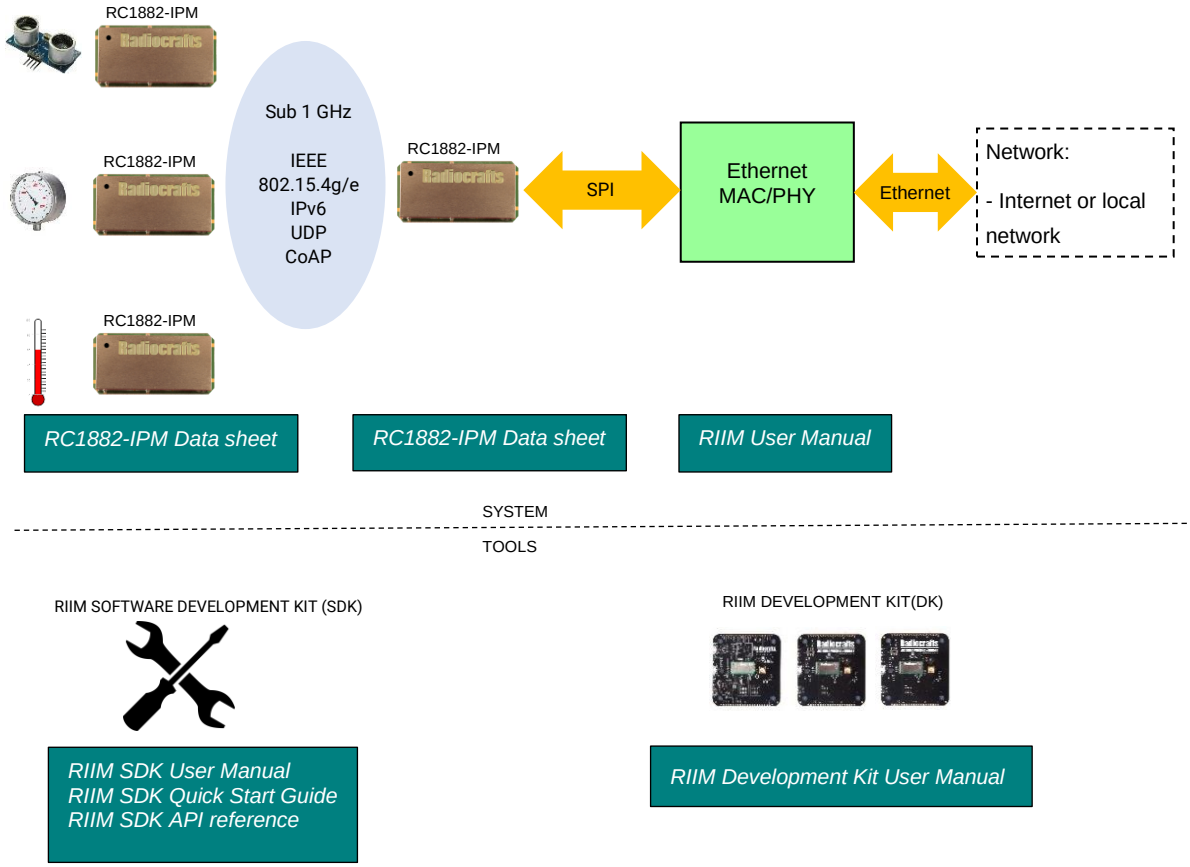

Figure 1. RIIM network – system and documentation overview

2 Introduction

This user manual shows how a custom ICI application for wireless sensors or actuators can be quickly and easily developed on the Radiocrafts RIIM (Radiocrafts Industrial IP Mesh). An ICI application is always running on the module on top of the Platform Image, to tailor the modules behaviour to the customers unique requirements.

The ICI application is written in a high-level C-language, using a powerful API that is available in the SDK. The API removes the need for the developer to understand the underlying architecture and resources in the module.
In its simplest form, the ICI application is just configuring the radio network, the modules hardware interfaces and defining when to read and write to those interfaces. This can typically be done with less than 100 lines of code and within a few hours. Examples included in the SDK are normally a good starting point.
The ICI application also have the capability of including complex data processing and advanced features, such as averaging and threshold detection using one or many sensors in combination or to create complex sensor interfaces. The flash space available for the ICI application is 32 kB.

Prerequisite is a basic knowledge of C programming, but no expertise. No expertise in wireless networking or device specific knowledge is required. The goal of this guide is to reduce the complexity and confusion of embedded wireless to something simple and easy for everybody.

The guide covers the following topics:

- how to quickly develop your application on ICI, the event-driven platform
- how to interface to sensors or actuators using APIs for I2C, ADC, GPIO, UART, SPI
- how to create events based on timers, GPIO and UART
- how to join networks, send and receive messages
- how to compile and load your ICI application using our free development tools

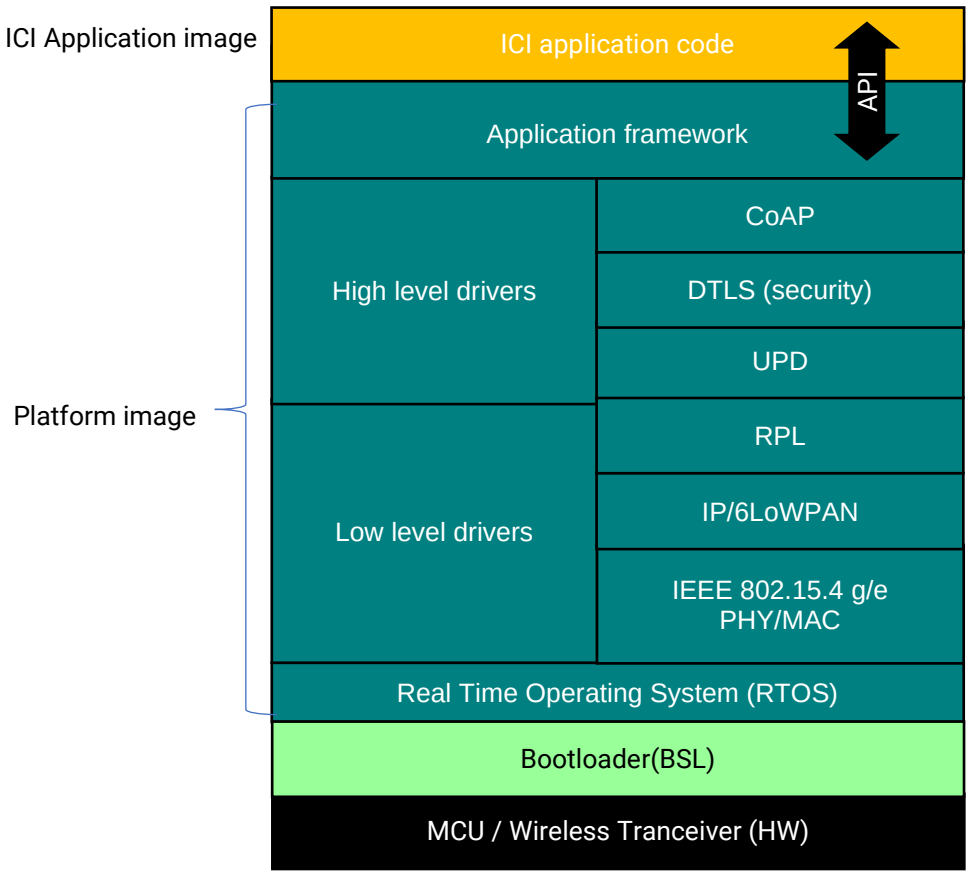

Figure 2. System overview

3 SDK structure

The SDK directory contains the following:

- **docs/**
  - The documentation
- **Framework/**
  - The SDK internals. **Do not modify the content unless you know what you are doing.**
- **Framework/Linker/**
  - Files used during linking.
- **Framework/Makefile**
  - Used by the build process.
- **Framework/Platform**
  - The location of the platforms. Also contains scripts for updating platform code
- **Framework/Tools/**
  - Containing the command line tools and scripts. These are described in detail later.
- **Framework/User_API/**
  - Containing the API header files
- **ICI_Applications/**
  - Your code as well as example ICI applications. **This is where most users will work.**

4 How to use ICI, the Event Driven Platform

4.1. ICI limitations and differences to "regular" C programming

ICI can use almost all features of the standard C programming language as you are used to. There are some exceptions to this:

4.1.1. Keep code execution time short

Execution time of an ICI function (event handler) should be kept short. If excessive time is used in a function, timing of other tasks such as system tasks may be off. A function taking more than 5 seconds to complete, may result in a full module reset, as the system will think the program has frozen.

4.1.2. You cannot use the **switch/case** – construct

```
    // Instead of this:
    switch(test){
        case 1:
            Func1();
        break;
        case 2:
            Func2();
        break;
    }

    // Use this:
    if(test==1){
        Func1();
    } else if (test==2){
        Func2();
    }
```

4.1.3. Variables must generally be global or static

```
    // Instead of this:
void Func1()
{
    NetworkState state;
    state=Network.getNetworkState();
    .....
}

// Use this:
void Func1()
{
    static NetworkState state;
    state=Network.getNetworkState();
    .....
}
```

4.1.4. Variables without the const keyword cannot be directly initialized when defined

```
void Func1()
{
    // This is illegal!
    static int a=10;

    // This is legal
    static int a;
    a=10;

    // This is also legal
    const int a=10;
}
```

4.2. Introduction

Everything in ICI is event-driven. It is up to the ICI user application to define the events and the handlers for these events.

Think of the ICI user application as a list of event handlers — **if**-this-**then**-that. **If** this event occurs, **then** do that in this handler.

As an example, a simple wireless sensor ICI application could be defined by the following event behaviors.

- if network state is offline then turn off the LED
- if network state is online then turn on the LED
- every 20 seconds, read data from the sensors on I2C
- after every sensor reading, send sensor data as CoAP message to a given IP address

Of course, a lot more is possible. Sensors and actuators can be accessed using different interfaces (I2C, ADC, GPIO, SPI, UART..). Periodic or one-shot timer events can be created. Custom actions can be triggered by network commands from the Border Router node. Global variables can be used to track internal states. Data can be saved to non-volatile memory to preserve them across power resets. Algorithms can be implemented to process or filter data.

Underneath, ICI silently takes care of the security and reliability of the network, any over-the-air updates, and the powerful real-time operating system (RTOS) that schedules the events.

The user code is is composed of `RIIM_SETUP()` and a list of event handlers. `RIIM_SETUP()` is the only function that **must** be implemented, and it is called from the application framwork when the node starts up. From within `RIIM_SETUP()`, you can define events and register handlers for these events — e.g. when you receive a message, or when a timer expires.

You can also chain events based on other events by defining an event within another event handler. For example, you can start a timer when a GPIO edge triggers.

5 An example ICI application

We will show you an example of a wireless voltage sensor using the built-in ADC.

This sensor will have the following behaviors:

- Finds and joins a network automatically
- Every 10 seconds, read data from the ADC
- Periodically, send a CoAP PUT message to a CoAP server – in this case the Border Router node

Take a look at the code implementation below. Only basic C knowledge is required. And keep in mind that the entire code is composed of RIIM_SETUP() and a list of event handlers.

Example : ICI code

```
#include "RIIM_UAPI.h"

static const uint32_t TimerPeriod=10;
static uint8_t CoAP_timer_handler, Sensor_timer_handler;
static const uint8_t Resource_Name[]="data";
static uint32_t milliVolts;

static void ReadSensor()
{
    ADC.convertToMicroVolts(ADC0, &milliVolts);
    milliVolts /= 1000;
    return;
}

static void SendCoAP()
{
    static IPAddr RootNodeIPAddr;

    // We first check if we are part of a network and the address of the border
router node
    if(Network.getBorderRouterAddress(&RootNodeIPAddr)!=UAPI_OK){
        return;
    }
    CoAP.connectToServer6(RootNodeIPAddr, false);

    // For this demo, we'll use JSON formatting. JSON is not size efficient,
    // but easy to read and parse
    uint8_t payload[100];
    int payload_length;

    payload_length=Util.sprintf((char*)payload,"{\"mV\": %i}", milliVolts);
    CoAP.send(CoAP_PUT, false, Resource_Name, payload, payload_length);

    return;
}

void ResponseHandler(const uint8_t *payload, uint8_t payload_size)
{
    Util.printf("# Got CoAP Response. Doing nothing with it...\n");
    return;
}
```

Example 1 - ADC sensor example

Hopefully you get an idea of how easy it is to define and register a handler for an event. In a little more than 30 lines of actual code you have created a full CoAP and Mesh-enabled ADC sensor.

6 SDK Setup

Please follow the **RIIM SDK Quick Start Guide** on how to install and setup RIIM SDK on your machine.

7 Building ICI Applications

Please follow the **RIIM SDK Quick Start Guide** on creating your first ICI application.

The ICI user application is edited in your favorite text editor, and compiled by invoking a simple script. The compiler used underneath is GCC, a very well known compiler. The user does not need to be familiar with compiler tools and configurations, as everything is taken care of by the scripts provided from Radiocrafts.

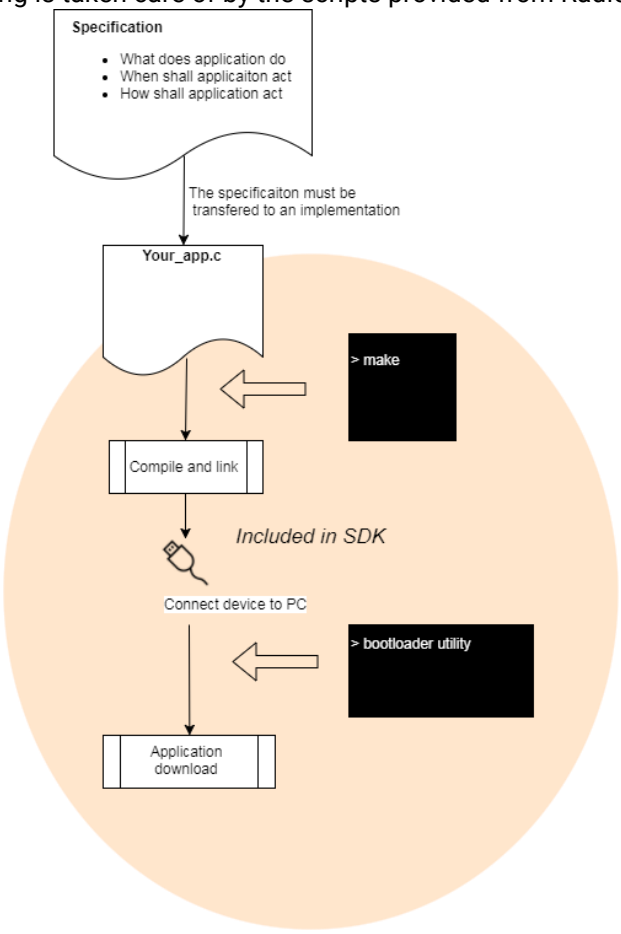

Figure 3. Workflow using RIIM SDK

The development tools runs on both Linux-like systems and Windows. Radiocrafts provides a complete development environment with everything that is needed that the user can download for free. Please see **RIIM SDK Quick Start Guide** on how to install the development environment. In the following examples, Linux is used. For windows, substitute the ".sh" file ending with ".bat" and "dev/ttyUSB0" with your associated COM port (for example "COM3").

7.1. Using Microsoft Visual Studio Code (VSCode)

The SDK provides a workspace file and setup for compilation and uploading ICI applications. To open the SDK in VSCode, open the file **RIIM.code-workspace** present at the top level of the SDK. To build an ICI application, simply

select the file you want to compile in the leftmost EXPLORER window and press **SHIFT-CTRL-B**. This builds whatever file you have selected and is open in the editor.
When building is finished, you are prompted «Enter port (or enter to abort) []:» in the bottom TERMINAL window. Here you enter the port your RIIM module is connected to, for instance **COM3** or **/dev/ttyUSB0**. Or just press **ENTER** to exit.

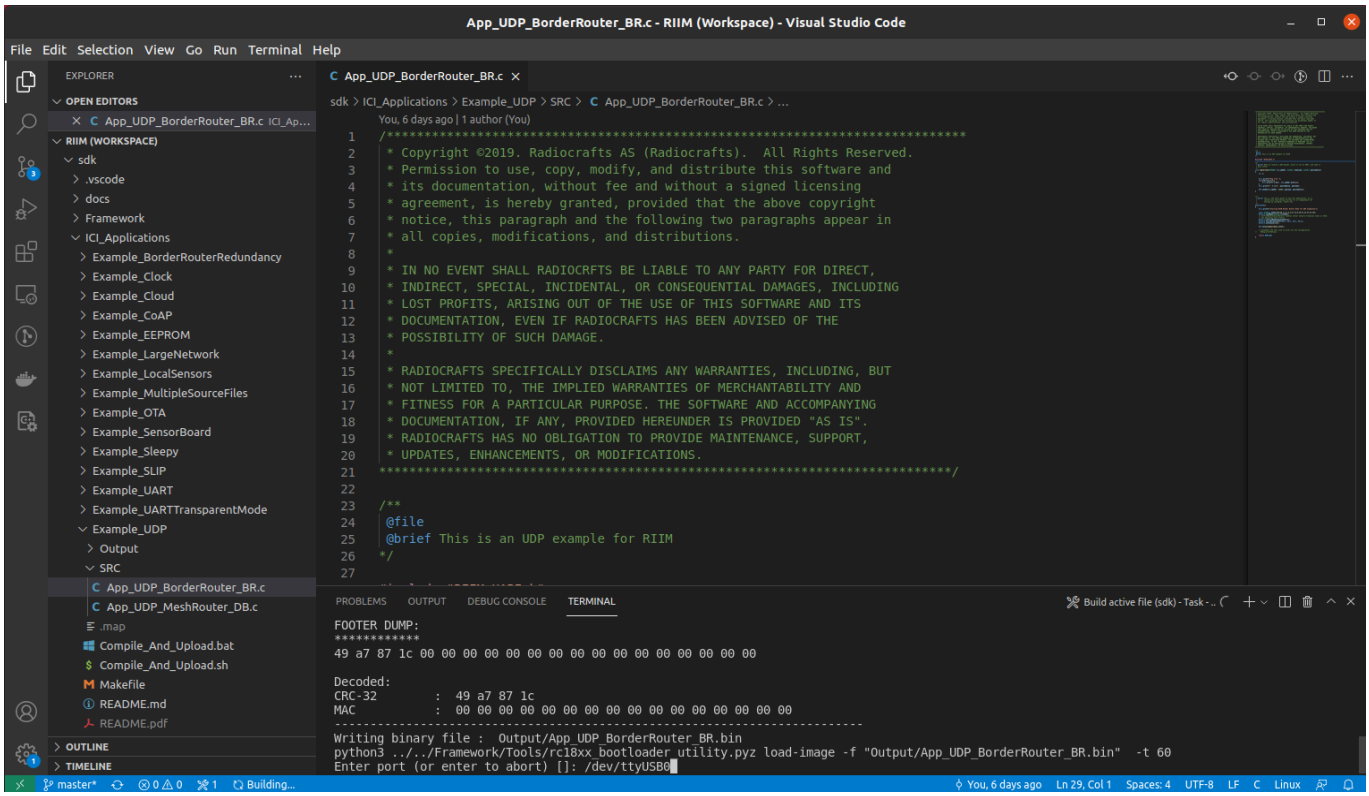

Figure 4 - Using VSCode for compilation and upload

7.2. Using makefiles

The build system is using makefiles, but the user does not need to know how they work. If you are not familiar with makefiles, the SDK provides a fully working and easy-to-edit makefile. You can copy it and use it in any new ICI user application project. It is located here: **RIIM_SDK/ICI_Applications/My_App/Makefile**
There are several default values in the makefile that can be overridden or edited, shown in the table below:

Value	Description
SOURCE_FILE	The source file that is to be compiled
OUTPUT_FILENAME	The output filename for your created ICI application. A «.bin» will be appended to the filename
PORT	Which serial port to use. In windows, this will be COM1, COM2 ...., and in linux it will be /dev/ttyUSB0, /dev/ttyUSB1...
AKEY	Filename for the ICI application encryption key
NKEY	Filename for the network encryption key
Framework_Path	The relative path to the framework

Table 1 - Makefile variables

The variables can be overridden as in these examples

```
$ make uploadImage PORT=COM3
$ make encrypted AKEY=MyKey.key
```

7.3. Not using makefiles

The makefiles uses a set of tools that is part of the SDK and the ARM compiler. You do not need to use makefiles, you can just use the tools directly or in your own script. The tools are located at RIIM_SDK/Framework/Tools . Below is a couple of examples executed from the SDK path RIIM_SDK/ICI_Applications/My_App :

This loads the image App_Router.bin onto the module connected to serial port ttyUSB0

```
$ ../../Framework/Tools/rc188x_bootloader_utility load-image -f Output/App_Router.bin -p /dev/ttyUSB0
```

This loads a new platform (the Border Router platform) onto the module connected to serial port ttyUSB0

```
$ ../../Framework/Tools/rc188x_bootloader_utility load-image -f
../../Framework/Platform/Output/RIIM_Platform_BorderRouter_0x010000.bin -p /dev/ttyUSB0
```

For all the tools, it is possible to specify the «-h» switch to get help, like this:

```
$ ../../Framework/Tools/rc188x_bootloader_utility -h
usage: bootloader_util [-h] [--version] [-f FILE] [-p PORT] [-t TIMEOUT]
                       command

positional arguments:
  command               info, load-image, load-app-image-key, load-network-
                        key, lock, run-app, autoconfig, autodetect

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -f FILE, --file FILE  image or key file
  -p PORT, --port PORT  Serial port, e.g. COM12. If unspecified, then the
                        default in the config file is used
  -t TIMEOUT, --timeout TIMEOUT
                        Timeout (in seconds) for trying to connect to
                        bootloader. Default is 10 seconds
```

And:

```
$ ../../Framework/Tools/rc188x_image_generator -h
RC188x Image Generator
usage: RC188X Image Generator [-h] [--app_version] [-p PLATFORM]
                              [-hw HARDWARE] [-r HARDWARE_REV]
                              [-t {app,platform}] [-v VERSION] [-erase_nv] -f
                              HEXFILE [-k KEYFILE] [-n NONCE]

optional arguments:
  -h, --help            show this help message and exit
  --app_version         show program's version number and exit
  -p PLATFORM, --platform PLATFORM
                        Platform ID. Example: 00A1
  -hw HARDWARE, --hardware HARDWARE
                        Hardware ID. Example: 00B0
  -r HARDWARE_REV, --hardware_rev HARDWARE_REV
                        Hardware Revision. Example: 0001
  -t {app,platform}, --image_type {app,platform}
                        Image type [app, platform]
  -v VERSION, --version VERSION
                        Image version. Format is hexadecimal 0xMMmmPP.
                        MM=Major version, mm=minor version, PP=patch version.
                        Example 0x010A04
```

```
  -erase_nv           Erase NV memory
  -f HEXFILE, --hexfile HEXFILE
                      Input hex file
  -k KEYFILE, --keyfile KEYFILE
                      Key file. File must contain 16 byte hex string.
  -n NONCE, --nonce NONCE
                      Nonce. Must be 11 byte hex string starting with 0x.
                      Example: 0x112233445566778899AABB
~/dev/RIIM_SDK/ICI_Applications/My_App$
```

Open the Makefile located at RIIM_SDK/Framework/Tools/RIIM.mk in a text editor for more examples.

7.4. Creating and Uploading the ICI User Application

The following example shows how to create (compile) an unencrypted ICI user application image and upload it to the node using the bootloader over USB.

```
$ make
$ make uploadImage
<USER MUST RESET THE MODULE>
```

7.5. Encrypting the ICI User Application

The ICI user application can be encrypted to protect the users Intellectual Property and secure the product from fraudulent software. To enable encryption of the ICI user application, two criteria must be met:

- The encryption key must be programmed into the bootloader
- The ICI user application image must be created with the same key

This is achieved using the scripts provided with the development environment from Radiocrafts. The example assumes that there exist a key file called **Application.key** in the same folder as the ICI application. The following sequence must be followed:

```
$ make encrypted
$ make uploadAppKey
<USER MUST RESET THE MODULE>
$ make uploadImage
```

7.6. Locking the module

Locking the module means that no one can upload unencrypted images to the module anymore. This feature helps protect the ICI application and is NOT reversible. It may be feasible to not lock the module if the user is actively developing on the module. Locking should be done before shipping the product to disable possible tampering.

```
$ make lock
<USER MUST RESET THE MODULE>
```

7.7. Read out the module serial number

Each module has a unique serial number that can be read out. This is done by using the bootloader utility like this:

```
$ ../../Framework/Tools/rc188x_bootloader_utility read-serial -p /dev/ttyUSB0
<USER MUST RESET THE MODULE>
```

7.8. Selecting and updating the platform

The SDK also contains the platforms as encrypted binaries. The platform images are located at **$RIIM_SDK/Framework/Platform/Output** . There are 3 platform types:

- RIIM_Platform_MeshRouter_0xXXXXXX-YYYY.bin – For Mesh Router nodes
- RIIM_Platform_BorderRouter_0xXXXXXX-YYYY.bin – For Border Routers
- RIIM_Platform_Leaf_0xXXXXXX-YYYY.bin – For low power leaf nodes

Here, XXXXXX denotes the platform version (e.g. 010200 for version 1.2.0), and YYYY denotes the MAC type (SingleChannel or TSCH).

To configure a module with a particular platform, you use the same tools and methods as you use for the ICI user application. Below is an example on how to transfer the Router Mesh platform to the module:

```
~/RIIM_SDK/ICI_Applications/My_App$ ../../Framework/Tools/rc188x_bootloader_utility load-image -f
../../Framework/Platform/Output/RIIM_Platform_MeshRouter_0x010000.bin -p /dev/ttyUSB0
Cannot connect to Bootloader. Failed attempt  1.0  of  10.0 .
MODULE INFORMATION
EUI64: 00124B001CBCABD8
Hardware ID: 0x0002 (Unknown)
Hardware Rev: 0x0002
Platform ID: 0x0002 (Unknown)
Platform Version: v1.9.0
App Version: v255.255.255
Bootloader Version: v2.1.2
Bootloader Variant: 0x02 (IP Mesh)
Lock State: 0xFF (Unlocked)

Loading Image
Waiting for Bootloader to initiate transfer...
Start transfer:
................................................................................................................
................................................................................................................
................................................................................................................
................................................................................................................
.....................................................................................End transfer.....
file upload successful
Waiting for bootloader status..
Bootloader Status: Success
```

8 API Usage

The following sections describes each of the API modules. The functions are presented in a simplified form here as a quick reference; please see the **RIIM SDK API Reference** document for details. The RIIM SDK API Reference is distributed with the SDK, and is located at **RIIM_SDK/docs/RIIM_API_Reference.html**

8.1. Multicast

Multicast is supported for IPv6 transmissions not requiring responses. This means that you can use multicast for the functions UDP.send6 and CoAP.sendNoAck. In the RIIM wireless network, multicast act as a Broadcast, as the target group is “all nodes”. The following example shows how to send a UDP packet to all nodes in the network.

Example : ICI code

```
const IPAddr Addr=UAPI_ADDR_ALL_RIIM_NODES;

UDP.send6(Addr, 12345, payload, payloadSize);
```

8.2. One-hop Multicast

RIIM supports multicast just to a node’s immediate neighbors as well. A multicast sent like this will not be able to hop, and will only be responded to by nodes within reach of the sending node.

Example : ICI code

```
const IPAddr Addr=UAPI_ADDR_LINKLOCAL_NODES;

UDP.send6(Addr, 12345, payload, payloadSize);
```

8.3. RIIM_UAPI

This is basically an include file that includes everything you need to develop in ICI. It includes all User API definitions and needed standard C header files.

Simply put this include line on top of your source file:

Example : ICI code

```
#include "RIIM_UAPI.h"
```

8.4. ADC

The ADC module lets you sample and convert the analog input of the module

RC1882-IPM supports 2 ADC channels, called ADC0 and ADC1.

The following example shows an example usage where the ADC is initialized, read, and the value is converted to millivolts and printed on the console.

Example : ICI code

```
static uint32_t milliVolts;

void ADC_DemoFunc()
{
    milliVolts=0;

    // Set up interface
    ADC.init(ADC0, ADC_FIXED_REFERENCE, ADC_SAMPLING_DURATION_10_6_US);

    ADC.convertToMicroVolts(ADC0, &milliVolts);
    milliVolts /= 1000;

    // Print directly to console to show the latest values
    Util.printf("ADC sensor read: Voltage(mV): %i\n",milliVolts);

    return;
}
```

Example 2 - ADC example code

The following functions are part of the ADC module.

Function	Description
init(Channel, Reference, SamplingDuration)	Initializes the ADC module input (either ADC0 or ADC1). It specifies which voltage reference to use and how long a sampling takes
convert(Channel, Value)	Convert an input voltage to its raw ADC value
convertToMicroVolts(Channel, Value)	Convert an input voltage to a integer representing the input voltage in microvolts

Table 2 - ADC functions

8.5. Clock

The Clock module provides ways to synchronize and distribute global time throughout the RIIM network. It requires TSCH to work. All the nodes part of a network are synchronized to within a few milliseconds of each other, enabling synchronous actions to take place across the RIIM network.
The Clock module provides two ways of representing time: Time of Day (ToD), and Epoch.
ToD provides a 24 hour clock represented by hour, minute, second and millisecond. It can be used to do an action at a specific time during a day, and can be repetitive.
Epoch provides an absolute time that is always increasing. It is represented in milliseconds, and can be used to measure absolute time or trigger an action at some specific time. It is not repetitive.

Function	Description
getToD(ToD)	Returns the ToD
setToD(ToD, AbsoluteTime)	Set the ToD
getEpoch(Epoch)	Returns the Epoch
setEpoch(Epoch, AbsoluteTime)	Set the Epoch
getAbsoluteTime()	Get absolute time – number of milliseconds since network was started
registerToDTimedAction(ToD, Repeat, Callback)	Register a ToD action
registerEpochTimedAction(Epoch, Callback)	Register an Epoch action
stopToDTimedAction(ID)	Stop a planned ToD action
stopEpochTimedAction(ID)	Stop a planned epoch action
sendClockToAllNodes()	Send Epoch and ToD to all nodes.
getClockFromBR()	Retrieve the Epoch and ToD from the border router
getOffsets(Offsets)	Get the raw offset values for ToD and epoch
setOffsets(Offsets)	Set the raw offset values for ToD and epoch
getLastSynchronizeTimestamp()	Get timestamp (after network was started) when the node last received a clock synchronization

Example : ICI code

```
EpochType epoch; // Set to zero in RIIM_SETUP()

// Epoch callback
void epochCallback(void)
{
    EpochType thisEpoch;
    Clock.getEpoch(&thisEpoch);
    Util.printf("Epoch callback at : %i", (int32_t)thisEpoch);

    // Set next callback to be in 5 seconds
    epoch=epoch+5000;
    Clock.registerEpochTimedAction(&epoch, epochCallback);
    Util.printf(" - New set to : %i\n", (int32_t)epoch);

    return;
}
```

8.6. CoAP

The CoAP module lets you start and run a CoAP server and lets you transmit CoAP messages to a server. The following example demonstrates how you initialize the CoAP module, set up a response handler and a resource access handler. When this node receives a PUT, POST, GET or DELETE CoAP message, it will call the **CoAP_Handler** routine. 5 CoAP resources can be registered, and CoAP.registerResouce will return an error if more recsorces are requested.

Example : ICI code

```
static const uint8_t Resource_Name[]="data";

static void CoAP_Handler(RequestType type, uint8_t *payload, uint8_t payloadSize,
uint8_t *response, uint8_t *responseSize)
{
    // In this example we only print the payload directly to the UART
    Debug.printPayload(payload, payloadSize);
    Util.printf("\n"); // Add a newline for readability in console
    return;
}

void ResponseHandler(const uint8_t *payload, uint8_t payload_size)
{
    Util.printf("# Got CoAP Response. Doing nothing with it...\n");
    return;
}

RIIM_SETUP()
{
    Network.startBorderRouter(NULL, NULL, NULL, NULL);

    // Setup coap resource
    CoAP.init(ResponseHandler);
    CoAP.registerResource(Resource_Name,0,CoAP_Handler);

    return UAPI_OK;
}
```

Example 3 - CoAP example code

The following functions are part of the CoAP module.

Function	Description
Init()	Initializes the CoAP module
connectToServer4(IPAddress, Secure)	Connect to a CoAP server using IPv4. This must be done before accessing it.
connectToServer6(IPAddress, Secure)	Connect to a CoAP server using IPv6. This must be done before accessing it.
setKey(Key, Length)	Set DTLS key
setIdentity(Identity, Length)	Set DTLS Identity
registerResponseHandler(Handler)	Register a handler to be executed when a CoAP response arrives
registerResource(Path, resourceID, CoAP_Handler)	Register a new resource that will be accessible on the node
generateResponse(Payload, PayloadSize, Response, ResponseSize)	Generate a response message and send it back to the caller
send(RequestType, Secure, Resource, Payload, PayloadSize)	Send a GET, PUT, POST or DELETE message to a CoAP server

sendNoAck(RequestType, Secure, Resource, Payload, PayloadSize)	Send a GET, PUT, POST or DELETE message to a CoAP server, but do not ask for a CoAP response
disconnectFromServer()	Disconnect from the CoAP server

Table 3 - CoAP functions

Note: CoAP packets will be retransmitted several times if they are lost on the way to their destination. Each retransmission with an increased timeout. The total time might be in the order of minutes, and hence the CoAP queue can easily fill up. It Is therefore important to wait for the CoAP response before sending new CoAP packets.

8.7. Debug

The Debug module contains useful functions for debugging.

The following example shows how you can dump the configuration of the node by calling Debug.printSetup().

Example : ICI code

```
RIIM_SETUP()
{
    Network.startBorderRouter(NULL, NULL, NULL, NULL);

    // Print out the configuration
    Debug.printSetup();

    return UAPI_OK;
}
```

Example 4 - Debug example code

The following functions are part of the Debug module.

Function	Description
printPayload(Payload, Payload_Size)	Dump the payload to the UART
printSetup()	Print the current node setup on the UART
printIPAddr(IPAddr)	Nicely print *IPAddr*
radioTest(TestType)	Test the radio in CW or MW mode. **IMPORTANT:** **1. This command is locking, meaning that the module needs to be externally reset after its called.** **2. This command cannot be called in RIIM_SETUP(), but must be executed in a callback**
getResetReason()	Return the reset code for last module reset

Table 4 - Debug functions

8.8. Ethernet

The Ethernet module is used to interface the Ethernet system. The Ethernet MAC is by default set by using the EUI64 / IEEE address of the module, which is unique to each module.

The following example shows how to read the MAC address

Example : ICI code

```
RIIM_SETUP()
{
    // Set Ethernet MAC address
    uint8_t Mac[6];
    Ethernet.getMACAddress(Mac);

    return UAPI_OK;
}
```

Function	Description
Init()	Initialize the GPIO module
`getMACAddress` (MacAddress)	Read the Ethernet MAC address

8.9. GPIO

The GPIO module is used to control the GPIOs. By default all GPIOs are tri-state inputs with no pullup or pulldown.

The following example shows how to configure GPIO 9 as output and setting it to logical '1'.

Example : ICI code

```
RIIM_SETUP()
{
    // Set up GPIO 9
    GPIO.setDirection(GPIO_9, OUTPUT);
    GPIO.setValue(GPIO_9, HIGH);

    return UAPI_OK;
}
```

Example 5 - GPIO example code

The following functions are part of the GPIO module.

Function	Description
Init()	Initialize the GPIO module
setDirection(Pin, Direction)	Set the GPIO direction to input or output
setPull(Pin,Pull)	Set pullup, pulldown or no pull on an input pin
setHandler(Pin, InterruptEdge, Handler)	Register a handler to be called when a specific transition happen on a GPIO
setValue(Pin, Value)	Set the value on an output GPIO to logical 0 or 1
toggle(Pin)	Invert the output value on an GPIO
getValue(Pin)	Read the value of a GPIO pin

Table 5 - GPIO functions

8.10. I2C

The I2C module has functions needed for interfacing the I2C bus. It always operates as master, and support clock stretching. It supports two speeds: 100KHz and 400KHz.

The following example demonstrates how the module is initialized and an example transmission towards a Sensirion SHT35 Temperature/Humidity sensor.

Example : ICI code

```
static void ReadSensor()
{
    static uint8_t wbuf[2];
    static uint8_t rbuf[10];

    // The command 0x2C0D starts a Medium repeatibility measuring
    // with clock stretching enabled. See Sensirion SHT35 datasheet.
    wbuf[0]=0x2C;
    wbuf[1]=0x0D;

    I2C.transfer(0x44,wbuf,2,rbuf, 7);

    .......

}

RIIM_SETUP()
{
    // Set up interface towards the SHT35-sensor
    I2C.init(I2C_400KHZ);

    .........

    return UAPI_OK;
}
```

Example 6 - I2C example code

The following functions are part of the I2C module.

Function	Description
init(Speed)	Initializes the I2C driver with the chosen Speed
transfer(SlaveAddress, WriteBuffer, WriteLenth, ReadBuffer, ReadLength)	Transfers data both to and from the slave at bus address SlaveAddress. Data is first written on bus, then read
read(SlaveAddress, ReadBuffer, ReadLength)	Perform only read on the I2C bus
write(SlaveAddress, WriteBuffer, WriteLength)	Perform only write on the I2C bus
close()	Closes the I2C interface

Table 6 - I2C functions

8.11. Network

The Network module has the functions needed for network operations. Before the node can join any network, it must be started by a call to either startBorderRouter, startMeshRouter or startLeaf. Some functions will only have effect depending on the RIIM variant (single channel or TSCH) used.

The following example shows how to start a node as a Router:

Example : ICI code

```
RIIM_SETUP()
{
    // Simply start as a router node. Nothing more is needed.
    Network.startMeshRouter();

    return UAPI_OK;
}
```

Example 7 - Network example code

The following functions are part of the Network module.

Function	SC	TSCH	Description
startBorderRouter(IPv6_Prefix, IPv4_Address, IPv4_Netmask, IPv4_Gateway)	x	x	Start the node as a Border Router. The arguments are optional, but must be provided for your chosen technology (IPv4 or IPv6) if you want to enable automatic connection to the outside world
startMeshRouter()	x	x	Start the node as a Mesh Router. This means that the node is capable of routing packets through itself
startLeaf()	x	x	Start the node as a Leaf. This means that the node is NOT capabe of forwarding packets to other nodes, but will use less energy
getBorderRouterAddress(IPAddress)	x	x	This function returns the address of the WSN Border Router node. It will only do so if the node is part of a WSN and the Border Router is reachable.
getParentRSSI()	x	x	Get the RSSI value of the last packet received
getNeighbors(NeighborTable)	x	x	The a list of all neighbors, including their RSSI
getNetworkState()	x	x	Get the state of the network
radio_On()	x		Turn radio on
radio_Off()	x		Turn radio off. Only applies to low power mode in Leaf nodes.
join()	x	x	(Re)-join a network
leave()			RESERVED
setFreqBand(FrequencyBand)	x	x	Sets the frequency band.
setPanId()	x	x	Set the PAN ID of the node
getPanId()	x	x	Get the PAN ID of the node
getAddress(IPAddress)	x	x	Get the IP (v6) address of the node
getAddress4(IPAddress)	x	x	Get the IPv4 address of the node
getNetworkLinks(Startindex, NumLinks, Links)	x	x	Retrieves a list of links between nodes. This is the topology of the WSN. Can only be called in Border Router nodes
setTxPower(TxPower)	x	x	Set the output transmission power. See the RIIM API Reference for possible values for the different modules.
getNodeType()	x	x	Return the type of node we are configured as
set Channel(ChannelNumber)	x		Set the RF channel to use
isPartOfNetwork(IP_Address)	x	x	Check if a gived IP address is part of the network. Can only be called from Border Router nodes
setNWKey(key)	x	x	Set the network (LLSEC) key

setRobustnessFactor(Robustnessfactor)	x	x	Set the node's robustness factor
setTschMaxBrodcastRate(rate_s).		x	Set the maximum network broadcast period
getNodeCount()	x	x	Get the number of nodes in the network
startSlip()	x	x	Start the SLIP interface
setTSCHMode(Mode)		X	Deprecated. Part of setTSCHParameters
setTSCHParameters(Active, UnicastPeriod, SharedPeriod, EBSFPeriod)		x	Sets TSCH Mode, Unicast period, shared period and EBSF period.

Table 7 - Network functions

8.12. Node

The Node module contains functions for retrieving node info.

Example:

Example : ICI code

```
static void DemoFunc()
{
    NodeInfo info;
    Node.getNodeInfo(&info);
    Util.printf("HW Revision is %i\n", NodeInfo.Hardware_Rev);

    return;
}
```

Example 8 - Node example code

The following functions are part of the Node module:

Function	Description
getNodeInfo(Node_Info)	Return info about the node: Hardware ID+Revision, Platform ID+Version

Table 8 - Node functions

8.13. SPI

The SPI module has all the functions needed for accessing the SPI bus. For chip select control, the user must use a GPIO. For all data transfer and initialization, the SPI module can be used. In the example below, usage of both GPIO and SPI is demonstrated. Only 4-wire SPI is supported

The following example shows how to set up SPI and access the ST LIS3DE accelerometer:

Example : ICI code

```
RIIM_SETUP()
{
    // Set up GPIO. We must use one GPIO for SPI chip select
    GPIO.setDirection(GPIO_3, OUTPUT);
    GPIO.setValue(GPIO_3, HIGH);

    // Set up interface towards the LIS3DE-sensor
    SPI.init(4000000,SPI_POL_0_PHA_0);

    // Setup accelerometer to update every second
    uint8_t wbuf[2];
    wbuf[0]=0x20; // 0 (write), 0(do not increment), 0x20 (register address)
    wbuf[1]=0x17; // 1 Hz, enable X,Y and Z axes

    GPIO.setValue(GPIO_3, LOW);
    SPI.transfer(2, wbuf, NULL);
    GPIO.setValue(GPIO_3, HIGH);

    return UAPI_OK;
}
```

Example 9 - SPI example code

The following functions are part of the SPI module:

Function	Description
init(Speed, SPIClockMode)	Initialize the SPI module. Speed and SPI mode must be specified
transfer(Count, txBuffer, rxBuffer)	Transfer bytes over the SPI bus. Reading bytes is implicit, but any of the buffers can be set to **NULL** to disable reading or writing (disable writing means putting zero on the MOSI line)

Table 9 - SPI functions

8.14. Timer

The Timer module enables the user to create various timers. The timers will call a function when they are triggered, and can be configured as periodic or one-shot, started and stopped.

The following example creates a periodic timer that calls the function **ReadSensor** every 10 seconds:

Example : ICI code

```
static const uint32_t TimerPeriod=10000;
static uint8_t Sensor_timer_handler;

static void ReadSensor()
{
    Util.printf("Reading sensors....\n");

    return;
}

RIIM_SETUP()
{
    // Create timer that sends the CoAP PUT message
    Sensor_timer_handler=Timer.create(PERIODIC, TimerPeriod, ReadSensor);
    Timer.start(Sensor_timer_handler);

    return UAPI_OK;
}
```

Example 10 - Timer example code

Function	Description
create(TimerMode, Period, Handler)	Create a new timer
start(TimerID)	Start the timer
stop(TimerID)	Stop the timer
isActive(TimerID)	Check if the timer is active
config(TimerID, TimerMode,Period, TimerHandler)	(Re)-configure a timer

Table 10 - Timer functions

8.15. UART

The UART module controls the UART port. The uart can be configured with respect to baudrate, parity, number of databits and number of stopbits.

Example:

Example : ICI code

```
static void TxCallback(uint8_t len)
{
    Util.printf("Sent %i bytes\n", len);

    return;
}

static void DemoFunc()
{
    const uint8_t buffer="Test";

    UART.init(115200, UART_PARITY_NONE, UART_DATA_8_BITS, UART_STOP_1_BIT);
    UART.startTransmit(buffer, sizeof(buffer), TXCallback);

    return;
}
```

Example 11 - UART example code

Function	Description
Init (Baudrate, Parity, DataBits,StopBits)	Initialize the UART module. Baudrate, Parity, Databits and Stopbits must be specified
startReceive (UARTReceiveCallback, length)	Start reception and call a function when a number of bytes are received
cancelReceive ()	Stop reception
startTransmit (Buffer, Length, UARTTransmitCallback)	Start transmission of a buffer of bytes
isTXActive()	Returns true if UART TX is active
isRXActive()	Returns true if UART RX is active
enableRXTX()	Enables RXTX functionality on CTS pin. Compatible with RS485
startReceive_Adv(callback, maxLength, intraByteTimeout, totalTimeout, terminationByte, dynamicLength)	Advanced version of startReceive. Used to specify different timeouts, termination characters and dynamic number of bytes to be received.
wakeOnUART(callback)	Enable or disable wake-on-uart. When enabled, a high-to-low transition on the RX pin will trigger a callback function. If a character is used to wake the module, it should be 0x00, as this only generates one falling edge (8N1).

Table 11 - UART functions

8.16. UDP

The UDP module controls the sending and reception of UDP packets. It has functions for receiving and sending over IPv6 or IPv4.

Example:

Example : ICI code

```
void udpCallback(IPAddr src_ipAddr, uint8_t *payload, uint8_t payloadSize)
{
    int i;

    Util.printf("Msg from ");
    for(i=0;i<16;i++){
        Util.printf("%.02x", src_ipAddr.byte[i]);
    }
    Util.printf(" : %.*s\n", payloadSize, payload);
    UDP.send6(src_ipAddr, 12345, payload, payloadSize);
}

RIIM_SETUP()
{
    .....
    UDP.setup(udpCallback,12345);
    .....
}
```

Example 12 - UDP example code

Function	Description
send4(Address, port, payload, payload size)	Send a UDP packet to an IPv4 address. Typically to the internet
send6(Address, port, payload, payload size)	Send a UDP packet to an IPv6 address. Typically inside the RIIM wireless network
setup (Handler, Port)	Setup a handler for reception of UDP packets

Table 12 - UDP functions

8.17. Util

The Util module contains useful helper functions. The module provides common C library functions for the ICI user application to use: printf, sprintf, snprinf. It also provides a random generator.

The following example shows how to print something on the console (UART)

Example : ICI code

```
static void DemoFunc()
{
    Util.printf("Reading sensors....\n");

    return;
}
```

Example 13 - Util example code

Function	Description
printf(const char *format,...)	Standard C-library printf
sprintf(char *buffer, const char *format,...)	Standard C-library sprintf
snprintf(char *buffer, size_t buf_size, const char *format,...)	Standard C-library snprintf
getRand()	Get a random number
reset()	Reset the module
getTemperature()	Get module temperature
strlen(const char *str)	Return string length
strcmp(const char *str1, const char *str2)	Standard C-library strcmp
strncmp(const char *str1, const char *str2, size_t n)	Standard C-library strncmp
getVoltage()	Get voltage on the supply pin of the module

Table 13 - Util functions

Document Revision History

Document Revision	Changes
1.00	Advance Information
1.10	Added commissioning, updated code examples, API description, changed module names
1.20	Updated and added examples. Removed unused API functions. Removed watermark
1.30	Updated examples, updated API functions, ICI limitations, tool description
1.40	Update APIs, updated SDK file structure, updated API names
1.50	Added new functions for SDK 1.1.0. Added UDP API, Multicast
1.60	Added new functions for SDK 1.2.0
2.00	Updated for SDK version 2.0.0 Added one-hop (link local) multicast Added SLIP interface Added Clock API Added TSCH modes and parameters Added serial number read-out
3.00	Updated for SDK version 3.0.0 Added Ethernet module Updated TSCH parameter settings Added new UART functionality
3.10	Added chapter about VSCode

Disclaimer

Radiocrafts AS believes the information contained herein is correct and accurate at the time of this printing. However, Radiocrafts AS reserves the right to make changes to this product without notice. Radiocrafts AS does not assume any responsibility for the use of the described product; neither does it convey any license under its patent rights, or the rights of others. The latest updates are available at the Radiocrafts website or by contacting Radiocrafts directly.

As far as possible, major changes of product specifications and functionality, will be stated in product specific Errata Notes published at the Radiocrafts website. Customers are encouraged to check regularly for the most recent updates on products and support tools.

Trademarks

RC232™ is a trademark of Radiocrafts AS. The RC232™ Embedded RF Protocol is used in a range of products from Radiocrafts. The protocol handles host communication, data buffering, error check, addressing and broadcasting. It supports point-to-point, point-to-multipoint and peer-to-peer network topologies.
RIIM™ is a trademark of Radiocrafts AS.

All other trademarks, registered trademarks and product names are the sole property of their respective owners.

Life Support Policy

This Radiocrafts product is not designed for use in life support appliances, devices, or other systems where malfunction can reasonably be expected to result in significant personal injury to the user, or as a critical component in any life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness. Radiocrafts AS customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Radiocrafts AS for any damages resulting from any improper use or sale.

Radiocrafts Webpage

For more info go to our web page : https://radiocrafts.com/
There you can find Knowledge base and Document Library that includes Application notes, Whitepapers, Declaration of Conformity, User Manuals, Data Sheet and more.

Contact Radiocrafts

Sales requests: https://radiocrafts.com/contact/