

# BANO node software development kit

`texane@gmail.com`

# 1 Overview

This document is intended for node developers. It documents the BANO node software development kit, including the build environment and programming interface.

## 1.1 Installing the SDK

The BANO SDK is retrieved by cloning the following repository:

```
git clone ssh://texane@dedibox/home/texane/repo/bano
```

Also, BANO depends on the NRF source code:

```
git clone ssh://texane@dedibox/home/texane/repo/nrf
```

## 2 Build environment

### 2.1 Makefile variables

BANO allows the developer to interact with the build system by defining or overriding default variables in the node makefile:

- BANO\_DIR: the BANO top directory. It must be defined by the developer,
- NRF\_DIR: the NRF top directory. By default, the SDK assumes that the NRF directory is located in the parent directory on directory,
- BANO\_C\_FILES: the list of C source files. This list can not be empty,
- BANO\_NODL\_ID: the NODL identifier as an hexadecimal 32 bits unsigned integer. If not defined, 0 is used,
- BANO\_NODE\_ADDR: the node address as an hexadecimal 32 bits unsigned integer. If not defined, a random value is generated by the build system,
- BANO\_NODE\_SEED: the node seed as an hexadecimal 32 bits unsigned integer. If not defined, a random value is generated by the build system,
- BANO\_NODE\_KEY: the key used to encrypt and decrypt messages. It is represented as a comma separated string of 16 8 bits unsigned integers in hexadecimal. If not defined, a random value is generated by the build system,

### 2.2 Example makefile

```
# node built for the arduino minipro 3.3v board
# it assumes that BANO_DIR is defined by the environment
# it consists of a single C file main.c
# the node address is statically defined

BANO_NODE_ADDR := 0x5c5f8548
BANO_C_FILES := main.c
include $(BANO_DIR)/build/node/minipro_3v3.mk
```

## 3 Programming reference

### 3.1 Overview

BANO offers a runtime and the corresponding programming interface that abstracts a node application logic from low level details such as hardware architecture and protocol implementation. The interface is mainly descriptive and event based: the developer first initializes node related information. The BANO runtime then calls application handlers whenever appropriate: network messages reception, timers, hardware related interrupts ...

### 3.2 Files

common/bano.common.h: constants and types common to base and node

node/bano.node.h: function and type declarations

node/bano.node.c: function implementations

### 3.3 Types

```
typedef struct
{
    /* 100 milliseconds units, max 10736 */
    uint16_t timer_100ms;

    /* waking event mask */
#define BANO_WAKE_NONE 0
#define BANO_WAKE_TIMER (1 << 0)
#define BANO_WAKE_MSG (1 << 1)
#define BANO_WAKE_POLL (1 << 2)
#define BANO_WAKE_PCINT (1 << 3)
    uint8_t wake_mask;

    /* module disabling mask */
#define BANO_DISABLE_ADC (1 << 0)
#define BANO_DISABLE_WDT (1 << 1)
#define BANO_DISABLE_CMP (1 << 2)
#define BANO_DISABLE_USART (1 << 3)
#define BANO_DISABLE_NONE 0x00
#define BANO_DISABLE_ALL 0xff
    uint8_t disable_mask;

    uint32_t pcint_mask;
} bano_info_t;

static const bano_info_t bano_info_default =
{
    .wake_mask = BANO_WAKE_NONE,
    .disable_mask = BANO_DISABLE_ALL
};
```

### 3.4 Functions

```
/* exported by the runtime */
uint8_t bano_init(const bano_info_t*);
uint8_t bano_fini(void);
uint8_t bano_send_set(uint16_t, uint32_t);
uint8_t bano_wait_event(bano_msg_t*);
uint8_t bano_loop(void);

/* implemented by the application */
extern uint8_t bano_set_handler(uint16_t, uint32_t);
extern uint8_t bano_get_handler(uint16_t, uint32_t*);
extern uint8_t bano_timer_handler(void);
extern uint8_t bano_pcint_handler(void);
```

## 4 Examples

### 4.1 Enable/disable a LED on BANO messages

```
#include <stdint.h>
#include <avr/io.h>
#include "bano/src/node/bano_node.h"

/* led routines */

#define LED_DDR DDRB
#define LED_PORT PORTB
#define LED_MASK (1 << 1)

static void led_set_high(void)
{
    LED_DDR |= LED_MASK;
    LED_PORT |= LED_MASK;
}

static void led_set_low(void)
{
    LED_DDR |= LED_MASK;
    LED_PORT &= ~LED_MASK;
}

/* event handlers */

#define LED_KEY 0x0000

static uint8_t led_value = 0;

uint8_t bano_get_handler(uint16_t key, uint32_t* val)
{
    /* called by the runtime on GET messages */

    if (key != LED_KEY) return (uint8_t)-1;
    *val = led_value;
    return 0;
}

uint8_t bano_set_handler(uint16_t key, uint32_t val)
{
    /* called by the runtime on SET messages */

    if (key != LED_KEY) return (uint8_t)-1;
    led_value = val;
    if (led_value == 0) led_set_low();
    else led_set_high();
    return 0;
}

int main(void)
{
    /* initialize the runtime and loop forever */

    bano_info_t info;

    info = bano_info_default;
    info.wake_mask |= BANO_WAKE_MSG;
    bano_init(&info);

    bano_loop();

    bano_fini();

    return 0;
}
```

## 4.2 Send periodic messages

```
#include <stdint.h>
#include <avr/io.h>
#include "bano/src/node/bano_node.h"

uint8_t bano_timer_handler(void)
{
    /* called every 10 seconds */

    bano_send_set(0x002a, 0xdeadbeef);
    return 0;
}

int main(void)
{
    bano_info_t info;

    info = bano_info_default;

    info.wake_mask |= BANO_WAKE_TIMER;
    info.timer_100ms = 100;

    bano_init(&info);

    bano_loop();

    bano_fini();

    return 0;
}
```

### 4.3 Send message when GPIO changes

```
#include <stdint.h>
#include <avr/io.h>
#include "bano/src/node/bano_node.h"

#define GPIO_KEY 0x0000

#define GPIO_DDR DDRD
#define GPIO_PIN PIND
#define GPIO_MASK (1 << 3)

uint8_t bano_pcint_handler(void)
{
    bano_send_set(GPIO_KEY, GPIO_PIN & GPIO_MASK);
    return 0;
}

int main(void)
{
    bano_info_t info;

    info = bano_info_default;

    info.wake_mask |= BANO_WAKE_PCINT;
    info.pcint_mask = 1UL << 19UL;
    bano_init(&info);

    bano_loop();

    bano_fini();

    return 0;
}
```