RUI API Reference

Version V1.2 | October 8, 2019

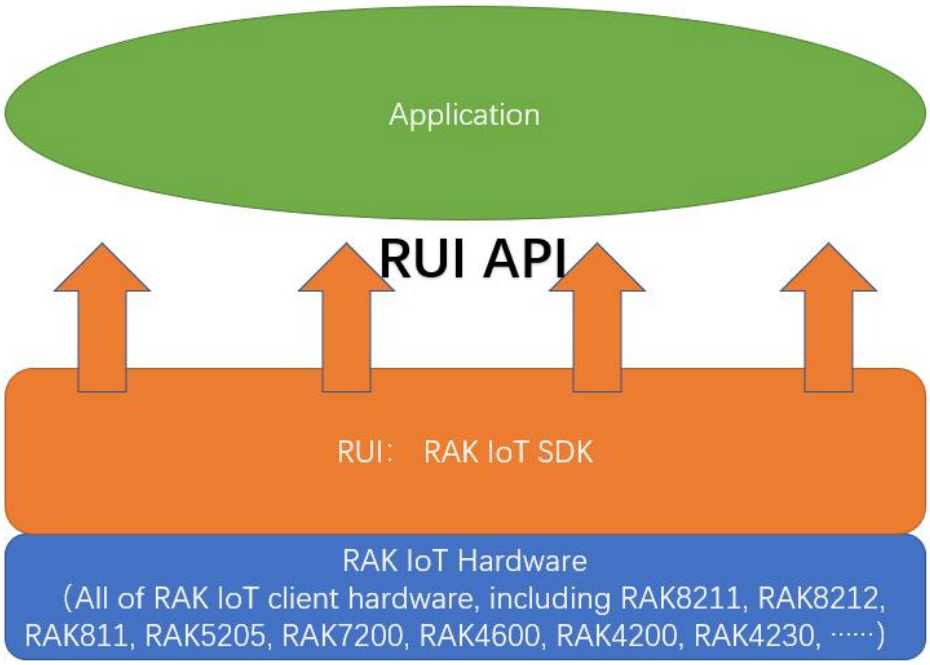
www.RAKwireless.com

Visit our website for more document.

Table of Contents

1. Overview

This document is the API reference of RUI.
You can develop an application as you want very easily and elegantly on RAK IoT products and modules by using these APIs.



Global definition:

```
#define SUCCESS          1
#define FAIL             0
typedef enum DRIVER_MODE
{
    RUI_NORMAL_MODE = 0,
    RUI_POWER_ON_MODE,
    RUI_POWER_OFF_MODE,
    RUI_SLEEP_MODE,
    RUI_STANDBY_MODE
}DRIVER_MODE;
```

2. Cellular

General format: **rui_cellular_xxx()**

2.1 rui_cellular_send

uint32_t rui_cellular_send(uint8_t *data)

```
*******************************************************************************
 *   @brief        This API is used to send data through cellular.
 *   @return       SUCCESS or FAIL
 *   @param        uint8_t *data:       the data which will be sent through cellular.
 *******************************************************************************
```

2.2 rui_cellular_response

uint32_t rui_cellular_response(uint8_t *response, uint32_t len, uint32_t timeout)

```
*******************************************************************************
 *   @brief        This API is used to wait for a correct response from cellular module.
 *   @return       SUCCESS or FAIL
 *   @param        uint8_t *response:      the response of server string
                   uint32_t len:           response data length
                   uint32_t timeout:       the time to wait for response until timeout.
*******************************************************************************
```

2.3 rui_cellular_register_recv_callback

typdef void (*cellular_receive)(uint8_t *data);
uint32_t rui_cellular_register_recv_callback(cellular_receive callback)

```
*******************************************************************************
 *   @brief        This API is used to register a callback function for cellular in application
                   so that application can receive cellular data automatically.
 *   @return       SUCCESS or FAIL
 *   @param        cellular_receive callback:  the callback function for receiving cellular
data.
*******************************************************************************
```

2.4 rui_cellular_set_mode

uint32_t rui_cellular_set_mode(DRIVER_MODE mode)

```
*******************************************************************************************
 *   @brief        This API is used to set the work mode of the cellular module.
 *   @return       SUCCESS or FAIL
 *   @param        DRIVER_MODE mode:    the cellular module's work mode
 *******************************************************************************************
```

2.5 rui_cellular_open_socket

```
uint32_t rui_cellular_open_socket(uint8_t* data)
*******************************************************************************************
*     @brief        This API is used to open a TCP socket with a remote server.
*     @return       SUCCESS or FAIL
*     @param        uint8_t* data:       open TCP link cmd, for example:
                                         AT+QIOPEN=1,0,"TCP","ip",%port,0,1"
*******************************************************************************************
```

2.6 rui_cellular_set_operator

```
uint32_t rui_cellular_set_operator(uint8_t *APN,uint8_t *operator_long_name,
                                   uint8_t *operator_short_name,uint8_t operator_net)
*******************************************************************************************
*     @brief        This API is used to configure parameters of Cellular operator.
*     @return       SUCCESS or FAIL
*     @param        uint8_t *APN:                    The APN name of Cellular operator.
                    uint8_t *operator_long_name:        The operator's long name.
                                                    For example, "CHINA MOBILE".
                    uint8_t *operator_short_name:   The operator's short name.
                                                    For example, "CMCC".
                    uint8_t operator_net:               The cellular network type.
*******************************************************************************************
```

3. LoRa

General format: **rui_lora_xxx()**
General definition:

```
typedef enum LORA_JOIN_MODE
{
    RUI_OTAA = 0,
```

```
    RUI_ABP
}LORA_JOIN_MODE;

typedef enum LORA_CLASS_MODE
{
    CLASS_A = 0,
    CLASS_B,
    CLASS_C
}LORA_CLASS_MODE;

typedef enum LORA_WORK_MODE
{
    RUI_LORAWAN = 0,
    RUI_P2P,
    RUI_TESTMODE
}LORA_WORK_MODE;

typedef enum LORA_REGION
{
    AS923,
    AU915,
    CN470,
    CN779,
    EU433,
    EU868,
    IN865,
    KR920,
    US915,
    US915_Hybrid
}LORA_REGION;

typedef struct RUI_DEVICE_STATUS
{
    RUI_UART_MODE uart_mode;
    uint16_t gps_searchtimer;
    bool voltage_status;
    bool gps_status;
    bool acc_status;
    bool magn_status;
    bool gyro_status;
    bool press_status;
    bool tmp_status;
    bool humidity_status;
    bool autosend_status;
```

```
}RUI_DEVICE_STATUS_T;

typedef struct RUI_LORA_STATUS
{
    RUI_LORA_WORK_MODE work_mode;
    RUI_LORA_CLASS_MODE class_status;
    RUI_LORA_JOIN_MODE join_mode;
    uint8_t lora_dr;
    uint8_t confirm;
    uint16_t lorasend_interval;
    bool IsJoined;
    bool AdrEnable;
    uint8_t region[5]; //region string e.g:"EU868"
}RUI_LORA_STATUS_T;
```

3.1 rui_lora_send

```
uint32_t rui_lora_send(uint8_t port,uint8_t* data,uint8_t len)

*******************************************************************************************
 *   @brief        rui_lora_send
 *   @return       SUCCESS or FAIL
 *   @param        uint8_t port:   send data port
                   uint8_t* data: send data string
                   uint8_t len:    send data length
*******************************************************************************************
```

3.2 rui_lora_register_recv_callback

```
typdef void (*lora_receive)(uint8_t *data);
uint32_t rui_lora_register_recv_callback(lora_receive callback)

*******************************************************************************************
 *   @brief        This API is used to register a callback function for LoRa in application,
                   so that application can receive the LoRa data automatically.
 *   @return       SUCCESS or FAIL
 *   @param        lora_receive callback:      the callback function for receiving LoRa
data.
*******************************************************************************************
```

3.3 rui_lora_set_device_mode

uint32_t rui_lora_set_device_mode(DRIVER_MODE mode)

```
*******************************************************************************************
 *   @brief        This API is used to set the work mode of LoRa module.
 *   @return       SUCCESS or FAIL
 *   @param        DRIVER_MODE mode:    lora peripheral work mode
*******************************************************************************************
```

3.4 rui_lora_set_dev_eui

uint32_t rui_lora_set_dev_eui(uint8_t *dev_eui)

```
*******************************************************************************************
 *   @brief        This API is used to set the device EUI for LoRaWAN OTAA mode.
 *   @return       SUCCESS or FAIL
 *   @param        uint8_t *dev_eui:  the device EUI.
*******************************************************************************************
```

3.5 rui_lora_set_app_eui

uint32_t rui_lora_set_app_eui(uint8_t *app_eui)

```
*******************************************************************************************
 *   @brief        This API is used to set the application EUI for LoRaWAN OTAA mode.
 *   @return       SUCCESS or FAIL
 *   @param        uint8_t *app_eui:        the application EUI.
*******************************************************************************************
```

3.6 rui_lora_set_app_key

uint32_t rui_lora_set_app_key(uint8_t *app_key)

```
*******************************************************************************************
 *   @brief        This API is used to set the application key for LoRaWAN OTAA mode.
 *   @return       SUCCESS or FAIL
 *   @param        uint8_t *app_key:        the application key.
*******************************************************************************************
```

3.7 rui_lora_set_dev_addr

uint32_t rui_lora_set_dev_addr(uint8_t *dev_addr)

```
*******************************************************************************************
 *   @brief        This API is used to set the device address for LoRaWAN ABP mode.
 *   @return       SUCCESS or FAIL
 *   @param        uint8_t *dev_addr:      the device address.
*******************************************************************************************
```

3.8 rui_lora_set_apps_key

uint32_t rui_lora_set_apps_key(uint8_t *apps_key)

```
*******************************************************************************************
 *   @brief        This API is used to set the application session key for LoRaWAN ABP
mode.
 *   @return       SUCCESS or FAIL
 *   @param        uint8_t *apps_key: the application session key.
*******************************************************************************************
```

3.9 rui_lora_set_nwks_key

uint32_t rui_lora_set_nwks_key(uint8_t *nwks_key)

```
*******************************************************************************************
 *   @brief        This API is used to set the network session key for LoRaWAN ABP
mode.
 *   @return       SUCCESS or FAIL
 *   @param        uint8_t *nwks_key:      the network session key.
*******************************************************************************************
```

3.10 rui_lora_set_channel_mask

uint32_t rui_lora_set_channel_mask(uint8_t channel, uint8_t on_off)

```
*******************************************************************************************
```

```
 *   @brief      This API is used to turn a certain channel on or off.
 *   @return     SUCCESS or FAIL
 *   @param      uint8_t channel:   the channel number you want to set.
                 Uint8_t on_off:    turn on or turn off.
*********************************************************************************************
```

3.11 rui_lora_class

uint32_t rui_lora_set_class(LORA_CLASS_MODE class)

```
*********************************************************************************************
 *   @brief      This API is used to set the LoRaWAN Class.
 *   @return     SUCCESS or FAIL
 *   @param      LORA_CLASS_MODE class: Class A, Class B, or Class C.
*********************************************************************************************
```

3.12 rui_lora_set_dr

uint32_t rui_lora_set_dr(uint8_t dr);

```
*********************************************************************************************
 *   @brief      This API is used to set the DR for LoRa node.
 *   @return     SUCCESS or FAIL
 *   @param      uint8_t dr:     the value of DR.
*********************************************************************************************
```

3.13 rui_lora_set_join_mode

uint32_t rui_lora_set_join_mode(LORA_JOIN_MODE mode)

```
*********************************************************************************************
 *   @brief      This API is used to set the join mode of LoRaWAN.
 *   @return     SUCCESS or FAIL
 *   @param      LORA_JOIN_MODE mode:      OTAA or ABP
*********************************************************************************************
```

3.14 rui_lora_set_work_mode

uint32_t rui_lora_set_work_mode(LORA_WORK_MODE mode)

```
*******************************************************************************
 *   @brief        This API is used to set the work mode of LoRa module.
 *   @return       SUCCESS or FAIL
 *   @param        LORA_WORK_MODE mode:    LaRaWAN, P2P, or Test mode.
*******************************************************************************
```

3.15 rui_lora_set_send_interval

```
/*******************************************************************************
 *   @brief        This API is used to set the interval time of sending data.
 *   @return       SUCCESS or FAIL
 *   @param            bool status:0 disable,1 enable
 *                     uint16_t app_interval:  the interval time of sending data.(unit:s)
*******************************************************************************/
```

uint32_t rui_lora_set_send_interval(bool status,uint16_t interval_time);

3.16 rui_lora_set_region

uint32_t rui_lora_set_region(LORA_REGION region)

```
*******************************************************************************
 *   @brief        This API is used to set the region of LoRaWAN you want it to work in.
 *   @return       SUCCESS or FAIL
 *   @param        LORA_REGION region:     the region of LoRaWAN.
*******************************************************************************
```

3.17 rui_lora_get_status

```
/*******************************************************************************
 *   @brief        This API is used to get all status about LoRa.
 *   @return       SUCCESS or FAIL
 *   @param            RUI_LORA_STATUS_T *status:      the status about LoRa.
*******************************************************************************/
```

uint32_t rui_lora_get_status(RUI_LORA_STATUS_T *status);

3.18 rui_lora_autosend_callback

```
/*********************************************************************************************
 *      @brief      This API is auto send data timeout callback by lora.
 *      @return     none
 *      @param          none
*********************************************************************************************/
void rui_lora_autosend_callback(void);
```

3.18 rui_lorap2p_set_parameters
3.19 rui_lorap2p_send
3.20 rui_lorap2p_register_recv_callback

4. BLE

General format: **rui_ble_xxx()**
General definition:

```
typedef enum
{
    BLE_CLIENT_EVT_DISCOVERY_COMPLETE = 1,
    BLE_CLIENT_EVT_NOTIFICATION,
    BLE_CLIENT_EVT_READ
} BLE_CLIENT_EVT_TYPE;

typedef struct
{
  uint8_t       *p_data;
  uint16_t       len;
} BLE_DATA;

typedef struct
{
  uint16_t write_handle;
  uint16_t read_notify_handle;
  uint16_t notify_cccd_handle;
} BLE_HANDLE;

typedef struct
{
```

```
    BLE_CLIENT_EVT_TYPE    evt_type;
    uint16_t    conn_handle;
    union
    {
        BLE_DATA          data;
        BLE_HANDLE      peer_db;
    } params;
} BLE_CLIENT_EVT;

typedef void (* BLE_CLIENT_EVT_HANDLER) (BLE_CLIENT * p_ble_rcs_c,
BLE_CLIENT_EVT * p_evt);

/**@brief Rak Custom Service Client structure. */
struct BLE_CLIENT
{
    uint16_t     connect_handle;
    HANDLE_LIST     peer_list;
    BLE_CLIENT_EVT_HANDLER evt_handler;
    uint8_t       uuid_type;
};

typedef enum{
    BLE_MODE_PERIPHERAL = 0,
    BLE_MODE_CENTRAL,
    BLE_MODE_OBSERVER
}BLE_WORK_MODE;
```

4.1 rui_ble_scan_adv

```
void rui_ble_scan_adv(int8_t rssi_value, uint8_t *p_adv_data, uint16_t adv_data_len,
uint8_t *p_device_mac)
*********************************************************************************
 *   @brief        This API is used to scan all around BLE devices.
                   This API is valid when the BLE works on master mode.
 *   @return       NULL
 *   @param        int8_t rssi_value:            peripheral rssi value.
 *   @param        uint8_t *p_adv_data:              the advertise data.
 *   @param        uint16_t adv_data_len:    the length of advertise data.
 *   @param        uint8_t *p_device_mac:    the peripheral's MAC address.
*********************************************************************************
```

4.2 rui_ble_set_work_mode

```
/*********************************************************************************************
 *  @brief        This API is used to set the work mode of BLE.
 *  @return       SUCCESS or FAIL
 *  @param        BLE_WORK_MODE    mode:        BLE_MODE_PERIPHERAL,
BLE_MODE_CENTRAL, BLE_MODE_OBSERVER
 *  @param        long_range_enable: true or false
*********************************************************************************************/
uint32_t rui_ble_set_work_mode(BLE_WORK_MODE mode, bool long_range_enable);
```

4.3 rui_ble_rx_data_notify

```
void rui_ble_rx_data_notify(uint8_t *pdata, uint16_t len)
*********************************************************************************************
 *  @brief        This API is used to receive the notify data which come from other
devices
                  through BLE.
 *  @return       NULL
*   @param        uint8_t *pdata:    the receive data.
 *  @param        uint16_t len:      the length of receive data.
*********************************************************************************************
```

4.4 rui_ble_rx_data_read

```
void rui_ble_rx_data_read(uint8_t *pdata, uint16_t len)
*********************************************************************************************
 *  @brief        This API is used to receive the data which you just read from another
BLE
                  device through BLE.
 *  @return       NULL
 *  @param        uint8_t *pdata:    the receive data.
 *  @param        uint16_t len:           the length of receive data.
*********************************************************************************************
```

4.5 rui_ble_tx_data_write

```
uint32_t rui_ble_tx_data_write(BLE_CLIENT * p_ble_rcs_c, uint8_t *pdata, uint16_t len)
*********************************************************************************************
 *  @brief        This API is used to write a data to another BLE device through BLE.
```

```
 *    @return        SUCCESS or FAIL
 *    @param         BLE_CLIENT * p_ble_rcs_c:     The BLE client instance.
 *    @param          uint8_t *pdata:                The data which will be sent.
 *    @param          uint16_t len:                      The length of data.
*********************************************************************************************
```

4.6 rui_ble_tx_data_read

uint32_t rui_ble_tx_data_read(BLE_CLIENT * p_ble_rcs_c)

```
*********************************************************************************************
 *    @brief         This API is used to read another BLE device's data through BLE.
 *    @return        SUCCESS or FAIL.
 *    @param         BLE_CLIENT * p_ble_rcs_c:    The BLE client instance.
*********************************************************************************************
```

5. Sensor

General format: **rui_<sensor type>_xxx()**

5.1 rui_temperature_get

uint32_t rui_temperature_get(double *temp)

```
*********************************************************************************************
 *    @brief         This API is used to get the value of temperature from temperature
sensor.
 *    @return        SUCCESS or FAIL
 *    @param         double *temp:       the value of temperature
*********************************************************************************************
```

5.2 rui_temperature_set_mode

uint32_t rui_temperature_set_mode(DRIVER_MODE mode);

```
*********************************************************************************************
 *    @brief         This API is used to set the work mode of the temperature sensor.
 *    @return        SUCCESS or FAIL
```

```
 *   @param      DRIVER_MODE mode:     temperature sensor's work mode
*********************************************************************************
```

5.3 rui_humidity_get

uint32_t rui_humidity_get(double *humidity)

```
*********************************************************************************
 *   @brief      This API is used to get the value of humidity from humidity sensor.
 *   @return     SUCCESS or FAIL
 *   @param      double *humidity:       the value of humidity
*********************************************************************************
```

5.4 rui_humidity_set_mode

uint32_t rui_humidity_set_mode(DRIVER_MODE mode)

```
*********************************************************************************
 *   @brief      This API is used to set the work mode of the humidity sensor.
 *   @return     SUCCESS or FAIL
 *   @param      DRIVER_MODE mode:     humidity sensor's work mode
*********************************************************************************
```

5.5 rui_pressure_get

uint32_t rui_pressure_get(double *pressure)

```
*********************************************************************************
 *   @brief      This API is used to get the value of pressure from the pressure sensor.
 *   @return     SUCCESS or FAIL
 *   @param      double *pressure:       the value of pressure
*********************************************************************************
```

5.6 rui_pressure_set_mode

uint32_t rui_pressure_set_mode(DRIVER_MODE mode)

```
*********************************************************************************
```

```
 *   @brief        This API is used to set the work mode of the pressure sensor.
 *   @return       SUCCESS or FAIL
 *   @param        DRIVER_MODE mode:     the pressure sensor's work mode
*******************************************************************************************
```

5.7 rui_acceleration_get

uint32_t rui_acceleration_get(float *x, float *y, float *z)

```
*******************************************************************************************
 *   @brief        This API is used to get the value of acceleration, including X, Y, Z axis.
 *   @return       SUCCESS or FAIL
 *   @param        float *x, float *y, float *z:     the value of acceleration X, Y, Z axis.
*******************************************************************************************
```

5.8 rui_acceleration_set_mode

uint32_t rui_acceleration_set_mode(DRIVER_MODE mode)

```
*******************************************************************************************
 *   @brief        This API is used to set the work mode of the acceleration sensor.
 *   @return       SUCCESS or FAIL
 *   @param        DRIVER_MODE mode:    the acceleration sensor's work mode
*******************************************************************************************
```

5.9 rui_magnetic_get

uint32_t rui_magnetic_get(float *magnetic_x, float *magnetic_y, float *magnetic_z)

```
*******************************************************************************************
 *   @brief        This value is used to get the value of magnetic, including X, Y, Z axis.
 *   @return       SUCCESS or FAIL
 *   @param        float *magnetic_x, float *magnetic_y,
                   float *magnetic_z:       the value of magnetic X, Y, Z axis.
*******************************************************************************************
```

5.10 rui_magnetic_set_mode

uint32_t rui_magnetic_set_mode(DRIVER_MODE mode)

```
*******************************************************************************************
 *   @brief        This API is used to set the work mode of the magnetic sensor.
 *   @return       SUCCESS or FAIL
 *   @param        DRIVER_MODE mode:    the magnetic sensor's work mode
*******************************************************************************************
```

5.11 rui_gyroscope_get

uint32_t rui_gyroscope_get(float *gyroscope_x, float *gyroscope_y, float *gyroscope_z)

```
*******************************************************************************************
 *   @brief        This value is used to get the value of gyroscope, including X, Y, Z axis.
 *   @return       SUCCESS or FAIL
 *   @param        float *gyroscope_x, float *gyroscope_y,
                   float *gyroscope_z:    the value of gyroscope X, Y, Z axis.
*******************************************************************************************
```

5.12 rui_gyroscope_set_mode

uint32_t rui_gyroscope_mode_set(RUI_DRIVER_MODE mode)

```
*******************************************************************************************
 *   @brief        This API is used to set the work mode of the gyroscope sensor.
 *   @return       SUCCESS or FAIL
 *   @param        RUI_DRIVER_MODE mode:     the gyroscope sensor's work mode.
*******************************************************************************************
```

5.13 rui_light_get_strength

uint32_t rui_light_get_strength(float *light_strength)

```
*******************************************************************************************
 *   @brief        This API is used to get the value of light strength.
 *   @return       SUCCESS or FAIL
 *   @param        float *light_strength:    the value of light strength.
```

```
*******************************************************************************************
```

5.14 rui_light_set_mode

uint32_t rui_light_set_mode(DRIVER_MODE mode)

```
*******************************************************************************************
 *   @brief        This API is used to set the work mode of the light sensor.
 *   @return       SUCCESS or FAIL
 *   @param        DRIVER_MODE mode:    the light sensor's work mode
*******************************************************************************************
```

5.15 rui_gps_get

uint32_t rui_gps_get(RUI_GGA_Data *gps_data)

```
*******************************************************************************************
 *   @brief        This API is used to get the GPS data from GPS module.
 *   @return       SUCCESS or FAIL
 *   @param        RUI_GGA_Data *gps_data:      the GPS data, GPGGA format.
*******************************************************************************************
typedef struct GGAData
{
    uint8_t UTC[4];
    uint8_t Date[3];
    bool LatitudeNS;                //0:N1:S
    uint8_t LatitudeDegree;
    uint32_t LatitudeMinute;
    double Latitude;
    bool LongitudaEW;               //0:E1:W
    uint8_t LongitudaDegree;
    double Longitude;
    uint32_t LongitudaMinute;
    int Quality;
    int NumSatellities;
    int16_t HDOP;
    int16_t Altitude;
    int16_t GeoidHeight;
}RUI_GGA_Data;
```

5.16 rui_gps_set_mode

uint32_t rui_gps_set_mode(DRIVER_MODE mode)

```
*****************************************************************************************
 *   @brief        This API is used to set the work mode of the GPS module.
 *   @return       SUCCESS or FAIL
 *   @param        DRIVER_MODE mode:     the GPS module's work mode.
*****************************************************************************************
```

5.17 rui_voltage_get

uint32_t rui_voltage_get(float *voltage)

```
*****************************************************************************************
 *   @brief        This API is used to get the current voltage value of the battery.
 *   @return       SUCCESS or FAIL
 *   @param        float *voltage:       the current voltage value of the battery
*****************************************************************************************
```

5.18 rui_voltage_set_mode

uint32_t rui_voltage_set_mode(DRIVER_MODE mode)

```
*****************************************************************************************
 *   @brief        This API is used to set whether to collect the voltage data of the battery.
 *   @return       SUCCESS or FAIL
 *   @param        DRIVER_MODE mode:     only two mode are valid for this parameter:
                                         RUI_POWER_ON_MODE means it will collect voltage
data.
                                         RUI_POWER_OFF_MODE means it won't collect voltage
                                         data.
*****************************************************************************************
```

6. Interface

General format: **rui_<Interface type>_xxx()**
General definition:

```
typedef enum RUI_UART_DEF
{
	RUI_UART0 = 0,
	RUI_UART1 = 1,
	RUI_UART2 = 2,
	RUI_UART3 = 3
}RUI_UART_DEF;

Typedef enum RUI_IF_READ_WRITE
{
	RUI_IF_READ = 0,
	RUI_IF_WRITE = 1
}RUI_IF_READ_WRITE;

typedef struct RUI_I2C_ST{
	uint32_t INSTANCE_ID;
	uint32_t PIN_SDA;	// SDA pin num
	uint32_t PIN_SCL;	// SCL pin num
	uint32_t FREQUENCY;
}RUI_I2C_ST;

typedef enum RUI_I2C_FREQ_ST{
	RUI_I2C_FREQ_100K,
	RUI_I2C_FREQ_400K
}RUI_I2C_FREQ_ST;

typedef struct{
	uint32_t INSTANCE_ID;
	uint32_t PIN_CS;	//
	uint32_t PIN_SCL;	//
	uint32_t PIN_MISO;	//
	uint32_t PIN_MOSI;	//
	uint32_t FREQUENCY;
} RUI_SPI_ST;

typedef enum RUI_SPI_FREQ_ST{
	RUI_I2C_FREQ_125K,
	RUI_I2C_FREQ_250K,
	RUI_I2C_FREQ_500K,
	RUI_I2C_FREQ_1M,
	RUI_I2C_FREQ_2M,
	RUI_I2C_FREQ_4M,
	RUI_I2C_FREQ_8M
}RUI_SPI_FREQ_ST;
```

```
typedef enum
{
    RUI_GPIO_PIN_DIR_INPUT, // Input.
    RUI_GPIO_PIN_DIR_OUTPUT // Output.
} RUI_GPIO_PIN_DIR_ST;

typedef enum
{
    RUI_GPIO_PIN_NOPULL,     //   Pin pull-up resistor disabled.
    RUI_GPIO_PIN_PULLDOWN, //   Pin pull-down resistor enabled.
    RUI_GPIO_PIN_PULLUP      //   Pin pull-up resistor enabled.
} RUI_GPIO_PIN_PULL_ST;

typedef struct{
    uint32_t pin_num;
    RUI_GPIO_PIN_DIR_ST dir;
    RUI_GPIO_PIN_PULL_ST pull;
}RUI_GPIO_ST;


typedef enum
{
    RUI_TIMER_MODE_SINGLE_SHOT,      /**< The timer will expire only once. */
    RUI_TIMER_MODE_REPEATED          /**< The timer will restart each time it
expires. */
} RUI_TIMER_MODE_ST;

typedef void (* TIMER_CALLBACK) (void);

typedef struct{
    uint32_t timeout_ms;
    RUI_TIMER_MODE_ST timer_mode;
    TIMER_CALLBACK timer_callback;
}RUI_TIMER_ST;
```

6.1 rui_uart_init

```
/**********************************************************************************************
 * @brief          This API is used to configure the parameters for uart.
 * @return         SUCCESS or FAIL
 * @param           RUI_UART_DEF uart_def:the instance of uart.
                    RUI_UART_BAUDRATE baudrate:Uart baudrate value.
```

**********/
uint32_t rui_uart_init(RUI_UART_DEF uart_def,RUI_UART_BAUDRATE baudrate);

6.2 rui_uart_uninit

```
/**********
 * @brief          This API is used to uninit uart.
 * @return         SUCCESS or FAIL
 * @param          RUI_UART_DEF uart_def:the instance of uart.
**********/
uint32_t rui_uart_unint(RUI_UART_DEF uart_def);
```

6.3 rui_uart_send

```
uint32_t rui_uart_send(RUI_UART_DEF uart_def, uint8_t *pdata, uint16_t len)
**********
 *   @brief          This API is used to send data via uart.
 *   @return         SUCCESS or FAIL
 *   @param          RUI_UART_DEF uart_def:      the instance of uart.
 *   @param          uint8_t *pdata:             the pointer of data you want to send.
 *   @param          uint16_t len:               the length of data.
**********
```

6.4 rui_uart_recv

```
void rui_uart_recv(RUI_UART_DEF uart_def, uint8_t *pdata, uint16_t len)
**********
 *   @brief          This API is used to receive data from uart.
 *   @return         NULL
 *   @param          RUI_UART_DEF uart_def:      the instance of uart.
 *   @param          uint8_t *pdata:             the pointer of data.
 *   @param          uint16_t len:               the length of data. This value is always
1.
**********
```

6.5 rui_uart_mode_config

```
/**********
 * @brief          This API is used to configure uart work mode.
```

```
 * @return          SUCCESS or FAIL
 * @param           RUI_UART_DEF uart_def:the instance of uart.
                    RUI_UART_MODE uart_mode: value for
RUI_UART_NORAMAL,RUI_UART_UNVARNISHED.
*******************************************************************************/
uint32_t rui_uart_mode_config(RUI_UART_DEF uart_def,RUI_UART_MODE
uart_mode) ;
```

6.6 rui_gpio_init

```
/*******************************************************************************
 *      @brief      This API is used to configure gpio.
 *      @return     SUCCESS or FAIL
 *      @param      RUI_GPIO_ST *rui_gpio:the instance of gpio.
*******************************************************************************/
uint32_t rui_gpio_init(RUI_GPIO_ST *rui_gpio);
```

6.7 rui_gpio_uninit

```
/*******************************************************************************
 *      @brief      This API is used to deinit gpio.
 *      @return     SUCCESS or FAIL
 *      @param      RUI_GPIO_ST *rui_gpio:the instance of gpio.
*******************************************************************************/
void rui_gpio_uninit(RUI_GPIO_ST *rui_gpio);
```

6.8 rui_gpio_rw

```
/*******************************************************************************
 *  @brief      This API is used to read or set a certain gpio's status.
 *  @return     SUCCESS or FAIL
 *  @param      RUI_IF_READ_WRITE rw: read or set.
                RUI_GPIO_ST *rui_gpio: the stru of gpio which you want to read or set.
                uint8_t* status:    the pointer of gpio value.
                                    0: low level, 1: high level.
*******************************************************************************/
uint32_t rui_gpio_rw(RUI_IF_READ_WRITE rw_status,RUI_GPIO_ST *rui_gpio,uint8_t*
status);
```

6.9 rui_timer_init

```
/**************************************************************************************
 *   @brief        This API is used to creat and init a timer.
 *   @return       SUCCESS or FAIL
 *   @param            void *obj:timer instance.
                       void ( *callback )( void ):timer event callback function.
**************************************************************************************/
uint32_t rui_timer_init(void *obj, void ( *callback )( void ));
```

6.10 rui_timer_uninit

```
/**************************************************************************************
 *   @brief        This API is used to uninit a timer.
 *   @return       SUCCESS or FAIL
 *   @param         void *obj:timer instance.
**************************************************************************************/
uint32_t rui_timer_uninit(void *obj);
```

6.11 rui_timer_setvalue

```
/**************************************************************************************
 *   @brief        This API is used to set value for timer.
 *   @return       SUCCESS or FAIL
 *   @param            void *obj:timer instance.
                       uint32_t value: timer value.
**************************************************************************************/
uint32_t rui_timer_setvalue(void *obj, uint32_t value );
```

6.12 rui_timer_start

```
/**************************************************************************************
 *   @brief        This API is used to start timer.
 *   @return       SUCCESS or FAIL
 *   @param            void *obj:timer instance.
**************************************************************************************/
```

```
uint32_t rui_timer_start(void *obj);
```

6.13 rui_timer_stop

```
/**********************************************************************************************
 *   @brief         This API is used to stop timer.
 *   @return        SUCCESS or FAIL
 *   @param             void *obj:timer instance.
**********************************************************************************************/
uint32_t rui_timer_stop(void *obj);
```

6.14 rui_adc_init

```
/**********************************************************************************************
 *         @brief         This API is used to init ADC
 *         @return        SUCCESS or FAIL
 *         @param             RUI_GPIO_ST *rui_gpio:the instance of gpio.
**********************************************************************************************/
uint32_t rui_adc_init(RUI_GPIO_ST *rui_gpio);
```

6.15 rui_adc_uninit

```
/**********************************************************************************************
 *         @brief         This API is used to uninit ADC
 *         @return        SUCCESS or FAIL
 *         @param             RUI_GPIO_ST *rui_gpio:the instance of gpio.
**********************************************************************************************/
uint32_t rui_adc_uninit(RUI_GPIO_ST *rui_gpio);
```

6.16 rui_adc_read

```
/**********************************************************************************************
 *   @brief         This API is used to read the value of ADC pin
 *   @return        SUCCESS or FAIL
 *   @param           uint8_t* value:   the value which is read from ADC.
                        RUI_GPIO_ST *rui_gpio : gpio stru
```

```
*****************************************************************************************/
uint32_t rui_adc_get(RUI_GPIO_ST *rui_gpio, uint16_t* value);
```

6.17 rui_i2c_init

```
/*****************************************************************************************
 *   @brief        This API is used to init I2C.
 *   @return       SUCCESS or FAIL
 *   @param          RUI_I2C_ST *rui_i2c：init parameters struct.
*****************************************************************************************/
uint32_t rui_i2c_init(RUI_I2C_ST *rui_i2c);
```

6.18 rui_i2c_rw

```
uint32_t rui_i2c_rw(RUI_IF_READ_WRITE rw, uint8_t devAddr, uint16_t regAddr, uint8_t*
data, uint16_t len)
*****************************************************************************************
 *   @brief        This API is used to read/write through I2C.
 *   @return       SUCCESS or FAIL
 *   @param        RUI_IF_READ_WRITE rw:       read or write through I2C.
                   uint8_t devAddr:            device address, in Hex format.
                   uint16_t regAddr:           the sensor's register address, in Hex
format.
                   uint8_t* data:              the data read out or will be write through iic.
                   uint16_t len:               the data length. the unit is byte.
*****************************************************************************************
```

6.19 rui_spi_init

```
/*****************************************************************************************
 *        @brief        This API is used to init SPI.
 *        @return       SUCCESS or FAIL
 *        @param          RUI_SPI_ST *rui_spi：init parameters struct.
*****************************************************************************************/
uint32_t rui_spi_init(RUI_SPI_ST *rui_spi);
```

6.20 rui_spi_rw

```
/*******************************************************************************************
 *      @brief      This API is used to read/write through SPI.
 *      @return     SUCCESS or FAIL
 *      @param      uint8_t *tx:        write through spi.
                    uint16_t tx_len:    tx length.
                    uint8_t *rx:        read buffer.
                    uint16_t rx_len:    rx length.
*******************************************************************************************/
uint32_t rui_spi_rw(uint8_t *tx, uint16_t tx_len, uint8_t *rx, uint16_t rx_len);
```

6.21 rui_delay_ms

```
/*******************************************************************************************
 *  @brief      This API is used to delay time (unit:ms).
 *  @return     SUCCESS or FAIL
 *  @param      uint32_t value:delay time value.
*******************************************************************************************/
uint32_t rui_delay_ms( uint32_t value );
```

6.22 rui_delay_us

```
/*******************************************************************************************
 *  @brief      This API is used to delay time (unit:us).
 *  @return     SUCCESS or FAIL
 *  @param          uint32_t value:delay time value.
*******************************************************************************************/
uint32_t rui_delay_us( uint32_t value );
```

6.23 rui_init

```
/*******************************************************************************************
 *      @brief      This API is used to init system.
 *      @return     none
 *      @param      none
*******************************************************************************************/
void rui_init(void);
```

6.24 rui_running

```
/*********************************************************************************************
 *      @brief      This API is used to run preset tasks, including low-power processing.
 *      @return     none
 *      @param      none
*********************************************************************************************/
void rui_running(void);
```

6.25 RUI_LOG_PRINTF

```
/*********************************************************************************************
 *      @brief      This API is used to print log.
 *      @return     none
 *      @param      custom
*********************************************************************************************/
void RUI_LOG_PRINTF (fmt, args...);
```

7. Device

General format: **rui_device_xxx()**

7.1 rui_device_version

```
uint32_t rui_device_version(uint8_t *version)

*********************************************************************************************
 *  @brief      This API is used to get the current firmware version.
 *  @return     SUCCESS or FAIL
 *  @param      uint8_t *version:   the current firmware version.
*********************************************************************************************
```

7.2 rui_device_reset

uint32_t rui_device_reset(void)

```
*******************************************************************************************
 *   @brief        This API is used to reset the device.
 *   @return       SUCCESS or FAIL
 *   @param        void
*******************************************************************************************
```

7.3 rui_device_sleep

void rui_device_sleep(uint32_t period)

```
*******************************************************************************************
 *   @brief        This API is used to let the device go to sleep mode.
 *   @return       NULL
 *   @param        uint32_t period:    sleep period
*******************************************************************************************
```

7.4 rui_device_boot

void rui_device_boot()

```
*******************************************************************************************
 *   @brief        This API is used to let the device go to boot mode.
 *   @return       NULL
 *   @param        NULL
*******************************************************************************************
```