RAK811 Lora AT Command User Guide V1.5

Content

1 Description

1.1 View

RAK811 Low-Power Long Range LoRa Technology Transceiver module, provides an easy to use, small size, low-power solution for long range wireless data transmission.

First, The RAK811 module complies with the latest LoRaWAN Class A & C protocol specifications, it is simple to access LWPA IOT platforms, such Actility etc. Second, it also support Lora Point to Point communications, this function can help customers implement their own private long range Lora network fast.

Module integrates semtech SX1276 and stm32L, offer user an serials At commands with UART Interface .It is easy to accomplish their applications, such as simple long range sensor data applications with external host MCU, low-power feature is suitable for battery applications.

1.2 Features

- Long Range LoraWAN operating in the LoraWAN 1.0.2 Regions frequency bands
- Lora Point to Point communication in the 860MHz-929.9MHz frequency
- Small size and low power
- High Receiver Sensitivity: down to -146 dBm
- TX Power: adjustable up to +14 dBm high efficiency PA, max PA boost up to 20dbm
- FSK, GFSK, and LoRa Technology modulation
- IIP3 = -11 dBm
- Up to 15 km coverage at suburban and up to 5 km coverage at urban area

1.3 Applications

- Automated Meter Reading
- Home and Building Automation
- Wireless Alarm and Security Systems
- Industrial Monitoring and Control
- Machine to Machine (M2M)
- Internet of Things (IoT)

2 Interface

2.1 UART

The default uart setting is as follows:

Baudrate---------- 115200

Data bits---------- 8

Stop bits---------- 1

Parity---------- NONE

Flow Control---- DISABLE

The baud rates supported by the RAK811 module are as follows:

9600 bps

19200 bps

38400 bps

57600 bps

115200 bps

230400 bps

460800 bps

921600 bps

2.2 Power On

Power up or Reset the module, the following message is transmitted from the module.

Welcome to RAK811\r\n

3 AT Command

3.1 Commands

- **Format**

  All commands are begin with <at+> and end with <CR><LF>.
  All response are end with <CR><LF>.

- **Error Code**

Name	Code
CODE_ARG_ERR	-1
CODE_ARG_NOT_FIND	-2
CODE_JOIN_ABP_ERR	-3
CODE_JOIN_OTAA_ERR	-4
CODE_NOT_JOIN	-5
CODE_MAC_BUSY_ERR	-6
CODE_TX_ERR	-7
CODE_INTER_ERR	-8
CODE_WR_CFG_ERR	-11
CODE_RD_CFG_ERR	-12
CODE_TX_LEN_LIMITE_ERR	-13
CODE_UNKNOWN_ERR	-20

- **Event Code**

Name	Code
STATUS_RECV_DATA	0
STATUS_TX_COMFIRMED	1
STATUS_TX_UNCOMFIRMED	2
STATUS_JOINED_SUCCESS	3
STATUS_JOINED_FAILED	4
STATUS_TX_TIMEOUT	5
STATUS_RX2_TIMEOUT	6
STATUS_DOWNLINK_REPEATED	7
STATUS_WAKE_UP	8
STATUS_P2PTX_COMPLETE	9
STATUS_UNKNOWN	100

■ Command List

Class	Command	Description
System	at+version	Get module version
	at+sleep	enter sleep mode
	at+reset=<mode>	Reset Module or LoRaWAN stack 0: Reset and restart module 1: Reset LoRaWAN stack and Module will reload LoRa configuration from EEPROM
	at+reload	Set LoraWAN or LoraP2P configurations to default
	at+mode[=<mode>]	Get/Set Module work on LoraWAN or LoraP2P mode, default 0. 0: LoraWAN Mode 1: LoraP2P Mode
	at+recv_ex[=<enable>]	Enable or Disable，at+recv command can add rssi,snr message，default 0，the parameter don't save，So reset module need set again. 0: Disable 1: Enable
LoraWAN	at+set_config=<key>:<value>[&<key>:<value>][&<key>:<value>]...	Set LoraWAN configurations, Keys as follows dev_addr, dev_eui, app_eui, app_key, nwks_key, apps_key, pwr_level, adr, dr, public_net, rx_delay1, rx2, ch_list, ch_mask, max_chs, join_cnt, nbtrans, class, duty
	at+get_config=<key>	Get LoraWAN configurations, follow above
	at+band[=<value>]	Get LoraWAN 1.0.2 region.(Set region function Supported after version 2.0.3.0)
	at+join=<mode>	join the configured LoraWAN network otaa : Over-The-Air Activation abp : Activation By personalization
	at+signal	Check the radio rssi, snr, update by latest received radio packet
	at+dr[=<dr>]	Get/Set the next send data rate

	at+link_cnt[=<uplinkCount>,<downlinkCount>]	Get/Set LoraWAN up/downlink Counter, 32bit Output in decimal formal
	at+abp_info	Query the relevant information,if use OTAA way to join the Network success. This relevant information can be use to ABP way to join Network. Return NetworkID, DevAddr, Nwkskey, Appskey. Below paramenter output in hexadecimal <NetworkID>: NetworkID 4Byte <DevAddr>: Network address 4Byte <Nwkskey>: Network session key 16Byte <Appskey>: Application session key 16Byte
	at+send=<type>,<port>,<data>	Send data to LoraWAN network <type> 0 : send unconfirmed packets 1 : send confirmed packets <port> 1-223: port number from 1 to 223 <data> hex value (no space)
	at+recv=<status>,<port>[,<rssi>][,<snr>],<len>[,<data>]	Receive Event and Data from LoraWAN or LoraP2P network <status >: see the Event code table <port>: LoraWAN application port, 0 when Event and LoraP2P <len>: LoraWAN or LoraP2P receive data len, max 64 <data>: LoraWAN or LoraP2P receive data, hex value (no space) , Null when Event <rssi> : Signal strength，at+recv_ex enable display <snr> : Signal to noise ratio，at+recv_ex enable display （only receive the data can effective）
	at+rf_config[=<freq>, <sf>, <bw>, <cr>, <prlen>, <pwr>]	Get/Set the p2p txd and rxd used RF parameters. <freq>: frequency, default 868100000

LoraP2P		(860000000 ~9299000000) <sf>: spread factor, default 12 (6-12) <bw>: Band-with, default 0 (0:125KHz, 1:250KHz, 2:500KHz) <cr>: coding Rate, default 1 (1:4/5, 2:4/6, 3:4/7, 4:4/8) <prlen>: Preamlen default 8 (8-65535) <pwr>: Tx power default 20 (5-20)
	at+txc =<cnts>,<interval>, <datahex>	Set the Lora tx continue : <cnts>: tx counts (1-65535) <interval>: tx interval between last send success or fail, (10-3600000ms) <datahex> hex value (no space) , max 64
	at+rxc=<report_en>	Set the Lora rx continue <report_en>: enable report to host or not
	at+tx_stop	Stop the Lora tx continue
	at+rx_stop	Stop the Lora rx continue
Radio	at+status[=0]	Get/Clear the radio statistics Null: Response TxSuccessCnt, TxErrCnt, RxSuccessCnt, RxTimeOutCnt, RxErrCnt, Rssi, Snr =0: Clear statistics
Peripheral	at+uart [=<baud>,<data_bits>, <parity>,<stop_bits>, <flow_ctrl>]	Get/Set UART configurations <baud>: (9600-921600) Supports baud list <data_bits>: (8) 8 data bits <parity>: (0/1/2) 0: PARITY_NONE 1: PARITY_EVEN 2: PARITY_ODD <stop_bits>:(1/2) 0: Stop bit length is 1 bit 1: Stop bit length is 2 bit <flow_ctrl>:(0/1) 0: disable flow control 1: enable flow control

	at+rd_reg=<reg_addr>[,<len>]	Read LoRa chip SX1276 register value <reg_addr>: register address ,can see SX1276 datasheet，Enter hex format <len>: Optional，default 1，Max 64 Return value display in hex format
	at+wr_reg=<reg_addr>,<len>,<databuf>	Modify the LoRa chip SX1276 register value <reg_addr>: register address,can see the SX1276 datasheet，Enter hex format. <len>: Max 64 <databuf>: Written data，Enter hex format，Max 64 Hex values.
	at+gpio=<pin>[,<level>]	Read or set assign PIN high and low electrical level <pin>: Refer the pin definition <level>: 1 Output high 0 Output low
	at+rd_adc=<pin>	Read assign ADC PIN's voltage value，ADC precision is12bit, Range 0-4096 <pin>: Refer the pin definition
	at+rd_iic=<dev_addr>,<data_addr>[,<len>]	Read assign I2C PIN's connect device register value, I2C frequency default 100KHz <dev_addr>: I2C device MAC 7-bit mode，Enter hex format, 8bit <data_addr>: I2C device register address，Enter hex format, 16bit <len>: Optional, default 1, Max 64 Return value display in hex format
	at+wr_iic=<dev_addr>,<data_addr>,<databuf>	Write data to assign I2C PIN's connect device register value，I2C frequency default 100KHz <dev_addr>: I2C device MAC 7-bit mode，Enter hex format. <data_addr>: I2C device register address，Enter hex format <databuf>: Enter hex format，Max 64 Hex value.

3.2 System

3.2.1 at+version

- **Command**

  at+version\r\n

- **Description**

  Get software version

- **Parameter**

  NULL

- **Response**

  OK<version>\r\n

  <version> = string representing current software version

  xx.xx.xx.xx mean major . customer. function . bug , versions

- **Event Response**

  NULL

3.2.2 at+sleep

- **Command**

  at+sleep\r\n

- **Description**

  Command is used to make module enter sleep mode with ultra-low power consumption.

  After device enters in sleep mode, host MCU can send any character to wake up it, when module is awake, **Event response** will be received.

- **Parameter**

  NULL

- **Response**

  OK\r\n----------Enter sleep mode successfully

  ERROR<code>\r\n

- **Event Response**

  at+recv=<status>,0,0\r\n

<status> = *STATUS_WAKE_UP* --------Module is awake

3.2.3 at+reset

- **Command**

  at+reset=<mode>\r\n

- **Description**

  Reset Module or LoRaWAN or LoraP2P stack

- **Parameter**

  <mode> = 0 Reset and restart module

  = 1 Reset LoRaWAN or LoraP2P stack and Module will reload LoRa configuration from EEPROM

- **Response**

  OK\r\n----------Reset successfully

  - Event Response

  NULL

3.2.4 at+reload

- **Command**

  at+reload\r\n

- **Description**

  Reload the default parameters of LoraWAN or LoraP2P setting

- **Parameter**

  NULL

- **Response**

  OK\r\n----------Reload successfully

  ERROR<code>\r\n

- **Event Response**

  NULL

3.2.5 at+mode

- **Command**

  at+mode[=<mode>]\r\n

■ **Description**

Get/Set Module work on LoraWAN or LoraP2P mode, default 0.

■ **Parameter**

<mode> = 0 LoraWAN Mode (default mode)

= 1 LoraP2P Mode

■ **Response**

OK\r\n----------Reset successfully

ERROR<code>\r\n

■ **Event Response**

NULL

3.2.6 at+recv_ex

■ **Command**

at+recv_ex[=<enable>]\r\n

■ **Description**

Set enable or disable， if enable, the at+recv command can receive rssi, snr message， default 0.

The parameter can not save. so if you reset the module, you should set again.

■ **Parameter**

<enable> = 0 disable (default)

= 1 enable

■ **Response**

OK\r\n----------set success

OK0\r\n----------query return

ERROR<code>\r\n

■ **Event Response**

NULL

3.3 LoraWAN

3.3.1 at+set_config

- **Command**

at+set_config=<key>:<value>[&<key>:<value>][&<key>:<value>]...\r\n

- **Description**

Set LoRa Configuration to flash and it will be available next Reset time or next join command set time .

- **Parameter**

  - *< dev_addr >:<address>*

    <address>----------4 bytes hex number representing the device address from 00000001 – FFFFFFFE

    This command configures the module with a 4-byte unique network device address <address>. The <address> MUST be UNIQUE to the current network. This must be directly set solely for activation by personalization devices. This parameter must not be set before attempting to join using over-the-air activation because it will be overwritten once the join process is over.

  - *<dev_eui>:<eui>*

    <eui>---------- 8-byte hexadecimal number representing the device EUI

    This command sets the globally unique device identifier for the module and the default value is derived from MCU's UUID.

  - *<app_eui>:<eui>*

    <eui>----------8-byte hexadecimal number representing the application EUI

    The AppEUI is a global application ID in IEEE EUI64 address space that uniquely identifies the entity able to process the JoinReq frame.
    The AppEUI is stored in the end-device before the activation procedure is executed.

  - *<app_key>:<key>*

    <key>----------16-byte hexadecimal number representing the application key

    The AppKey is an AES-128 root key specific to the end-device.1 Whenever an end-device joins a network via over-the-air activation, the AppKey is used to derive the session keys NwkSKey and AppSKey specific for that end-device to encrypt and verify network communication and application data.

- *<nwks_key>:<key>*

  <key>----------16-byte hexadecimal number representing the network session key

  The NwkSKey is a network session key specific for the end-device. It is used by both the network server and the end-device to calculate and verify the MIC (message integrity code) of all data messages to ensure data integrity. It is further used to encrypt and decrypt the payload field of a MAC-only data messages.

- *<apps_key>:<key>*

  <key>---------- 16-byte hexadecimal number representing the application session key

  The AppSKey is an application session key specific for the end-device. It is used by both the application server and the end-device to encrypt and decrypt the payload field of application-specific data messages. Application payloads are end-to-end encrypted between the end-device and the application server, but they are not integrity protected. That means, a network server may be able to alter the content of the data messages in transit. Network servers are considered as trusted, but applications wishing to implement end-to-end confidentiality and integrity protection are recommended to use additional end-to-end security solutions, which are beyond the scope of this specification.

- *<tx_power>:<dbm>*

  <dbm>---------- LoRaWAN Tx Power [deprecated V2.0.2.0+]

  This command sets the output power to be used on the next transmissions. Refer to the LoRaWAN™ Specification for the output power list.
  EU868: {20, 14, 11, 8, 5, 2} dbm
  US915: { 20, 18, 16, 14, 12, 10} dbm

- *<pwr_level >: 0 - 15*

  Base on LoraWAN 1.0.2 This command sets the default tx power level , the true *power dbm is Max EIRP - 2* pwr_level .*

  The setting level depend on Regions define.

  For example EU868: EU868_DEFAULT_MAX_EIRP 16.0f
  EU868_MAX_TX_POWER TX_POWER_0
  EU868_MIN_TX_POWER TX_POWER_7

- *<adr>:<status>*

  <status>---------- string value representing the state, either "on" or "off".

This command sets if the adaptive data rate (ADR) is to be enabled or disabled. The server is informed about the status of the module's ADR in every uplink frame it receives from the ADR field in uplink data packet. If ADR is enabled, the server will optimize the data rate and the transmission power of the module based on the information collected from the network.

- *<dr>:<data rate>*

  <data rate>----------decimal number representing the data rate,but within the limits of the data rate range for the defined channels. See Regions define

This command sets the default tx data rate .

- < public_net >:<status>

  <status>---------- *string value representing the state, either "on" or "off".*

This command will set the LoRaWAN Sync word, if status is "on", Sync word = 0x34, else Sync word = 0x12(LoraP2P use it)

- < rx_delay1 >:<delay>

  <delay>----------*decimal number representing the delay between the transmission and the first Reception window in milliseconds, from 0 to 65535.*

This command will set the delay between the transmission and the first Reception window to the <rxDelay> in milliseconds. The delay between the transmission and the second Reception window is calculated in software as the delay between the transmission and the first Reception window + 1000 (ms).

- < ch_list >:<id>,<status>,<frequency>,<dr_min>,<dr_max>

  limit ***See Regions define***

  <id>----------decimal number representing the channel number.

  <status>----------set the channel on or off, ascii "on" or "off"

  <frequency**>**----------customer band list setting .

  <dr_min>----------decimal number representing the minimum tx data rate range.

  <dr_max>----------decimal number representing the maximum tx data rate range.

- <ch_mask>:<id>,<mask>

The ch_mask can configure fast choose which channels will be used as uplink channel by default.

<id>: id=0, channel 0-15 for region, id = 1 means 16-31 ,etc ...

<mask>: as hex, 0003 means use channel 0, 1 .

For example Region US915_CH0-7, should set id 0 mask 00FF, id 1 mask 0, id 2 mask 0, id 3 mask 0, id 4 mask 0

- <max_chs>:

Only read, return the max channels the used region defined

- < rx2 >: <dataRate>,<frequency> limit *See Regions define*

<dataRate>---------- *decimal number representing the rx data rate,*

<frequency>----------*decimal number representing the frequency,*

- < retrans >:<number>

<number>----------*decimal number representing the number of retransmissions for an uplink confirmed packet, from 1 to 255,default 8*

- <join_cnt>:<number>

The join cnt set the join OTAA times , limit *See Regions define*

- <nbtrans>:<number>

NbTrans is the number of transmissions for each uplink message. This applies only to “unconfirmed” uplink frames. The default value is 1 corresponding to a single transmission of each frame. The valid range is 1-15.

- < duty>:<string>

<string>----------off

on

- <class>:<number>
  <number>----------0 :ClassA default 0
  1 : ClassB (temporary does not support)
  2 : classC

■ **Response**

OK\r\n----------Save successfully

ERROR<code>\r\n

<code> : CODE_ARG_ERR----------Argument is invalid

■ **Event Response**

NULL

3.3.2 at+get_config

■ **Command**

at+get_config=<key>\r\n

■ **Description**

Get LoRa Configuration from flash via parameter key

■ **Parameter**

- < dev_addr >
- < dev_eui >
- < app_eui >
- < app_key >
- < nwks_key >
- < apps_key >
- < tx_power > [deprecated]
- <pwr_level>
- < adr >
- < dr >
- < public_net >
- < rx_delay1 >
- < ch_list >

- <ch_mask>
- <max_chs>
- < rx2 >
- <join_cnt>
- <nbtrans>
- <retrans>
- <class>
- < duty>

■ **Response**

OK\r\n----------Save successfully

ERROR<code>\r\n

<code> : CODE_ARG_ERR----------Argument is invalid

CODE_ARG_NOT_FIND--------*Argument is not available*

■ **Event Response**

NULL

3.3.3 at+band

■ **Command**

at+band[=<value>]\r\n

■ **Description**

Get/Set the LoraWAN region , general one firmware only support one region.(Set region function Supported after version 2.0.3.0, Default EU868)

■ **Parameter**

<value> Support AS923, EU868, AU915, US915, IN865, KR920

■ **Response**

If send at+band\r\n:

OKEU868\r\n

Or

OKUS915\r\n

If send at+band=EU868\r\n

OK\r\n

3.3.4 at+join

- **Command**

  at+join=<mode>\r\n

- **Description**

  Activation of the module can be achieved in two ways, either via Over-The-Air Activation (*<mode>=otaa*) when an end-device is deployed or reset, or via Activation By personalization (*<mode>=abp*) in which the two steps of end-device personalization and activation are done as one step.

  If *<mode>=otaa*, the configuration of *dev_eui*, *app_eui, app_key* Must be available.

  If *<mode>=abp*, the configuration of *dev_addr, nwks_key, apps_key* Must be available.

  See set lora config command

- **Parameter**

  <mode> = otaa over the air activation

  = abp activation by personalization

- **Response**

  OK\r\n----------Start to join network

  ERROR<code>\r\n ----------Can't join network

  <code> = *CODE_ARG_ERR*

  CODE_JOIN_ABP_ERR

  CODE_JOIN_OTAA_ERR

- **Event Response**

  If the way of join is *otaa*, event will be received.

  at+recv=<status>,0,0\r\n

  <status> = STATUS_JOINED_SUCCESS -------Join procedure was successful

  STATUS_JOINED_FAILED -------Join procedure was failed

  STATUS_RX2_TIMEOUT -------Join procedure was timeout , gateway not response

3.3.5 at+signal

- **Command**

  at+signal\r\n

- **Description**

  Get the signal from the Lora gateway or base station, via last receive packet.

- **Parameter**

  NULL

- **Response**

  OK<rssi><snr>\r\n

  <rssi> Received Signal Strength Indication (dbm), this is a negative number, larger show the signal is better.

  <snr> Signal to noise ratio (db), larger show the signal is better.

- **Event Response**

  NULL

3.3.6 at+dr

- **Command**

  at+dr[=<dr>]\r\n

- **Description**

  Use to change the next send data rate temporary when adr function is off.

  It will not be save to internal flash.

- **Parameter**

  *<dr> : **See Regions define***

- **Response**

  OK\r\n

- **Event Response**

  NULL

3.3.7 at+link_cnt

- **Command**

at+link_cnt[=<uplinkCount>,<downlinkCount>]\r\n

- **Description**

Get/Set LoraWAN Up/Downlinkcount value.

- **Parameter**

NULL

- **Response**

OK<upcnt>,<downcnt>\r\n

3.3.8 at+abp_info

- **Command**

at+abp_info\r\n

- **Description**

Query the relevant information,if use OTAA way to join the Network success. This relevant information can be use to ABP way to join Network. Return NetworkID, DevAddr, Nwkskey, Appskey.

- **Parameter**

NULL

- **Response**

OK<NetworkID>,<DevAddr>,<Nwkskey>,<Appskey>\r\n

Below paramenter output in hexadecimal

<NetworkID>: NetworkID 4Byte

<DevAddr>: Network address 4Byte

<Nwkskey>: Network session key 16Byte

<Appskey>: Application session key 16Byte

3.3.9 at+send

- **Command**

at+send=<type>,<port>,<data>\r\n

- **Description**

Send the packets string on a specified port number after join.

- **Parameter**

<type> = 0	send unconfirmed packets
= 1	send confirmed packets (Ack timeout Retry default 8)

<port> = 1-223 port number from 1 to 223

<data>= <hex value> hex value(no space). The Maximum length of <data> 64 bytes

- **Response**

OK\r\n----------Start to send packets

ERROR<code>\r\n----------Can't send packets

<code> = CODE_MAC_BUSY_ERR

= CODE_NOT_JOIN

= CODE_TX_ERR

- **Event Response**

at+recv=<status>,<port>[,<rssi>][,<snr>],<len>,<data>\r\n\r\n

3.3.10 at+recv

- **Command**

at+recv=<status >,<port>[,<rssi>][,<snr>],<len>[,<data>]\r\n

- **Description**

Receive the module event and data, auto notify to the host.

- **Parameter**

NULL

- **Response**

<status> = STATUS_RECV_DATA---------- received data from server or P2P client

STATUS_TX_COMFIRMED--------Transmission was successful and received ACK from server

STATUS_TX_UNCOMFIRMED----Transmission was successful

STATUS_TX_TIMEOUT----------Transmission was timeout

STATUS_RX2_TIMEOUT----------Transmission was unsuccessful, ACK not received back from the server

STATUS_P2PTX_COMPLETE------LoraP2P Continues Transmissions is completed

STATUS_UNKNOWN----------Status is unknown

<port> = 0-223----------If <status> = STATUS_RECV_DATA, it will be 1-223 and others will always be 0

<len> = 0-64----------If <status> = STATUS_RECV_DATA, it will be greater than 0.

<data> = <hex value>----------If <len>=0,this field will be empty.

<rssi> : signal strength，at+recv_ex enable can display

<snr> : signal and noise ratio，at+recv_ex enable can display

3.4 LoraP2P

3.4.1 at+rf_config

- **Command**

  at+rf_config[=<freq>, <sf>, <bw>, <cr>, <prlen>, <pwr>]\r\n

- **Description**

  Set LoRaP2P Configuration , it will used to the txc and rxc command. User can use this command to build their point to point communication or RFtest command. It will save to flash.

  Null Parameter is Get the current config.

- **Parameter**

  <freq>:---------- frequency, default 868100000 range: (860000000 ~9299000000)

  <sf>: ---------- spread factor, default 12 (6-12) more low more fast datarate

  <bw>: ---------- Band-with, default 0 (0:125KHz, 1:250KHz, 2:500KHz)

  <cr>: ---------- coding Rate, default 1 (1:4/5, 2:4/6, 3:4/7, 4:4/8)

  <prlen>: ----------Preamlen default 8 (8-65535)

  <pwr>: ---------- Tx power default 20 (5-20)

- **Response**

  OK[<freq>, <sf>, <bw>, <cr>, <prlen>, <pwr>]*\r\n*

  ERROR<code>\r\n

3.4.2 at+txc

- **Command**

  at+txc =<cnts>,<interval>,<datahex>\r\n

- **Description**

  Set LoRaP2P Tx continues, module will send the counts packet with rfconfig until receive the tx_stop command or reset, and insert the interval. This interval is begin with last packet send success or fail. This command also can used to RFtest , test the tx performance. If used

normal tx ,can set the cnts to 1.

- **Parameter**

<cnts>: ---------- tx counts (1-65535)

<interval>: ---------- tx interval between last send success or fail, (10-3600000ms)

<datahex>: ----------hex value (no space) , max 64

- **Response**

OK\r\n

ERROR<code>\r\n

- **Event Response**

at+recv=<status >,<port>,<len>,<data>\r\n

STATUS_P2PTX_COMPLETE ------ LoraP2P Continues Transmissions is completed

3.4.3 at+rxc

- **Command**

at+rxc=<report_en>\r\n

- **Description**

Set LoRaP2P Rx continues, module will receive packets with rfconfig until receive the rx_stop command or reset. This command also can used to RFtest , test the rx sensitivity, and you can set report_en :0 when RFtest. If used normal rx ,can set the report_en :1, report data to host.

- **Parameter**

<report_en>: ---------- enable report to host or not

- **Response**

OK\r\n

ERROR<code>\r\n

- **Event Response**

at+recv=<status >,<port>,<len>,<data>\r\n

STATUS_RECV_DATA---------- received data from server or P2P client

3.4.4 at+tx_stop

- **Command**

  at+tx_stop\r\n

- **Description**

  Set LoRaP2P Tx continues stop. Radio will switch to sleep mode .

- **Parameter**

  NULL

- **Response**

  OK\r\n

  ERROR<code>\r\n

3.4.5 at+rx_stop

- **Command**

  at+rx_stop\r\n

- **Description**

  Set LoRaP2P Rx continues stop.Radio will switch to sleep mode .

- **Parameter**

  NULL

- **Response**

  OK\r\n

  ERROR<code>\r\n

3.5 Radio

3.5.1 at+status

- **Command**

  at+status[=0]\r\n

- **Description**

  Get/Clean the radio statistics

- **Parameter**

  Null: Response TxSuccessCnt, TxErrCnt, RxSuccessCnt, RxTimeOutCnt, RxErrCnt, Rssi, Snr of last packet

  =0: Clear statistics.

- **Response**

  OK[<TxSuccessCnt>,<TxErrCnt>,<RxSuccessCnt>,<RxTimeOutCnt>,<RxErrCnt>,<Rssi>,<Snr>]*\r\n*

  ERROR<code>\r\n

3.6 Peripheral

3.6.1 at+uart

- **Command**

  at+uart[=<baud>,<data_bits>,<parity>,<stop_bits>,<flow_ctrl>]\r\n

- **Description**

  Set UART parameters and it will be available next reset time.

- **Parameter**

<baud>=<9600-921600>	Supports baud list
<data_bits>=<8>	8 data bits
<parity>=<0/1/2>	0=PARITY_NONE
	1=PARITY_EVEN
	2=PARITY_ODD
<stop_bits>=<1/2>	0=Stop bit length is 1 bit
	1=Stop bit length is 2 bit
<flow_ctrl>=<0/1>	0=disable flow control
	1=enable flow control

- **Response**

  OK[<baud>,<data_bits>,<parity>,<stop_bits>,<flow_ctrl>]\r\n ----------Save successfully

  ERROR<code>\r\n

  <code> = CODE_ARG_ERR---------Argument is invalid

- **Event Response**

  NULL

3.6.2 at+rd_reg

- **Command**

  at+rd_reg=<reg_addr>[,<len>]\r\n

■ **Description**

Read the module internal SX1276 register value.

■ **Parameter**

<reg_addr>: register address, reference SX1276 datasheet. Enter hex format.

<len>: Optional，default 1，Maximum 64

■ **Response**

OKxxxx\r\n ----------return value show in hex format
ERROR<code>\r\n
<code> = CODE_ARG_ERR---------Invalid argument

■ **Event response**

NULL

3.6.3 at+wr_reg

■ **Command**

at+wr_reg=<reg_addr>,<len>,<databuf>\r\n

■ **Description**

Modify the internal module SX1276 register value.

■ **Parameter**

<reg_addr>: Register address see SX1276 datasheet，Enter hex format.
<len>: Maximum 64
<databuf>: write data，Enter hex format， Maximun 64 hex value.

■ **Response**

OK\r\n ----------write success
ERROR<code>\r\n
<code> = CODE_ARG_ERR---------Invalid argument

■ **Event response**

NULL

3.6.4 at+gpio

■ **Command**

at+gpio=<pin>[,<level>]\r\n

■ **Description**

Read or set the assign PIN's high or low voltage level

■ **Parameter**

<pin>: Refer the appendix pin definition
<level>: 1 Output high 0 Output low

■ **Response**

OK1\r\n ---------- return voltage level
OK\r\n ---------- set success
ERROR<code>\r\n
<code> = CODE_ARG_ERR---------Invalid argument

■ **Event response**

NULL

3.6.5 at+rd_adc

■ **Command**

at+rd_adc=<pin>\r\n

■ **Description**

Read assign ADC PIN's voltage level，ADC precision is 12bit, Range 0-4096

■ **Parameter**

<pin>: Refer the appendix pin definition-ADC

■ **Response**

OK0\r\n ---------- return voltage level 0-4096 (0-3.3V)
ERROR<code>\r\n
<code> = CODE_ARG_ERR---------Invalid argument

■ **Event response**

NULL

3.6.6 at+rd_iic

■ **Command**

at+rd_iic=<dev_addr>,<data_addr>[,<len>]\r\n

■ **Description**

Read assign I2C PIN's connect device register value，I2C ferquency default 100KHz.

■ **Parameter**

<dev_addr>: I2C device MAC 7-bit mode，Enter hex format 8bit

<data_addr>: I2C device register address，Enter hex format 16bit

<len>: optional default 1 Max 64

■ **Response**

Return value show in hex format.

OK33\r\n ----------return register value 0x33

ERROR<code>\r\n

<code> = CODE_ARG_ERR---------Invalid argument

<code> = CODE_UNKNOWN_ERR---------device no response Timeout 10S

■ **Event response**

NULL

3.6.7 at+wr_iic

■ **Command**

at+wr_iic=<dev_addr>,<data_addr>,<databuf>\r\n

■ **Description**

Write assign I2C PIN's connect device register value. I2C frequency default 100KHz

■ **Parameter**

<dev_addr>: I2C device MAC 7-bit mode，enter hex format 8bit

<data_addr>: I2C device register address，enter hex format 16bit

<databuf>: enter hex format，Max 64

- **Response**

OK\r\n ----------modify success

ERROR<code>\r\n

<code> = CODE_ARG_ERR---------Invalid argument

<code> = CODE_UNKNOWN_ERR---------device no response Timeout 10S

- **Event response**

NULL

4 Command Examples

4.1 Join-Otaa

Welcome to RAK811

at+mode=0 /* SET LoraWAN work mode */
OK

at+band /* GET Region V2.0.2.0+ support */
OKEU868

at+get_config=dev_eui /* GET Dev_EUI check ,if need can set*/
OK3037343644357402

at+set_config=app_eui:39d7119f920f7952&app_key:a6b08140dae1d795ebfa5a6dee1f4dbd
/* SET LoraGateway app_eui and app_key , big endian*/
OK

at+join=otaa /* Join OTAA type*/
OK
at+recv=3,0,0 /* Join status success*/

4.2 Join-Abp

Welcome to RAK811

at+mode=0 /* SET LoraWAN work mode */
OK

at+band /* GET Region V2.0.2.0+ support */
OKEU868

at+set_config=dev_addr:00112233&nwks_key:3432567afde4525e7890cfea234a5821&apps_key:a48adfc393a0de458319236537a11d90 /* SET LoraGateway dev_addr nwks_key and apps_key , big endian*/
OK

at+join=abp /* Join ABP type*/
OK

4.3 LoraWAN send&recv

```
 /*After join gateway success, then can send and receive data*/
at+send=0,2,010203040506 /*APP port:2, unconfirmed message*/
at+recv=2,0,0    /*unconfirmed mean tx success*/

at+send=1,2,010203040506 /*APP port :2, confirmed message*/
at+recv=1,0,0    /*confirmed mean receive ack from gateway*/

 /*If gateway has data to send module, will receive date meanwhile ack */
at+recv=0,2,10,30313233343536373839    /*APP port :2, receive size 10, hex:
                                                    30313233343536373839*/
```

4.4 P2P send&recv

```
/* Module A Rx Side*/
Welcome to RAK811

at+mode=1                          /* SET LoraP2P work mode */
OK

at+rf_config=867700000,10,0,1,8,14   /* SET LoraP2P Frequency:867.7MHz, SF10,Bandwith
125KHz, coding Rate:4/5, Preamlen:8, tx power:14dbm */
OK

at+rxc=1                          /* SET LoraP2P Rx continue enable report rx data */
OK

at+rx_stop                        /* If want stop Rx continue */

/* Module B Tx Side*/
Welcome to RAK811

at+mode=1                          /* SET LoraP2P work mode */
OK

at+rf_config=867700000,10,0,1,8,14   /* SET LoraP2P Frequency:867.7MHz, SF10,Bandwith
125KHz, coding Rate:4/5, Preamlen:8, tx power:14dbm */
OK

at+txc=100,1000,800100000600010002da9557e142d9   /* SET LoraP2P Tx continue ,100 packets,
1S interval, hex data */
OK
```

```
at+recv=9,0,0      /*When Tx complete */
at+tx_stop                  /* If want stop Tx continue */
```

4.5 Peripheral operation

```
/* read gpio PIN2*/
 at+gpio=2
 OK0

/* write gpio PIN2 high level*/
 at+gpio=2,1
 OK

/* read adc at PIN2, 3.3V*/
at+rd_adc=2
OK4095

/* read i2c device lis2dh, slave addr 0x32, reg addr 0x0f (WHO_AM_I), len 1*/
at+rd_iic=32,0f,1
OK33   // return 0x33

/* write i2c device lis2dh, slave addr 0x32, reg addr 0x1f (TEMP_CFG_REG ), data C0*/
at+wr_iic=32,1f,c0
OK
```

5 Pin Define

Can configuration PIN list :

PIN	Name	PIN	Name
2	PB12/ADC	18	PB8/I2C_SCL
3	PB14/ADC	19	PB9/I2C_SDA
4	PB15/ADC	20	PA2/ADC
5	PA8	22	PA1/ADC
8	PA12	23	PA0/ADC
9	PB4	25	PB10
14	PA15	26	PB11
15	PB3	27	PB2
16	PB5		

6 Regions Define

Region	Max EIRP (dbm)	Tx power level	Default Tx Dr	ChanId	Freq	Tx Dr	Rx Dr	JoinCnts
EU868	16.0	0 - 7	0 - 5	0 - 15	TBD	0 - 7	0 - 7	>=48
US915	30.0	0 - 10	0 - 5	0 - 71	TBD	0 - 4	8 - 13	>=2
AU915	30.0	0 - 10	0 - 6	0 - 71	TBD	0 - 6	8 - 13	>=2
KR920	14.0	0 - 7	0 - 5	0 - 15	TBD	0 - 5	0 - 5	>=48
AS923	16.0	0 - 7	0 - 5	0 - 15	TBD	2 - 7	2 - 7	>=1
IN865	30.0	0 - 10	0 - 5	0 - 15	TBD	0 - 7	0 - 7	>=48

7 Version

Version	Major Changes	Date	Author
V1.0	Initial release	2016-06-08	lv
V1.1	Add LoraP2P mode	2016-11-15	junhua
V1.2	Modify some descriptions	2017-01-20	xc.cao
V1.3	Add command at+band, at+dr Modify at+set_config descriptions	2017-04-19	junhua
V1.4	Change LoraWAN 1.0.1 to 1.0.2 Add the gpio, i2c, adc peripheral operation command	2017-10-18	junhua
V1.5	Add region switching function	2018-10-17	chace