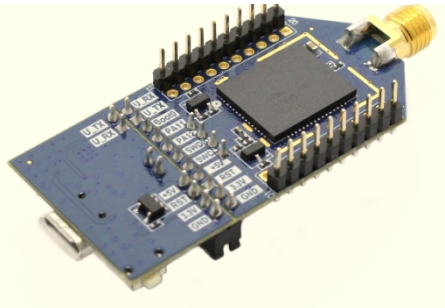
S76S/S78S Commands Set Reference



Document Name	**S76S/S78S Commands Set Reference**
Version	
Doc No	**K (FW ver. v1.6.0)**
Date	**Oct 11, 2017**

Date	Revised Contents	by	on

Nov 11 , 2016	Initial version supports EU868, US902 and US915 bands.	Leo Tseng	A
Nov 25, 2016	CN470-510 band added and Typo errors fixed. 6 new commands mac set_tx_mode, mac get_tx_mode, mac set_batt, mac get_batt, mac set_tx_confirm, mac get_tx_confirm,	Leo Tseng Leo Tseng	B C
Dec 22, 2016	2 new commands sip sleep, sip set_baudrate 5 CLAA commands mac set_claa, mac get_claa mac set_getchinfo, mac set_gettimeinfo mac set_jumboframe	Leo Tseng	D
Dec 23, 2016	8 new commands (v1.2.0) sip get_hw_model_ver sip set_gpio_mode sip set_gpio sip get_gpio rf fsk rf lora_tx_start rf lora_tx_stop rf lora_rx_start rf lora_rx_stop	Leo Tseng	E
Jan, 17, 2017	1 new commands (v1.2.1) sip get_uuid 7 new commands (v1.3.1) rf set_fdev rf get_fdev rf set_cad rf get_cad rf cad	Leo Tseng Leo Tseng	F G
Jan, 23, 2017	mac set_lbt mac get_lbt Change CN470 frequency table 6 new commands (v1.4.2) mac set_uplink_dwell mac get_uplink_dwell mac set_downlink_dwell mac get_downlink_dwell mac set_max_erip mac get_max_erip	Leo Tseng	H
Mar, 1, 2017	Support AS923 Band & LoRaWAN v1.0.2 5 new commands (v1.4.5) mac set_ch_count mac get_ch_count	Leo Tseng Leo Tseng	I J
Apr 24, 2017			

June, 9, 2017 Sep, 11, 2017 Oct, 11, 2017	sip set_storage sip get_storage mac set_keys 6 new commands (v1.5.5) sip set_iap sip set_iap_mode mac set_tx_interval mac get_tx_interval mac set_rx1_freq mac get_rx1_freq 17 new commands (v1.6.0) mac set_auto_join mac get_auto_join rm set_gpio rm set_port_uplink rm set_port_downlink rm set_gpio_switch rm set_adc rm set_adc_switch rm set_mode rm set_trigger rm get_gpio rm get_port rm get_gpio_switch rm get_adc rm get_adc_switch rm get_mode rm get_trigger	Leo Tseng	K

Index

1. Introduction

2. Configuration
- 2.1 Hardware Configuration
- 2.2 Software Configuration

3. Commands Set Reference
- 3.1 SIP commands
  - 3.1.1 sip factory_reset
  - 3.1.2 sip get_ver
  - 3.1.3 sip get_hw_deveui
  - 3.1.4 sip reset
  - 3.1.5 sip get_hw_model
  - 3.1.6 sip set_echo
  - 3.1.7 sip set_log
  - 3.1.8 sip sleep
  - 3.1.9 sip_baudrate
  - 3.1.10 sip_get_hw_model_ver
  - 3.1.11 sip_set_gpio_mode
  - 3.1.12 sip_set_gpio
  - 3.1.13 sip_get_gpio
  - 3.1.14 sip_get_uuid
  - 3.1.15 sip set_storage
  - 3.1.16 sip get_storage
  - 3.1.17 sip set_iap
  - 3.1.18 sip set_iap_mode
- 3.2 MAC commands
  - ~~3.2.1 mac set_band~~ (since v1.4.3 removed)
  - 3.2.2 mac tx
  - 3.2.3 mac join
  - 3.2.4 mac save
  - 3.2.5 mac get_join_status
  - 3.2.6 mac set_linkchk
  - 3.2.7 mac set_deveui
  - 3.2.8 mac set_appeui
  - 3.2.9 mac set_appkey
  - 3.2.10 mac set_devaddr
  - 3.2.11 mac set_nwkskey
  - 3.2.12 mac set_appskey
  - 3.2.13 mac set_power
  - 3.2.14 mac set_dr
  - 3.2.15 mac set_adr
  - 3.2.16 mac set_txretry
  - 3.2.17 mac set_rxdelay1
  - 3.2.18 mac set_rx2
  - 3.2.19 mac set_sync

3.2.20 mac set_ch_freq
3.2.21 mac set_ch_dr_range
3.2.22 mac set_ch_status
3.2.23 mac set_dc_ctl
3.2.24 mac set_dc_band
3.2.25 mac set_join_ch
3.2.26 mac set_upcnt
3.2.27 mac set_downcnt
3.2.28 mac set_class
3.2.29 mac get_devaddr
3.2.30 mac get_deveui
3.2.31 mac get_appeui
3.2.32 mac get_nwkskey
3.2.33 mac get_appskey
3.2.34 mac get_appkey
3.2.35 mac get_dr
3.2.36 mac get_band
3.2.37 mac get_power
3.2.38 mac get_adr
3.2.39 mac get_txretry
3.2.40 mac get_rxdelay
3.2.41 mac get_rx2
3.2.42 mac get_sync
3.2.43 mac get_ch_para
3.2.44 mac get_ch_status
3.2.45 mac get_dc_ctl
3.2.46 mac get_dc_band
3.2.47 mac get_join_ch
3.2.48 mac get_upcnt
3.2.49 mac get_downcnt
3.2.50 mac get_class
3.2.51 mac set_tx_mode
3.2.52 mac get_tx_mode
3.2.53 mac set_batt
3.2.54 mac get_batt
3.2.55 mac set_tx_confirm
3.2.56 mac get_tx_confirm
3.2.57 mac set_claa
3.2.58 mac get_claa
3.2.59 mac set_getchinfo
3.2.60 mac set_gettimeinfo
3.2.61 mac set_jumboframe
3.2.62 mac set_lbt
3.2.63 mac get_lbt
3.2.64 mac set_uplink_dwell
3.2.65 mac get_uplink_dwell
3.2.66 mac set_downlink_dwell
3.2.67 mac get_downlink_dwell
3.2.68 mac set_max_erip

3.2.69 mac get_max_erip
3.2.70 mac set_ch_count
3.2.71 mac get_ch_count
3.2.72 mac set_keys
3.2.73 mac set_tx_interval
3.2.74 mac get_tx_interval
3.2.75 mac set_rx1_freq
3.2.76 mac get_rx1_freq
3.2.77 mac set_auto_join
3.2.78 mac get_auto_join
3.3 RF commands
3.3.1 rf rx
3.3.2 rf tx
3.3.3 rf set_freq
3.3.4 rf set_pwr
3.3.5 rf set_sf
3.3.6 rf set_bw
3.3.7 rf set_cr
3.3.8 rf set_prlen
3.3.9 rf set_crc
3.3.10 rf set_iqi
3.3.11 rf set_sync
3.3.12 rf save
3.3.13 rf get_freq
3.3.14 rf get_pwr
3.3.15 rf get_sf
3.3.16 rf get_bw
3.3.17 rf get_prlen
3.3.18 rf get_crc
3.3.19 rf get_iqi
3.3.20 rf get_cr
3.3.21 rf get_sync
3.3.22 rf rx_con
3.3.23 rf fsk
3.3.24 rf lora_tx_start
3.3.25 rf lora_tx_stop
3.3.26 rf lora_rx_start
3.3.27 rf lora_rx_stop
3.3.28 rf set_fdev
3.3.29 rf get_fdev
3.3.30 rf set_cad
3.3.31 rf get_cad
3.3.32 rf cad
3.4 RM commands
3.4.1 rm set_gpio
3.4.2 rm get_gpio
3.4.3 rm set_gpio_switch
3.4.4 rm get_gpio_switch
3.4.5 rm set_adc

3.4.6 rm get_adc
3.4.7 rm set_adc_switch
3.4.8 rm get_adc_switch
3.4.9 rm set_port_uplink
3.4.10 rm set_port_downlink
3.4.11 rm get_port
3.4.12 rm set_mode
3.4.13 rm get_mode
3.4.14 rm set_trigger
3.4.15 rm get_trigger
4. Example
4.1 LoRaWAN
4.1.1 ABP
4.1.2 OTAA
4.1.3 Confirmed Uplink and Downlink
4.2 Node to Node
4.3 Remote Mode
4.3.1 Report GPIO, ADC Data & Uplink to Server
4.3.2 Downlink from Server & Control GPIO Pins

1. Introduction

The S76S is designed & manufactured in a smallest form factor - SiP (System in Package). It integrates with Semtech SX1276 and a 32-bit ultra-low power Cortex M0+ MCU (STM32L073x), supporting global 868 MHz or 915 MHz ISM-Bands. (S78S supports 433MHz or 470 MHz by using SX1278 and the identical MCU with S76S) Capable of 2-way communication and reach over 16 km (10 miles) distance in our field test.

This product is designed with multiple easy to use interfaces (I2C/SPI/UART/GPIO), fine-tuned RF performance and will be offered with complete SDK library & ready to go HDK, it can significantly help the users to shrink the size of end device and simplify the development efforts for any LoRa applications.

For faster development, AcSiP provides an EKB named EK-S76SXB. The EKB equips with a UART-To-USB bridge IC and can be powered by USB. Besides, a SMA antenna connector is also provided for easy antenna installation. Figure1.1 describes the related components above.

S76S module provides a commands set interface that can use LoRa™ and LoRaWAN™ communication through UART interface. And S76S LoRaWAN™ protocol has been certificated by LoRaWAN alliance.

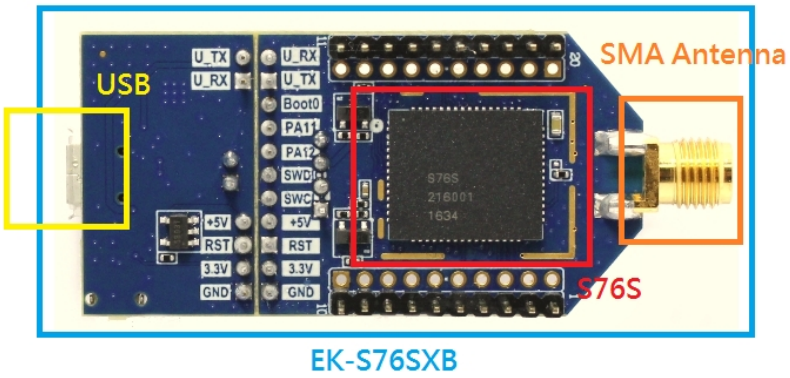


Figure 1.1 (Take S76S as example)

2. Configuration

2.1 Hardware Configuration

S76S/S78S EKB can be controlled by connecting TX(PA9) and RX(PA10) UART1 interface to other MCU, as shown in Figure 2.1, or by connecting micro USB port directly to PC/NB as shown in Figure 2.2, The control commands can be sent from PC or other MCU to S76S/S78S.

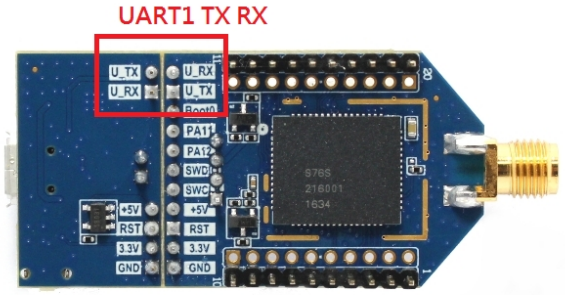


Figure 2.1

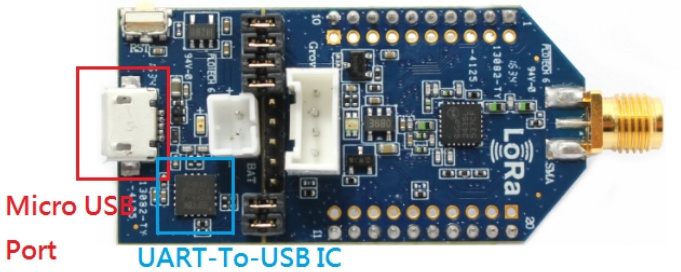


Figure2.2

2.2 Software Configuration

The default baud rate of S76S/S78S UART1 is set at 115200. And the rest of UART1 setting, please follow these below settings:

Baud rate: **115200**
Data bits: **8**
Stop bits: **1**
Parity: **none**
Flow Control: **none**
Forward: **none**

To quickly start using S76S/S78S EVB, the 1st step is using USB cable to connect EVB to PC/NB via micro USB port. The next step is checking whether the UART-To-USB bridge IC driver can be properly installed on PC/NB. By using win7/win10, the UART-To-USB bridge IC driver could be installed automatically and shows a USB serial com port after connecting well between EVB and PC/NB via USB cable.

After successful installation of USB driver, you can use any terminal program (suggesting free terminal software: termite) to connect to EVB. The commands set can be used through the terminal program.

By using termite or other terminal software, be aware of not being appended nothing in the end of a UART string (Figure 2.3).

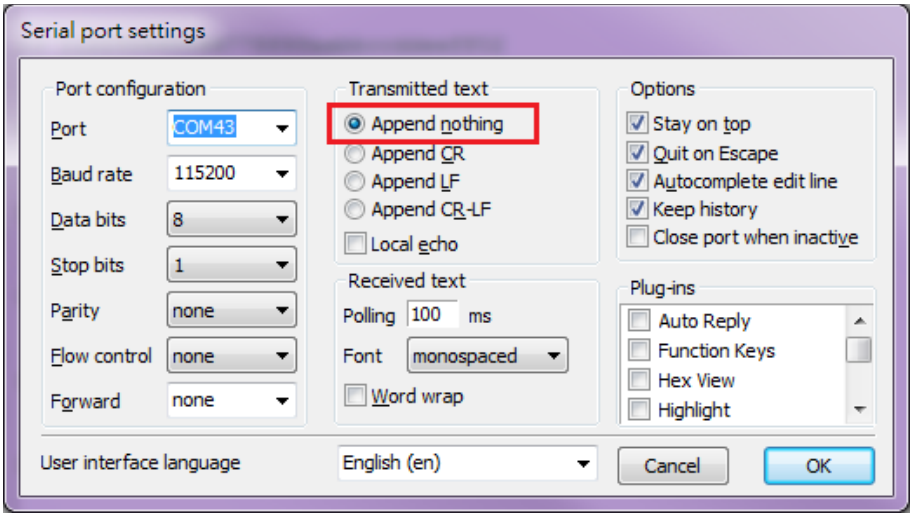

Serial port settings
Port configuration
Port
COM43
Baud rate
115200
Data bits
8
Stop bits
1
Parity
none
Flow control
none
Forward
none
Transmitted text
Append nothing
Append CR
Append LF
Append CR-LF
Local echo
Received text
Polling
100
ms
Font
monospaced
Word wrap
Options
Stay on top
Quit on Escape
Autocomplete edit line
Keep history
Close port when inactive
Plug-ins
Auto Reply
Function Keys
Hex View
Highlight
User interface language
English (en)
Cancel
OK

Figure 2.3

3. Commands Set Reference

S76S/S78S commands set can be categorized into 4 types: SIP command, MAC (LoRaWAN™) command, RF command and RM command. SIP commands are controlling commands that are relevant or direct MCU control and not related to radio transmission. MAC commands are used to utilize LoRaWAN™ protocol to communicate with Servers or modify LoRaWAN™ related parameters. RF commands can be used to send or receive LoRa raw packet with going through LoRaWAN™ protocol. RM (remote mode) commands can report S76S/S78S GPIO, ADC data uplink to server, or control GPIO states downlink from server.

The command set interface is readable ASCII string. S76S/S78S starts to accept command if the string starts from “sip”, “mac”, “rf” and “rm”, and the response string from S76S/S78S starts with two ‘>’ characters and one ‘space’. For example, the first line of the below demo is a module command, and the second line is the response from S76S/S78S.

sip get_ver
>> v1.0.8

3.1 SIP commands

3.1.1sip factory_reset

Response: A string representing firmware version.

Purpose: All LoRaWAN and radio configuration parameters will be set to default value.

Example:
sip factory_reset
>> v1.0.8

3.1.2sip get_ver

Response: A string representing firmware version.
Purpose: Get current firmware version.
Example:
sip get_ver
>> v1.0.8

3.1.3sip get_hw_deveui

Response: A string representing hardware EUI in hexadecimal. Get hardware EUI.
Purpose: Get a deveui calculated from MCU UUID registers, this value can’t be change by “mac set_deveui” command.
Example:
sip get_hw_deveui
>> 9c65f9fffe123456

3.1.4sip reset

Response: The beginning information since FW starts.
Purpose: This command resets the module and start FW over again.
Example:
sip reset
>> S76S - v1.0.8 - Nov 10 2016 - 17:05:43

3.1.5sip get_hw_model

Response: A string representing hardware model.
Purpose: Get hardware model name.
Example:
sip get_hw_model
>> S76S

3.1.6sip set_echo <Status>

<Status>: A string representing echo status, it can be **on** or **off**.
Response: **Ok**, if <Status> string is valid.
Invalid, if <Status> string is not valid.
Purpose: Enable or disable UART echo mode.
Example:
sip set_echo on
>> Ok

3.1.7sip set_log <Log_Level>

<Log_Level> **debug**: show all logs, **info**: only shows commands set when input/output.
Response: **Ok**, if <Status> string is valid.
Invalid, if <Status> string is not valid
Purpose: Select which level when logs show
Example:
sip set_log info
>> Ok

3.1.8sip sleep <Time> <Interruptible>

<Time> A decimal string representing S76S/S78S sleep time in seconds, it can be assigned from **2** to **65536**.
<Interruptible> During the period of sleep mode, it can be decided to be interrupted by UART or not. **uart_on** means it can be interrupted (waked up) by UART; **uart_off** means it can't be interrupted by UART.
Response: **Ok**, if <Status> string is valid.
Invalid, if <Status> string is not valid
Purpose: Let S76/78S enter sleep mode by an assigned sleep time
Example:
sip sleep 100 uart_on
>> Ok

3.1.9sip set_baudrate <baudrate> <password>

<baudrate> A decimal string representing UART1 baudrate, it can be assigned as 4 kinds of rate, **9600**, **19200**, **57600** and **115200**.
<password> A decimal string representing password that provides baudrate protection. This baudrate setting command can only be delivered when password is correct. And the assigned baudrate setting would be stored in EEPROM and would not be changed by sip factory_reset command.
Response: **Ok**, if <Status> string is valid.
Invalid, if <Status> string is not valid
Purpose: Change UART1 baudrate immediately.
Example:
sip set_baudrate 9600 12345678
>> Ok

3.1.10 (production verification) sip get_hw_model_ver

Response: A string representing hardware model & firmware version.

Purpose: Get hardware model name and firmware version by only using this command.

Example:

sip get_hw_model_ver

>> module=S76S ver=v1.0.8

3.1.11 (production verification) sip set_gpio_mode <Gpio_Group> <Gpio_Pin_Number> <Gpio_Mode>

<Gpio_Group> A string representing STM32 GPIO pin groups, it can be these characters **A, B, C, D, E, F** and **H** (note: no G).

<Gpio_Pin_Number> A decimal string representing STM32 GPIO pin number, it can be set from **1** to **15**.

<Gpio_Mode > A decimal string representing STM32 GPIO pin mode, it can be assigned as output or input, set **1** would let pin state be output mode and **0** let it as input mode.

Response: **Ok**, if input arguments are valid.
Invalid, if input argument are not valid or out of range.

Purpose: Assign STM32 GPIO pin mode as input or output.

Example (Set PA_0 as output mode):

sip set_gpio_mode A 0 1

>> Ok

3.1.12 (production verification) sip set_gpio <Gpio_Group> <Gpio_Pin_Number> <Gpio_Value>

<Gpio_Group> A string representing STM32 GPIO pin groups, it can be these characters **A, B, C, D, E, F** and **H** (note: no G).

<Gpio_Pin_Number> A decimal string representing STM32 GPIO pin number, it can be set from **0** to **15**.

<Gpio_Value > A decimal string representing STM32 GPIO pin value, it can be assigned as high or low, set **1** would let pin be high state and **0** let it as low state.

Response: **Ok**, if input arguments are valid.
Invalid, if input argument are not valid or out of range.

Purpose: Assign STM32 GPIO pin state as high or low.

Example (Set PA_0 as output high state):

sip set_gpio_mode A 0 1

>> Ok

sip set_gpio A 0 1

>> Ok

3.1.13 (production verification) sip get_gpio <Gpio_Group> <Gpio_Pin_Number>

<Gpio_Group> A string representing STM32 GPIO pin groups, it can be these

characters **A, B, C, D, E, F** and **H** (note: no G).
<Gpio_Pin_Number> A decimal string representing STM32 GPIO pin number, it can be set from **0** to **15**.

Response: **1**, if this GPIO pin state is high.
0, if this GPIO pint state is low.
Ok, if input arguments are valid.
Invalid, if input argument are not valid or out of range.

Purpose: Get the pin state from the assigned GPIO pin.

Example (get PA_2 pin state):
sip get_gpio A 2
>> 1

3.1.14 (production verification) sip get_uuid

Response: A string representing hardware STM32L0 MCU UUID 96-bit value.
Purpose: Each STM32 MCU device has its own unique UUID, use this command to read it out

Example:
sip get_uuid
>> uuid=002400413630373619473630

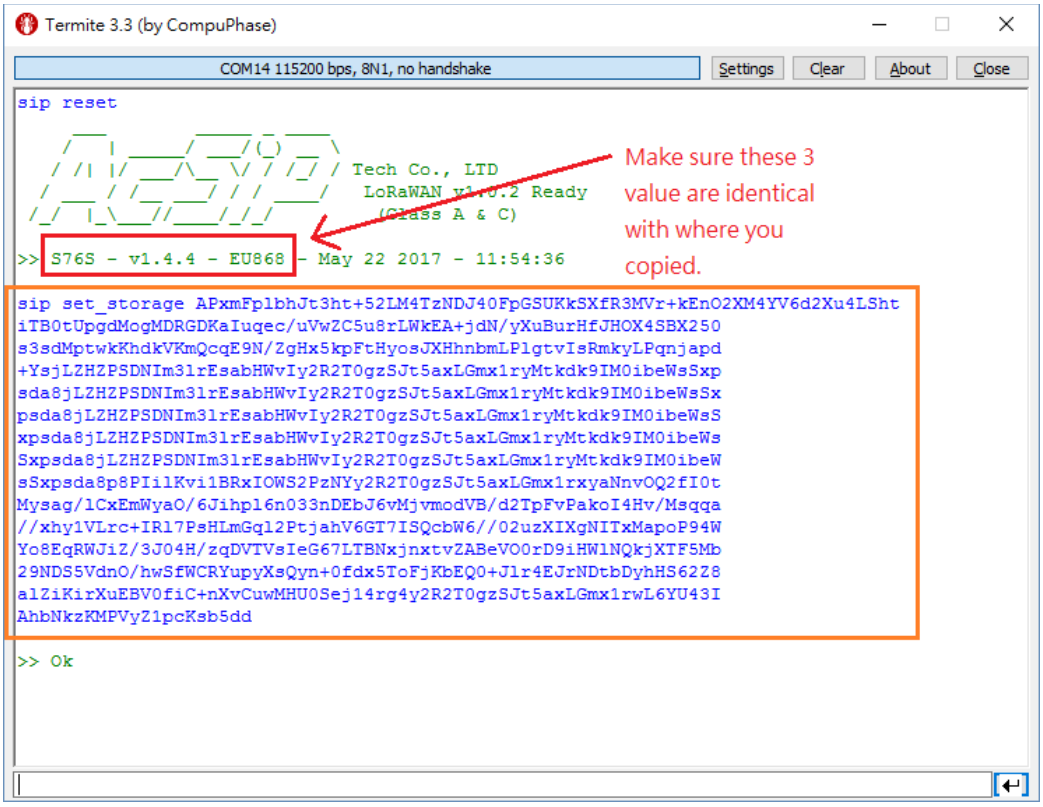
3.1.15 sip set_storage <EEPROM_Encrypted>

<EEPROM_Encrypted>: a series of ASCII string representing the stored EEPROM encrypted data.

Purpose: To overwrite whole EEPROM data in just one-time, it allows to set its own EERPOM from another S76S/S78S EEPROM data (*they must use the same HW model and FW version*).

Response: **Ok**, if <EEPROM_Encrypted> string is valid
Data format error, if <EEPROM_Encrypted> format or length not matched
Checksum format error, wrong checksum format or length of <EEPROM_Encrypted> not match.
Not enough memory space, no enough internal RAM to execute this command, please execute “sip reset” and try again.
Decrypted length not same as encrypted one, <EEPROM_Encrypted> length not match with what it comes from “sip get_storage”, it could be using a different FW version between set & get commands.
AES decrption error, AES execution error occurs.
Invalid, if anything is wrong in executing this command.

Example:



3.1.16 sip get_storage

Purpose: To overwrite whole EEPROM data in just one-time, after caller gets EEPROM encrypted data by using this commands, it allows to overwrite other device's EERPOM (*they must use the same HW model and FW version*).

Response: a series of ASCII string representing the stored EEPROM encrypted data. Copy these ASCII characters and paste into "sip set_storage" as parameters.

Example:

3.1.17 (Only IAP version supports) sip set_iap <Switch>

<Switch>: A string representing whether S76S will enable IAP boot-up feature, it could be used to enable IAP bootloader when the next boot-up, the string can be **on** or **off.**

Response: **Ok**, if <Switch> string is valid.
Invalid, if <Switch> string is not valid.

Purpose: To enable IAP bootloader when the next boot-up by using "sip reset" or power off/on if IAP bootloader is exist in the current S76S flash. Besides, this command can only be used in the version which is after v1.5.0.

Example:
sip set_iap on

>> Ok

3.1.18 (Only IAP version supports) sip set_iap_mode <Mode>

<Mode>: A string representing the IAP mode is set to either "Normal" or "Silent" mode, the string can be **normal** or **silent.**

Response: **Ok**, if <Mode> string is valid.
Invalid, if <Mode> string is not valid.

Purpose: When IAP is on, the next boot-up would enter IAP bootloader directly, but what if user want to set IAP show normal or silent message in IAP bootloader before it enters IAP bootloader, user can use this command to assign IAP mode (either normal or silent) even when running in LoRaWAN firmware. When IAP mode is assigned, the next boot-up of IAP bootloader will running at the selected mode.

Example:
sip set_iap_mode normal

>> Ok
sip set_iap_mode silent

>> Ok

3.2 MAC commands

~~3.2.1mac set_band <FreqBand>~~ (since v1.4.3 removed)

3.2.2mac tx <Type> <PortNum> <Data>

<Type>: a string representing type of transmitting message, can be **cnf** (confirmed) or **ucnf**
(unconfirmed).
<PortNum>: a decimal string representing port number used for transmission, it can be from **1** to **223**.
<Data>: a hexadecimal string representing data to be transmitted.
(e.g. *98ba34fd means "0x98, 0xba, 0x34, 0xfd 4bytes would be sent out")*

Response: there are two responses after entering this command. The first response will be received after entering command. The second response will be received after transmission.

First response: **Ok**, if <Type>, <PortNum> and <Data> strings are valid.
Invalid, if <Type>, <PortNum> and <Data> strings are not valid.
not_joined, module is not joined LoRaWAN.
no_free_ch, no channels are available.
busy, internal state is busy.
invalid_data_length, data length is larger than the allowed data length by LoRaWAN.
exceeded_data_length, data length is larger than 250 bytes.

Second response: **tx_ok**, successfully transmit data.
mac rx <portnum> <data>, there is downlink data.
<portnum> - a decimal string representing receiving port
<data> - a hexadecimal string representing received data.
err, acknowledgement is not received, if confirmed message is used.

Purpose: Star transmission by following LoRaWAN™ uplink format.

Example:
mac tx ucnf 15 98ba34fd
>> Ok
>> tx_ok
mac tx ucnf 15 6805
>> Ok
>> mac rx 4 1234abcd
(Got Downlink Data 0x12, 0x34, 0xab and 0xcd from Port 4)

If the device class is set to class C, a downlink data would be received at any time. The downlink data of class C is outputted by S76S/S78S in rx <Portnum> <Data> format.
Example:
>> mac rx 4 1234abcd

3.2.3mac join <Mode>

<Mode>: a string representing join mode of LoRaWAN, can be **otaa** (over-the-air activation) or **abp**

(activation by personalization).

Response: there is two responses after entering this command. The first response, used to indicate that whether command is valid or parameters is set appropriately, will be received after entering command. The second response will be received after join procedure.

First response: **Ok**, if <Mode> string is valid.
Invalid, if <Mode> string is not valid.
keys_not_init, keys are not configured.
no_free_ch, no channels are available.
busy, internal state is busy.

Second response: **accepted**, successfully join LoRaWAN.
unsuccess, join procedure is unsuccessful.

Purpose: Start join procedure of LoRaWAN.
Example:
mac join abp
>> Ok
>> accepted

Note: With ABP join, there is no over-the-air communication during the join process (see alternative method OTAA). The devaddr and keys are just being set up in the mac layer of the end node ready for use. For this reason the 'accepted" response doesn't actually prove that the end-node is communicating with a network, it just means the parameters have been set up in the mac layer correctly.

3.2.4mac save

Response: **Ok**
Purpose: Save LoRaWAN configuration parameters to flash.
Example:
mac save
>> Ok

3.2.5mac get_join_status

Response: a string representing whether module is joined successfully. Returned string can be: **joined**, **unjoined**.
Purpose: Get join status of LoRaWAN.
Example:
mac get_join_status
>> joined

3.2.6mac set_linkchk

Response: **Ok**.
Purpose: Next packet sent to server will include a Link Check MAC command. The downlink of sent packet will contain the response of Link Check MAC command. The response includes:

DemoMargin: link margin in dB of the last successfully received Link Check MAC command, value from 0 to 255
NbGateways: gateway number that successfully received the last Link Check MAC command, value from 0 to 255

Example:
mac set_linkchk
>> Ok
mac tx ucnf 11 55
>> Ok
>> DemodMargin = 19
>> NbGateways = 1
>> tx_ok

3.2.7mac set_deveui <DevEUI>

<DevEUI>: an 8-byte hexadecimal string representing Device EUI used for LoRaWAN.

Response: **Ok**, if <DevEUI> string is valid
Invalid, if <DevEUI> string is not valid.
Purpose: Set Device EUI used for LoRaWAN.
Example:
mac get_deveui 9c65f9fffe123456
>> Ok

Note: This assigned DevEUI would be stored into EEPROM immediately after it is changed. No need to use "mac save" command to store it.

3.2.8mac set_appeui <AppEUI>

<AppEUI>: an 8-byte hexadecimal string representing Application EUI used for LoRaWAN.
Response: **Ok**, if <AppEUI> string is valid.
Invalid, if <AppEUI> string is not valid.
Purpose: Set Application EUI used for LoRaWAN.
Example:
mac set_appeui 0000000000000000
>> Ok

3.2.9mac set_appkey <AppKey>

<AppKey>: a 16-byte hexadecimal string representing Application Key used for LoRaWAN.
Response: **Ok**, if <AppKey> string is valid
Invalid, if <AppKey> string is not valid.
Purpose: Set Network Session Key used for LoRaWAN.
Example:
mac set_appkey 2b7e151628aed2a6abf7158809cf4f3c
>> Ok

3.2.10 mac set_devaddr <DevAddr>

<DevAddr>: a 4-byte hexadecimal string representing Device Address used for LoRaWAN.
Response: **Ok**, if <DevAddr> string is valid
Invalid, if <DevAddr> string is not valid.
Purpose: Set Device Address used for LoRaWAN.
Example:
mac set_devaddr 12345678
>> Ok

3.2.11 mac set_nwkskey <NwkSessionKey>

<NwkSessionKey>: a 16-byte hexadecimal string representing Network Session Key used for LoRaWAN.

Response: **Ok**, if <NwkSessionKey> string is valid
Invalid, if <NwkSessionKey> string is not valid.

Purpose: Set Network Session Key used for LoRaWAN. Example:

mac set_nwkskey 2b7e151628aed2a6abf7158809cf4f3c
>> Ok

3.2.12 mac set_appskey <AppSessionKey>

<AppSessionKey>: a 16-byte hexadecimal string representing Application Session Key used for LoRaWAN.

Response: **Ok**, if <AppSessionKey> string is valid
Invalid, if <AppSessionKey> string is not valid.

Purpose: Set Application Session Key used for LoRaWAN.

Example:

mac set_appskey 2b7e151628aed2a6abf7158809cf4f3c
>> Ok

3.2.13 mac set_power <Power>

<Power>: a decimal string representing transmitting power in dBm, can be **2, 5, 8, 11, 14, 20** (non-915 band); **30, 28, 26, 24, 22, 20, 18, 16, 14, 12, 10** (915 band); **17, 16, 14, 12, 10, 7, 5, 2** (470 band)

Response: **Ok**, if <Power> string is valid
Invalid, if <Power> string is not valid.

Purpose: Set transmitting power.

Example:

mac set_power 14
>> Ok

3.2.14 mac set_dr <Datarate>

<DataRate>: a decimal string representing data rate used for LoRaWAN, it can be from **0** to **6.** (US902 is limited from **0** to **4**; CN470 is limited from **0** to **5)**

Response: **Ok**, if <DataRate> string is valid
Invalid, if <DataRate> string is not valid.

Purpose: Set uplink's data rate used for LoRaWAN.

Example:

mac set_dr 0
>> Ok

3.2.15 mac set_adr <State>

<State>: a string representing whether ADR is **on** or **off**.

Response: **Ok**, if <State> string is valid
Invalid, if <State> string is not valid.

Purpose: Set the state of ADR.

Example:

mac set_adr on
>> Ok

3.2.16 mac set_txretry <RetryCount>

<RetryCount>: a decimal string representing retry number of transmission, it can be from **0** to **255**. Response: **Ok**, if <RetryCount> string is valid
Invalid, if <RetryCount> string is not valid.
Purpose: Set retry number of transmission.
Example:
mac set_txretry 8
>> Ok

3.2.17 mac set_rxdelay1 <Delay>

<Delay>: a decimal string representing delay interval in milliseconds used for receive window 1, it can be from **0** to **65535**. Delay interval of receive window 2 will be set to **<Delay>+1.**
Response: **Ok**, if <Delay> string is valid
Invalid, if <Delay> string is not valid.
Purpose: Set delay interval of receive window 1.
Example:
mac set_rxdelay1 1000
>> Ok

3.2.18 mac set_rx2 <DataRate> <Frequency>

<DataRate>: a decimal string representing data rate of second receive window, it can be **0** to **7** (868 band); **0** to **5** (470 band); **0** to **15** (902-924 band).
<Frequency>: a decimal string representing operation frequency of second receive window in Hz, can be from **862000000** to **932000000**.

Response: **Ok**, if <DataRate> and <Frequency> strings are valid
Invalid, if <DataRate> and <Frequency> strings are not valid.
Purpose: Set data rate and operation frequency used for second receive window.
Example:
mac set_rx2 0 868000000
>> Ok

3.2.19 mac set_sync <SyncWord>

<SyncWord>: a hexadecimal string representing sync word, it can be from **0** to **FF**.
Response: **Ok**, if <SyncWord> string is valid
Invalid, if <SyncWord> string is not valid.
Purpose: Set the sync word used for communication.
Example:
> mac set_sync 34
>> Ok

3.2.20 mac set_ch_freq <ChannelId> <Frequency>

<ChannelId>: a decimal string representing channel number, its value range depends on different regional band (e.g. EU868 range falls in **0** to **15**; US902 range falls in **0** to **71**).
<Frequency>: a decimal string representing operation frequency of specified channel in Hz, it can be from **902000000** to **932000000** (902-924 band); from **470000000** to **510000000** (470 band); from **433000000** to **932000000** (other bands).
Response: **Ok**, if <ChannelId> and <Frequency> strings are valid.
Invalid, if <ChannelId> and <Frequency> strings are not valid.
Purpose: Set operation frequency of specified channel.

Example:
mac set_ch_freq 0 868000000
>> Ok

3.2.21 mac set_ch_dr_range <ChannelId> <MinDR> <MaxDR>

<ChannelId>: a decimal string representing channel number, its value range depends on different regional band (e.g. EU868 range falls in **0** to **15**; US902 range falls in **0** to **71**).
<MinDR>: a string representing minimum data rate, can be from **0** to **6**.
<MaxDR>: a string representing maximum data rate, can be from **0** to **6**.

Response: **Ok**, if <ChannelId>, <MinDR> and <MaxDR> strings are valid.
Invalid, if <ChannelId>, <MinDR> and <MaxDR> strings are not valid.
Purpose: Set data rate range of specified channel.
Example:
mac set_ch_dr_range 0 0 6
>> Ok

3.2.22 mac set_ch_status <ChannelId> <Status>

<ChannelId>: a decimal string representing channel number, its value range depends on different regional band (e.g. EU868 range falls in **0** to **15**; US902 range falls in **0** to **71**).
<Status>: a string representing whether the specified channel is **on** or **off**.

Response: **Ok**, if <ChannelId> and <Status> strings are valid.
Invalid, if <ChannelId> and <Status> strings are not valid.
Purpose: Enable of disable specified channel. Example:
mac set_ch_status 0 on
>> Ok

3.2.23 mac set_dc_ctl <Status>

<Status>: a string representing duty cycle status, it can be **on** or **off**.
Response: **Ok**, if <Status> string is valid.
Invalid, if <Status> string is not valid. Enable or disable duty cycle check at transmitting packet.
Example:
mac set_dc_ctl on
>> Ok

3.2.24 mac set_dc_band <BandID> <DutyCycle>

<BandID>: a decimal string representing band number, it can be from **0** to **15**.
<DutyCycle>: a decimal string representing duty cycle of specified band, can be from **0** to **65535**.
0: means 0%.
1-65535: duty cycle is equal to 1/<duty cycle>.
Response: **Ok**, if <BandID> and <DutyCycle> strings are valid.
Invalid, if <BandID> and <DutyCycle> strings are not valid.
Purpose: Set frequency range and duty cycle of specified band.
Example:
mac set_dc_band 1 100
>> Ok

3.2.25 mac set_join_ch <ChannelId> <Status>

<ChannelID>: a decimal string representing channel number, it can be from **0** to **15**.
<Status>: a string representing whether the specified join channel is **on** or **off**.

Response: **Ok**, if <ChannelID> and <Status> string is valid.
Invalid, if <ChannelID> and <Status> string is not valid.
Purpose: Set frequency channel for join request.
Example:
mac set_join_ch 1 on
>> Ok

3.2.26 mac set_upcnt <UplinkCounter>

<UplinkCounter>: a decimal string representing uplink counter, it can be from **0** to **4294967295**.
Response: **Ok**, if <UplinkCounter> string is valid.
Invalid, if <UplinkCounter> string is not valid. Set uplink counter that will be used for next uplink transmission.
Note: Not suggested to change Uplink counter when executing LoRaWAN™ protocol.
Example:
mac set_upcnt 1
>> Ok

3.2.27 mac set_downcnt <DownlinkCounter>

<DownlinkCounter>: a decimal string representing downlink counter, it can be from **0** to **4294967295**.
Response: **Ok**, if <DownlinkCounter> string is valid.
Invalid, if <DownlinkCounter> string is not valid. Set downlink counter that will be used for next downlink reception.
Example:
mac set_downcnt 1
>> Ok

3.2.28 mac set_class <Class>

<Class>: A or C.
Response: **Ok**, if <Class> is valid.
Invalid, if <Class> is not valid.
already_joined, if this command executes after joined either by OTAA or ABP.
Purpose: Set class type of LoRaWAN™.
Behavior: 1. RX2 window would not open immediately after “mac set_class” executed. 2. It only opens RX2 windows after joined. 3. Not allow to execute “mac set_class” after joined.
Example:
mac set_class C
>> Ok

3.2.29 mac get_devaddr

Response: a hexadecimal string representing Device Address used for LoRaWAN™.
Purpose: Return Device Address used for LoRaWAN™.
Example:
mac get_devaddr

>> 12345678

3.2.30 mac get_deveui

Response: a hexadecimal string representing Device EUI used for LoRaWAN™.
Purpose: Return Device EUI used for LoRaWAN™.
Example:
mac get_deveui
>> 000b78ffff000000

3.2.31 mac get_appeui

Response: a hexadecimal string representing Application EUI used for LoRaWAN™.
Purpose: Return Application EUI used for LoRaWAN™.
Example:
mac get_appeui
>> 0000000000000000

3.2.32 mac get_nwkskey

Response: a hexadecimal string representing Network Session Key used for LoRaWAN™.
Purpose: Return Network Session Key used for LoRaWAN™.
Example:
mac get_nwkskey
>> 2b7e151628aed2a6abf7158809cf4f3

3.2.33 mac get_appskey

Response: a hexadecimal string representing Application Session Key used for LoRaWAN™.
Purpose: Return Application Session Key used for LoRaWAN™.
Example:
mac get_appskey
>> 2b7e151628aed2a6abf7158809cf4f3c

3.2.34 mac get_appkey

Response: a hexadecimal string representing Application Key used for LoRaWAN™.
Purpose: Return Application Key used for LoRaWAN™.
Example:
mac get_appkey
>> 2b7e151628aed2a6abf7158809cf4f3c

3.2.35 mac get_dr

Response: a decimal string representing data rate used for LoRaWAN™, it can be from 0 to 6.
Purpose: Return data rate used for LoRaWAN™.
Example:
mac get_dr
>> 0

3.2.36 mac get_band

Response: a string representing current frequency list name, it can be **470, 868, 902, 915, 922, and 924**.

Purpose: Get current frequency list name.
Example:
mac get_band
>> 915

3.2.37 mac get_power

Response: a decimal string representing transmitting power in dBm.
Purpose: Return transmitting power.
Example:
mac get_power
>> 14

3.2.38 mac get_adr

Response: a string representing whether ADR is on or off. Return the state of ADR.
Purpose: Returned string can be: on, off. Example:
mac get_adr
>> on

3.2.39 mac get_txretry

Response: a decimal string representing retry number of transmission, it can be from **0** to **255**.
Purpose: Get retry number of transmission.
Example:
mac get_txretry
>> 8

3.2.40 mac get_rxdelay

Response: <rxdelay1> <rxdelay2>
<rxdelay1> - delay interval in **milliseconds** used for receive window 1, it can be from
0 to **65535**.
<rxdelay2> - delay interval in milliseconds used for receive window 2, it can be from
0 to **65535**.
Purpose: Get delay interval of receive window 1 and receive window 2.
Example:
mac get_rxdelay
>> 1000 2000

3.2.41 mac get_rx2

Response: <DR> <freq>
<DR> - data rate of second receive window, it can be **0** to **15**.
<freq> - operation frequency of second receive window in Hz, can be from **862000000** to **932000000**.
Purpose: Get data rate and operation frequency used for second receive window.
Example:
mac get_rx2
>> 0 868000000

3.2.42 mac get_sync

Response: a hexadecimal string representing current sync word. Default: **34**

Purpose: Return current sync word used for LoRaWAN™ communication.
Example:
mac get_sync
>> 12

3.2.43 mac get_ch_para <ChannelId>

<ChannelId>: a decimal string representing channel number, its value range depends on different regional band (e.g. EU868 range falls in 0 to 15; US902 range falls in 0 to 71).
Response: **<uplink frequency> <minimum DR> <maximum DR> <bandID> <downlink frequency>**, if <ChannelId> is valid.
<uplink frequency> - uplink frequency of specified channel in Hz, its range depends on "mac set_ch_frequency" command range.
<minimum DR> - minimum DR can be used, it can be from 0 to 6.
<maximum DR> - maximum DR can be used, it can be from 0 to 6.
<bandID> - a decimal string representing band number, it can be from 0 to 15
<downlink frequency> - downlink frequency of specified channel in Hz.
Invalid, if <ChannelId> string is not valid.
Purpose: Get the uplink frequency, maximum & minimum DR, dc band and downlink frequency of a specified channel.
Example:
mac get_ch_para 0
>> 868000000 0 5 0 0

3.2.44 mac get_ch_status <ChannelId>

<ChannelId>: a decimal string representing channel number, its value range depends on different regional band (e.g. EU868 range falls in 0 to 15; US902 range falls in 0 to 71).
Response: **on** or **off**, state of specified channel.
Invalid, if <ChannelId> string is not valid.

Purpose: Get state of specified channel. **on** means the channel is enabled, and **off** means the channel is disabled.
Example:
mac get_ch_status 0
>> on

3.2.45 mac get_dc_ctl

Response: state of duty cycle, **on** or **off**.
Default: **off**
Purpose: Get state of duty cycle checking. "**on"** means the checking is enabled, and "**off"** means the checking is disabled.
Example:
mac get_dc_ctl
>> on

3.2.46 mac get_dc_band <BandID>

<BandID>: a decimal string representing band number, can be from **0** to **15**.
Response: **<duty cycle>**, if <BandID> is valid.
<duty cycle> - duty cycle of specified band, can be from **0** to **65535**.

0: means 0%.
1-65535: duty cycle is equal to 1/<duty cycle>.
Invalid, if <BandID> string is not valid.

Purpose: Get frequency range and duty cycle of specified band. If a specific frequency is overlapped with more than one band ID, the smallest band ID will be selected. The default band setting of S76S is as following (Only 868 Band, other bands only have one Band ID 0):

Band ID	Duty Cycle
0	100 (1%)
1	100 (1%)
2	1000 (0.1%)
3	10 (10%)
4	100 (1%)
5	1 (100%)
6	1 (100%)
7	1 (100%)
8	1 (100%)
9	1 (100%)
10	1 (100%)
11	1 (100%)
12	1 (100%)
13	1 (100%)
14	1 (100%)
15	0 (0%)

Example:
mac get_dc_band 2
>> 1000

3.2.47 mac get_join_ch

Response: a list of channel ID for join request. Default: **0, 1 and 2**
Purpose: Get frequency channel ID for join request. The default channel ID for join request is **0, 1 and 2**.
Example:
mac set_join_ch
>> 0 1 2

3.2.48 mac get_upcnt

Response: uplink counter that will be used at next transmission. Default: **1**
Purpose: Get uplink counter that will be used at next transmission.
Example:
mac get_upcnt
>> 9

3.2.49 mac get_downcnt

Response: downlink counter that will be used at next transmission. Default: **0**
Purpose: Get downlink counter that will be used at next transmission.
Example:
mac get_downcnt
>> 5

3.2.50 mac get_class

Response: class type of LoRaWAN, can be **A** or **C**. Default: **A**

Purpose: Get class type of LoRaWAN™.
Example:
mac get_class
>> A

3.2.51 mac set_tx_mode <Cycle>

<Cycle>: A string representing TX signal would be sent continuously or not, it can be **cycle** or **no_cycle.**
Response: **Ok**, if <Cycle> string is valid.
Invalid, if <Cycle> string is not valid.
Purpose: **no_cycle** mode means no any TX signal would be sent after joining, user needs to send TX signal manually by "mac tx" command; **cycle** mode allows TX signal keep running, but the TX interval is assigned by other duty cycle related command.
Example:
mac set_tx_mode no_cycle
>> Ok

3.2.52 mac get_tx_mode

Response: A string representing TX signal would be sent continuously or not, it can be **cycle** or **no_cycle.**
Purpose: See "mac set_tx_mode" command.
Example:
mac get_tx_mode
>> cycle

3.2.53 mac set_batt <Battery>

<Battery>: a decimal string representing battery level indication, it can be from **0** to **255.**
Response: **Ok**, if < Battery > string is valid.
Invalid, if < Battery > string is not valid or out of range.
Purpose: It allows user to give a battery level which is complianted with DevStatusAns MAC command of LoRaWAN™ alliance.
Example:
mac set_batt 254
>> Ok

3.2.54 mac get_batt

Response: a decimal string representing battery level indication, it can be from **0** to **255.**
Purpose: A battery level which is complianted with DevStatusAns MAC command of LoRaWAN™ alliance.
Example:
mac get_batt
>> 254

3.2.55 mac set_tx_confirm <Confirm>

<Confirm>: A string representing whether S76S TX uplink needs server's ACK in downlink, it can be **on** or **off.**
Response: **Ok**, if <Confirm> string is valid.
Invalid, if <Confirm> string is not valid.
Purpose: Every uplink from devices like S76/78S can request the following downlink whether includes ACK filed. So user can use this command to decide it.

If <Confirm> is **on,** the next and later uplink all would requests ACK filed in downlink from server, which is following the behavior of LoRaWAN™ alliance.
Example:
mac set_tx_confirm on
>> Ok

3.2.56 mac get_tx_confirm

Response: A string representing whether the current S76S TX uplink needs server's ACK in downlink, it would be **on** or **off.**
Purpose: See "mac set_tx_confirm" command.
Example:
mac get_tx_confirm
>> Ok

3.2.57 (CN470 CLAA only) mac set_claa <claamode>

<Claamode>: A string representing an assigned CLAA mode, it can be **A, B, C, D** or **E.**
Response: **Ok**, if <Claamode> string is valid.
Invalid, if <Claamode> string is not valid.
Purpose: CLAA CN470-510 releases their own rule about behaviors when joining by OTAA or ABP. User can assign one of 5 modes before joining. And then the joining behavior would start the assigned mode when trying to join.
The detailed MAC commands description can refer to the related CLAA documents.
Example:
mac set_claa A
>> Ok

3.2.58 (CN470 CLAA only) mac get_claa

Response: A string representing the current CLAA mode, it would be **A, B, C, D** or **E.**
Purpose: See "mac set_claa" command.
Example:
mac get_claa
>> D

3.2.59 (CN470 CLAA only) mac set_getchinfo

Response: **Ok**.
Next packet sent to server will include a CLAA GetChInfoReq MAC command. The downlink of sent packet will contain the response of GetChInfoAcK MAC command. GetChInfoAcK response includes:
CLAAMode: the current using mode of CLAA gateway.
CHMap: gateway opened channels.
RX2CH: the channel number which can be used as RX2.
The detailed MAC commands description can refer to the related CLAA documents.
Example:
mac set_getchinfo
>> Ok

3.2.60 (CN470 CLAA only) mac set_gettimeinfo

Response: **Ok**.
Next packet sent to server will include a CLAA GetTimeInfoReq MAC command.

The downlink of sent packet will contain the response of GetTimeInfoAcK MAC command. GetTimeInfoAcK response includes:
Year: one of server date information.
Month: one of server date information.
Day: one of server date information.
Hour: one of server time information.
Minutes: one of server time information.
Second: one of server time information.
The detailed MAC commands description can refer to the related CLAA documents.
Example:
mac set_gettimeinfo
>> Ok

3.2.61 (CN470 CLAA only) mac set_jumboframe

Response: **Ok**.
Next packet sent to server will include a CLAA JumboframeReq MAC command. The downlink of sent packet will contain the response of JumboframeAck MAC command. The detailed MAC commands description can refer to the related CLAA documents.
Example:
mac set_gettimeinfo
>> Ok

3.2.62 mac set_lbt <Switch>

<Switch>: A string representing whether S76S enables its LBT feature, it could be used to listen the TX channel before executing TX uplink, it can be **on** or **off.**
Response: **Ok**, if <Switch> string is valid.
Invalid, if <Switch> string is not valid.
Purpose: By TELEC request, it needs to detect a channel is using or not before using this channel, so if LBT is on, S76S/S78S can listen before talk (LBT) the channel symbol signal strength before any TX uplink like joining or normal uplinks. If the channel is occupying, S76S/S78S would skip to another channel and LBT again until it finds an available channel.
Example:
mac set_lbt on

>> Ok

sip set_log debug

>> Ok

mac tx cnf 3 11223344
...
--> CAD found at 868100000 rssi is -30 dBm
(It means it found the symbol RSSI is too strong at 868.1MHz, it must be small than -80dBm)
(Change to another channel and LBT again)
--> CAD found at 868300000 rssi is -40 dBm
(It means it found the symbol RSSI is still too strong at 868.1MHz)
...

3.2.63 mac get_lbt

Response: A string representing the current S76S/S78S LBT setting, it would be **on** or **off.**
Purpose: See “mac set_lbt” command.
Example:
mac get_lbt
>> Ok

3.2.64 mac set_uplink_dwell <UplinkDwell>

<UplinkDwell>: A string representing UplinkDwell defined in LoRaWAN v1.0.2, it can be **on** or **off. On** means 400ms limit and **off** means no limit.
Response: **Ok**, if <UplinkDwell> string is valid.
Invalid, if <UplinkDwell> string is not valid.
Default value is **off.**
Set UplinkDwell defined in LoRaWAN v1.0.2. UplinkDwell would affect maximum payload size of each uplink DR. The command is only useful in when firmware is running at AS923 band.
Example:
mac set_uplink_dwell on

>> Ok

3.2.65 mac get_uplink_dwell

Response: A string representing the current UplinkDwell setting for AS923 band, it would be **on** or **off.**
Purpose: See “mac set_uplink_dwell” command.
Example:
mac get_uplink_dwell
>> off

3.2.66 mac set_downlink_dwell <DownlinkDwell>

<UplinkDwell>: A string representing DownlinkDwell defined in LoRaWAN v1.0.2, it can be **on** or **off. On** means 400ms limit and **off** means no limit.
Response: **Ok**, if <DownlinkDwell> string is valid.
Invalid, if <DownlinkDwell> string is not valid.
Default value is **off.**
Set DownlinkDwell defined in LoRaWAN v1.0.2. DownlinkDwell would affect maximum payload size of each downlink DR. The command is only useful in when firmware is running at AS923 band.
Example:
mac set_downlink_dwell on

>> Ok

3.2.67 mac get_downlink_dwell

Response: A string representing the current DownlinkDwell setting for AS923 band, it would be **on** or **off.**
Purpose: See “mac set_downlink_dwell” command.
Example:
mac get_downlink_dwell
>> off

3.2.68 mac set_max_eirp <MaxEIRP>

<MaxEIRP>: A decimal string representing MaxEIRP index defined in LoRaWAN v1.0.2, it can be **0** to **15.**

Response: **Ok**, if <MaxEIRP> string is valid.
Invalid, if <MaxEIRP> string is not valid.

Default value is **4**.

Set MaxEIRP Index defined in LoRaWAN v1.0.2. So this command is implemented for compliance in certain regulatory region. The relationship of MaxEIRP index and corresponding MaxEIRP is as following table:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MaxEIRP (dBm)	8	10	12	13	14	16	18	20	21	24	26	27	29	30	33	36

Example:
mac set_max_eirp 4

>> Ok

3.2.69 mac get_max_eirp

Response: A decimal string representing the current MaxEIRP index setting for certain band, it would be **0** to **15**.

Purpose: See “mac set_max_eirp” command.

Example:
mac get_max_eirp
>> 4

3.2.70 mac set_ch_count <ChannelsCount> <BW>

<ChannelsCount>: a decimal string representing channel count, it can only be **8, 16, 32, 48, 64, 80** and **96**.

<BW>: a decimal string representing which channels group different from bandwidth, it can only be **125** or **500**.

Response: **Ok**, if <ChannelID> and <Status> string is valid.
Invalid, if <ChannelID> and <Status> string is not valid.

Purpose: it allows to enable multiple channels, it’s similar with another command, “mac set_ch_status”, but it’s more effective. US902 ISM band has two uplink channels groups, one contains 64 channels that they are all running at BW 125 KHz and another group has 8 channesl which are all running at 500 KHz.

Example:
mac set_ch_count 16 125 (To enable 0~15th channels for 125KHz uplink channel group)

>> Ok

3.2.71 mac get_ch_count

Response: A decimal string representing the current enabled channels counts, its range would be from **0** to **96**.

Purpose: See “mac set_ch_count” command.

Example:

mac get_ch_count

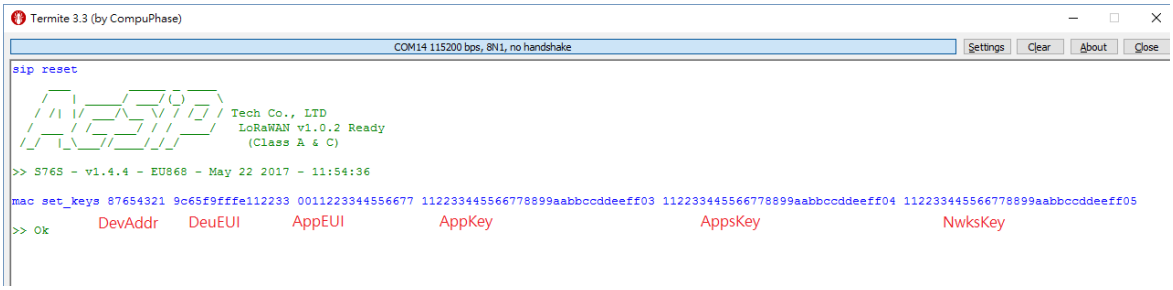
>> 8

3.2.72 mac set_keys <DevAddr> <DevEUI> <AppEUI> <AppKey> <AppsKey> <NwksKey>

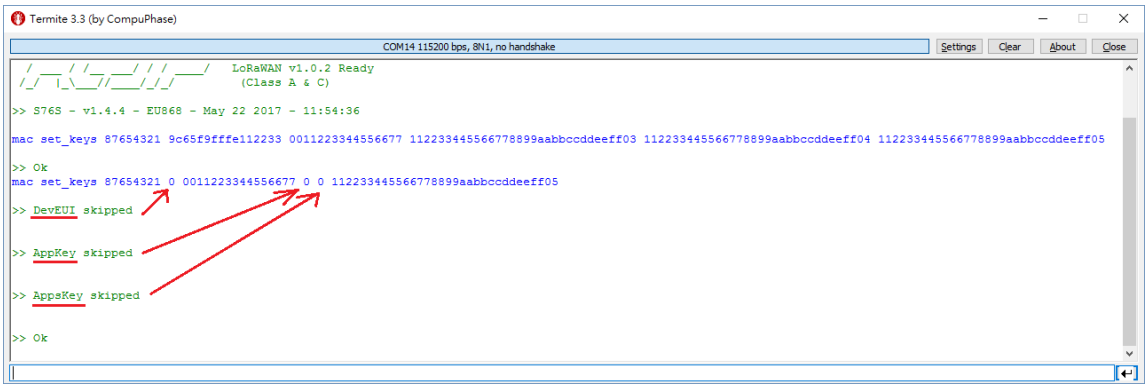
<DevAddr>: it follows “mac set_devaddr” command input format.
<DevEUI>: it follows “mac set_deveui” command input format.
<AppEUI>: it follows “mac set_appeui” command input format.
<AppKey>: it follows “mac set_appkey” command input format.
<AppsKey>: it follows “mac set_appskey” command input format.
<NwksKey>: it follows “mac set_nwkskey” command input format.

Purpose: After this command is executed, the 6 keys would be updated and stored into EEPROM immediately without calling “mac save”.

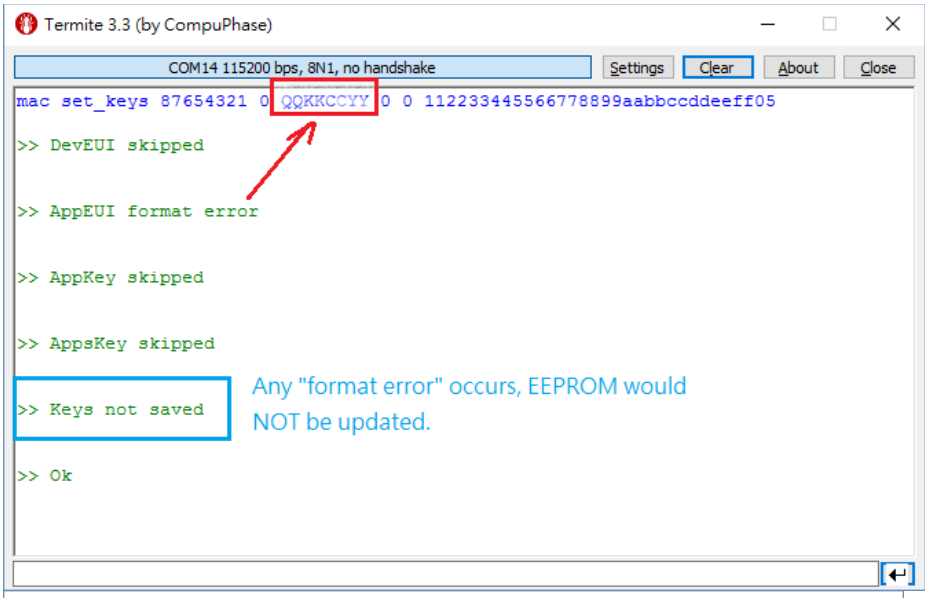
Response: **Ok**, if input ASCII strings are all valid.
Invalid, if one of input strings is not valid.
Example:

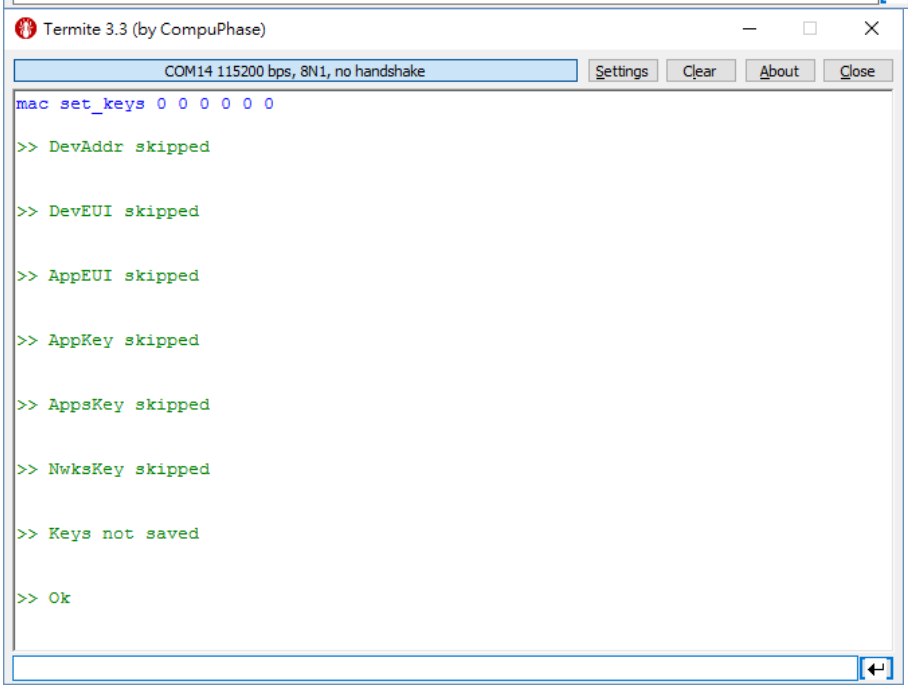


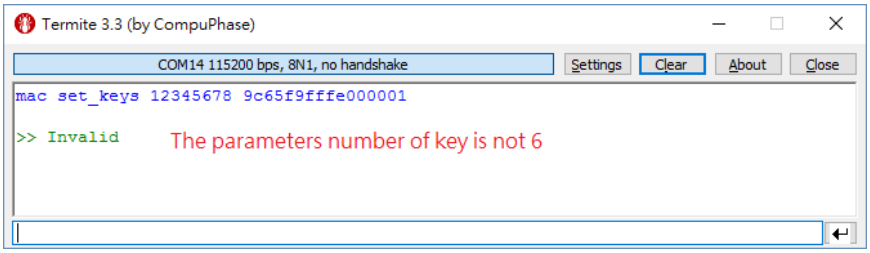
If user just wants to set one or two keys (Not all 6 keys), caller can let the rest of keys be “0”. The “0” value of a certain key would not be modified.
(e.g. the below shows DevEUI, AppKey and AppsKey won’t be modified, others are updated and also stored into EEPROM)



(Incorrect examples)
Any incorrect input parameters occur, the value of keys would NOT be updated into EEPROM.







3.2.73 mac set_tx_interval <Interval>

<Interval>: A decimal string representing TX (LoRaWAN Uplink) interval (ms), it can be between **5000** to **86400000 (86400000 is not included; 86399999 is OK)**.

Response: **Ok**, if <Interval> string is valid.
Invalid, if <Interval> string is not valid.

Default value is **5000**.

Purpose: User can assign the interval between two TXs (LoRaWAN Uplinks) when it's under tx cycle mode and the duty cycle control is turned off.

Example:
mac set_tx_mode no_cycle

>> Ok
mac set_dc_ctl off

>> Ok
mac set_tx_interval 6000

>> Ok
(The uplinks of LoRaWAN would be uploaded automatically in every 6s)

3.2.74 mac get_tx_interval

Response: A decimal string representing the current TX intval setting value, its range would be from **5000** to **86399999**.

Purpose: See "mac set_tx_interval" command.

Example:
mac get_tx_interval

>> 6000

3.2.75 mac set_rx1_freq <Rx1_Freq_Begin> <Rx1_Step> <Rx1_Count>

<Rx1_Freq_Begin>: a decimal string representing the beginning of setting rx1 frequency in Hz, it can be set from **902000000** to **932000000** (US915); from **470000000** to **510000000** (CN470); from **433000000** to **932000000** (other region); "**0**" value can let Tx/Rx1 frequency be set back to the identical frequency value (Rx1 would follow the "mac set_ch_freq" value, same as Tx1 frequency).
<Rx1_Step>: a decimal string representing the incremental frequency step, it can be assigned from **0** to **600000**. The normal usage is 20000 when BW is set at 125KHz.
<Rx1_Count>: a decimal string representing the RX1 channels count, it can't exceed the maximum allowed channels number defined in LoRaWAN v1.0.2 regional parameters setting.

Response: **Ok**, if <ChannelID> and <Status> string is valid.
Invalid, if <ChannelID> and <Status> string is not valid.

Purpose: It allows to set RX1 frequency for multiple channels just by one command. The 1st RX1 frequency is beginning from <Rx1_Freq_Begin>, the next 2nd RX1 would be set at <Rx1_Freq_Begin> + <Rx1_Step> * 1, the 3rd RX1 is <Rx1_Freq_Begin> + <Rx1_Step> * 2; So the Nth Rx1 channel frequency would be <Rx1_Freq_Begin> + <Rx1_Step> * (<Rx1_Count> - 1);

Example:
(Set RX1 downlink frequency from 500.3MHz, incremental frequency is 200 KHz, sets Channel 0 to Channel 7)

mac set_rx1_freq 500300000 200000 8

>> Ok
(List the result of setting above)
mac get_ch_para 0

>> 470300000 0 5 0 500300000
mac get_ch_para 1

>> 470500000 0 5 0 500500000
mac get_ch_para 2

>> 470700000 0 5 0 500700000
mac get_ch_para 3

>> 470900000 0 5 0 500900000
mac get_ch_para 4

>> 471100000 0 5 0 501100000
mac get_ch_para 5

>> 471300000 0 5 0 501300000
mac get_ch_para 6

>> 471500000 0 5 0 501500000
mac get_ch_para 7

>> 471700000 0 5 0 501700000

(e.g. TX & RX1 using identical frequency setting)
mac set_rx1_freq 0

>> Ok
mac get_ch_para 0

>> 470300000 0 5 0 0 (※ The last "0" mean the downlink frequency RX1 is the same as TX)
mac get_ch_para 1

>> 470500000 0 5 0 0
mac get_ch_para 2

>> 470700000 0 5 0 0

3.2.76 mac get_rx1_freq

Response: three decimal string representing the Rx1 related setting mentioned in "mac set_rx1_freq" command.

Purpose: See "mac set_rx1_freq" command.

Example:
mac get_rx1_freq

>> 500300000 200000 8

3.2.77 mac set_auto_join <Switch> <Join_Type> <Join_Count>

<Switch>: A string representing whether auto join mode is **on** or **off**.
Response: **Ok**, if < Swtich > string is valid
Invalid, if < Switch > string is not valid.
<Join_Type>: A string representing the selected join type of LoRaWAN, it can be **otaa** (over-the-air activation) or **abp** (activation by personalization).
<Join_Count>: If <Join_Type> is selected as otaa, <Join_Count> can be **1** to **65535** and its meaning is the re-try times of OTAA when it's failed to join; If joining by ABP, the <Join_Count> is un-necessary and leave it empty.

Purpose: When using remote mode, user might want to let S76S start to join automatically after rebooted by "sip reset" or power off/on. By setting this commands and then execute "mac save", the next boot-up would execute the joining behavior by the previous auto join setting.

Example:
(The next boot-up, it would try to join by OTAA for three times)
mac set_auto_join on otaa 3

>> Ok
(The next boot-up, it would try to join by ABP (ABP only needs one-time joining)
mac set_auto_join on abp

>> Ok
(Disable Auot Join behavior)
mac set_auto_join off

>> Ok
(Don't forget to save the settings, or it would not take effect after reboot)
mac save

>> Ok

3.2.78 mac get_auto_join

Response: To indicate the current setting of auto join mode, please see the below demo example.

Purpose: See "mac set_auto_join" command.

Example:
mac set_auto_join on otaa 3

>> Ok
mac get_auto_join

>> otaa 3
mac set_auto_join on abp

>> Ok
mac get_auto_join

>> abp

mac set_auto_join off

>> Ok
mac get_auto_join

>> off

3.3 RF commands

3.3.1rf tx <Data>

<Data>: a hexadecimal string representing data to be transmitted.
Response: there are two responses after entering this command. The first response used to indicate that whether command is valid or not, will be received after entering command. The second response will be received after transmitted.

Maximum transfer length: 255 Bytes.

First response: **Ok**, if <Data> string is valid.
Invalid, if <Data> string is not valid.
Second response: **radio_tx_ok**, if transmission is successful.
radio_err, if transmission is failed.
Example:
rf tx 5ab69f
>> Ok
>> radio_tx_ok

3.3.2rf rx <RxWindowTime>

<RxWindowTime>: a decimal string representing receiving window in milliseconds, can be from **0** to
65535. **0** means waiting until receiving a packet.

Response: there are two responses after entering this command. The first response, it used to indicate that whether command is valid or not, will be received after entering command. The second response will be received after received a packet or time out occurred.

First response: **Ok**, if <RxWindowTime> string is valid.
Invalid, if <RxWindowTime> string is not valid.
Second response: **radio_rx** <data> <rssi> <snr>, if reception is successful.
<data> - received data representing in hexadecimal.
<rssi> - received signal strength in decimal.
<snr> - received signal-to-noise value in decimal.
raido_err, if reception failed or time out occurred.
Example:
rf rx 1000
>> Ok
>> radio_rx 5432 -90 -50

3.3.3rf set_freq <Frequency>

<Frequency>: a decimal string representing communication frequency in Hz, it can be values from **862000000** to **932000000 (868 to 924 bands); 137000000** to **525000000 (433 or 470 band)**.
Response: **Ok**, if <Frequency> string is valid
Invalid, if <Frequency> string is not valid.
Set current communication frequency.

Example:
rf set_freq 915000000
>> Ok

3.3.4rf set_pwr <Power>

<Power>: a decimal string representing transmitting power in dBm, it can be from **2** to **20**.

Response: **Ok**, if <Power> string is valid
Invalid, if <Power> string is not valid.

Set current transmitting power.

Example:
rf set_pwr 14
>> Ok

3.3.5rf set_sf <SpreadingFactor>

<SpreadingFactor>: a string representing spreading factor used for communication, it can be: **7, 8, 9, 10, 11 and 12**.

Response: **Ok**, if <SpreadingFactor> string is valid
Invalid, if <SpreadingFactor> string is not valid.

Set current spreading factor.

Example:
rf set_sf 8
>> Ok

3.3.6rf set_bw <BandWidth>

<BandWidth>: a string representing signal bandwidth in kHz, it can be: **125, 250, 500**.

Response: **Ok**, if <BandWidth> string is valid
Invalid, if <BandWidth> string is not valid.

Set current signal bandwidth.

Example:
rf set_bw 250
>> Ok

3.3.7rf set_cr <CodingRate>

<CodingRate>: a string representing coding rate, can be: **4/5, 4/6, 4/7, 4/8**.

Response: **Ok**, if <CodingRate> string is valid
Invalid, if <CodingRate> string is not valid.

Set current coding rate used for communication.

Example:
rf set_cr 4/5
>> Ok

3.3.8rf set_prlen <PreambleLength>

<PreambleLength>: a decimal string representing preamble length, it can be from **0** to **65535**. Response: **Ok**, if <PreambleLength> string is valid
Invalid, if <PreambleLength> string is not valid.

Set current preamble length.

Example:
rf set_prlen 12
>> Ok

3.3.9rf set_crc <State>

<State>: a string representing whether the CRC header is **on** or **off**.
Response: **Ok**, if <State> string is valid
Invalid, if <State> string is not valid.
Set current status of the CRC header.
Example:
rf set_crc on
>> Ok

3.3.10 rf set_iqi <Invert>

<Invert>: a string representing whether the Invert IQ functionality is **on** or **off**.
Response: **Ok**, if <Invert> string is valid
Invalid, if <Invert> string is not valid.
Set the status of Invert IQ functionality.
Example:
rf set_iqi off
>> Ok

3.3.11 rf set_sync <SyncWord>

<SyncWord>: a hexadecimal string representing sync word, it can be from **0** to **FF**.
Response: **Ok**, if <SyncWord> string is valid
Invalid, if <SyncWord> string is not valid.
Set the sync word used for communication.
Example:
rf set_sync 12
>> Ok

3.3.12 rf save

Response: **Ok**
Save p2p configuration parameters to EEPROM.
Example:
rf save
>> Ok

3.3.13 rf get_freq

Response: a decimal string representing communication frequency in Hz.
Default value: **922500000**
Return current communication frequency.
Returned string can be from **862000000** to **932000000 (868 to 924 bands); 137000000 to 525000000 (433 or 470 band)**.
Example:
rf get_freq
>> 922500000

3.3.14 rf get_pwr

Response: a decimal string representing transmitting power in dBm.
Default: **14**
Return current transmitting power. Returned string can be from **2** to **20**.
Example:
rf get_pwr
>> 14

3.3.15 rf get_sf

Response: a string representing spreading factor used for communication.
Default: **7**
Return current spreading factor. Returned string can be: **7, 8, 9, 10, 11 and 12**.
Example:
rf get_sf
>> 7

3.3.16 rf get_bw

Response: a string representing signal bandwidth in kHz.
Default: **125**
Return current signal bandwidth. Returned string can be: **125, 250 and 500**.
Example:
rf get_bw
>> 125

3.3.17 rf get_prlen

Response: a decimal string representing preamble length.
Default: **12**
Return current preamble length. Returned strings can be from **0** to **65535**.
Example:
rf get_prlen
>> 12

3.3.18 rf get_crc

Response: a string representing whether the CRC header is **on** or **off**.
Default: **on**
Return current status of the CRC header. Returned string can be: **on**, **off**.
Example:
rf get_crc
>> on

3.3.19 rf get_iqi

Response: a string representing whether the Invert IQ functionality is **on** or **off**.
Default: **off**
Return current status of the Invert IQ functionality. Returned string can be: **on, off.**
Example:
rf get_iqi
>> off

3.3.20 rf get_cr

Response: a string representing current coding rate.
Default: **4/6**
Return current coding rate used for communication. Returned string can be: **4/5, 4/6, 4/7, 4/8**.
Example:
rf get_cr
>> 4/6

3.3.21 rf get_sync

Response: a hexadecimal string representing current sync word.
Default: **12**
Return current sync word used for communication.
Example:
rf get_sync
>> 12

3.3.22 rf rx_con <Continuous>

<Continuous>: a string representing whether Rx continuous mode is **on** or **off**.
Response: **Ok**, if < Continuous > string is valid
Invalid, if < Continuous > string is not valid.
Set Rx continuous mode can be **on** or **off**.
Example:
rf rx_con on
>> Ok

3.3.23 (production verification) rf fsk <Switch>

<Switch>: a string representing whether FSK mode is **on** or **off**.
Response: **Ok**, if < Swtich > string is valid
Invalid, if < Switch > string is not valid.
Set RF FSK mode can be **on** or **off**. If value is **on**, the RF FSK mode is enabled end keep TX emitting by using FSK mode.
And it's not allowable to send others command when executing FSK mode beside "*rf fsk off*"
Example:
rf fsk on
>> Ok
sip reset
>> FSK running
rf fsk off
>> Ok

3.3.24 rf lora_tx_start <Times> <Interval> <Data>

<Times>: a decimal string representing how many time of TX counts, it can be values from **0** to **100000**, "0" means TX would not stop until "*rf lora_tx_stop*" send.
<Interval>: a decimal string representing LoRa TX interval in **ms**, it can be values from **3** to **300000**.
<Data>: a hexadecimal string representing data to be transmitted. The maximum transfer length: **255** bytes

Response: there are two responses after entering this command. The first response used to indicate that whether command is valid or not, it will be received after entering command. The second response will be received after every successful LoRa TX.

First response: **Ok**, if <Data> string is valid.
Invalid, if <Data> string is not valid.
Second response: **rf lora_tx(N)**, if transmission is successful and N means how many time TX already emits successfully.

Start LoRa TX after executing this command.

Example: (TX times is 100, TX interval is 200ms, TX data is 0xaa, 0xaa, 0x55, 0x55 that is 4 bytes)
rf lora_tx_start 100 200 aaaa5555
>> Ok
>> rf lora_tx(1)

>> rf lora_tx(10)

>> rf lora_tx(20)
...
Note: (rf lora_tx(n) would be only shown when n is 1, 10, 20, 30, ...)

3.3.25 rf lora_tx_stop

First response: **rf lora_tx=N**, N means how many times TX had already emitted by using "*rf lora_tx_start*" command. If N is 0, it means no any successful TX or no any "*rf lora_tx_start*" ever sent.
Second Response: **Ok**, if command is correct and no any argument sees.
Invalid, if any argument is given.

Stop LoRa TX which started from "rf lora_tx_start".

Example:
rf lora_tx_stop
>> rf lora_tx=10

>> Ok

3.3.26 rf lora_rx_start <Data>

<Data>: a hexadecimal string representing that demands to be matched. Max length limitation is 255 bytes.

Response: there are two responses after entering this command. The first response, it used to indicate that whether command is valid or not, will be received after entering command. The second response will be received after received a packet.

First response: **Ok**, if <RxWindowTime> string is valid.
Invalid, if <RxWindowTime> string is not valid.
Second response: **rf lora_rx_start**(<num>) **rssi**(<rssi>) **snr**(<snr>), if reception is successful.
<num> - received data representing in hexadecimal.
<rssi> - received signal strength in decimal.
<snr> - received signal-to-noise value in decimal.
Example:
rf lora_rx_start aaaa5555
>> Ok
>> rf lora_rx(1) rssi(-96) snr(32)

>> rf lora_rx(10) rssi(-96) snr(30)

>> rf lora_rx(20) rssi(-97) snr(30)

Note: (rf lora_rx(n) would be only shown when n is 1, 10, 20, 30, ...)

3.3.27 rf lora_rx_stop

First response: **rf lora_rx=N**, N means how many times RX had already received by using "*rf lora_rx_start*" command. If N is 0, it means no any successful RX (or has RX but payload data dis-matched) or no any "*rf lora_rx_start*" ever sent.
Second Response: **Ok**, if command is correct and no any argument sees.
Invalid, if any argument is given.

Stop LoRa RX which started from "rf lora_rx_start".

Example:
rf lora_rx_stop
>> rf lora_rx=51

>> Ok

3.3.28 rf set_fdev <FreqDeviation>

<FreqDeviation>: a string representing frequency deviation used only for FSK communication (rf fsk command), it can be from **0** to **65535**.
Response: **Ok**, if <FreqDeviation> string is valid
Invalid, if <FreqDeviation> string is not valid.
Set current frequency deviation value which is only used for FSK, the default value is **0**.
Example:
rf set_fdev 100

>> Ok

3.3.29 rf get_fdev

Response: a string representing frequency deviation used for FSK communication.
Default: **0**
Return the current frequency deviation value.
Example:
rf get_fdev

>> 0

3.3.30 rf set_cad <Switch>

<Switch>: A string representing whether S76S enables its CAD feature, it could be used to listen a RF channel when executing "rf lora_tx_start" command, it can be **on** or **off.**
Response: **Ok**, if <Switch> string is valid.
Invalid, if <Switch> string is not valid.

Purpose: SX1276/SX1278 of S76S/S78S has embedded CAD feature, it can execute a channel activation detection (CAD) behavior to detect the channel

symbol signal strength. If the channel is occupying by another RF device (not just only LoRa signal but also RF signal running at the particular frequency if its power strength is larger than -80dBm), S76S/S78S could stop TX while in the middle of executing "rf lora_tx_start" until it finds the using channel is available.

Example:
rf set_cad on

>> Ok

rf lora_tx_start 0 50 11223344
...
>> paused by cad 922500000 10 times, avg rssi is -56 dBm
(It means it found the symbol RSSI is too strong at 922.5MHz, it must be small than -80dBm)
(The average RSSI value in these 10 times is around -56 dBm)
...

3.3.31 rf get_cad

Response: A string representing the current S76S/S78S CAD setting, it would be **on** or **off.**
Purpose: See "rf set_cad" command.
Example:
rf get_cad
>> Ok

3.3.32 rf cad <Frequency> <SF> <BW> <SyncWord> <SkipRX>

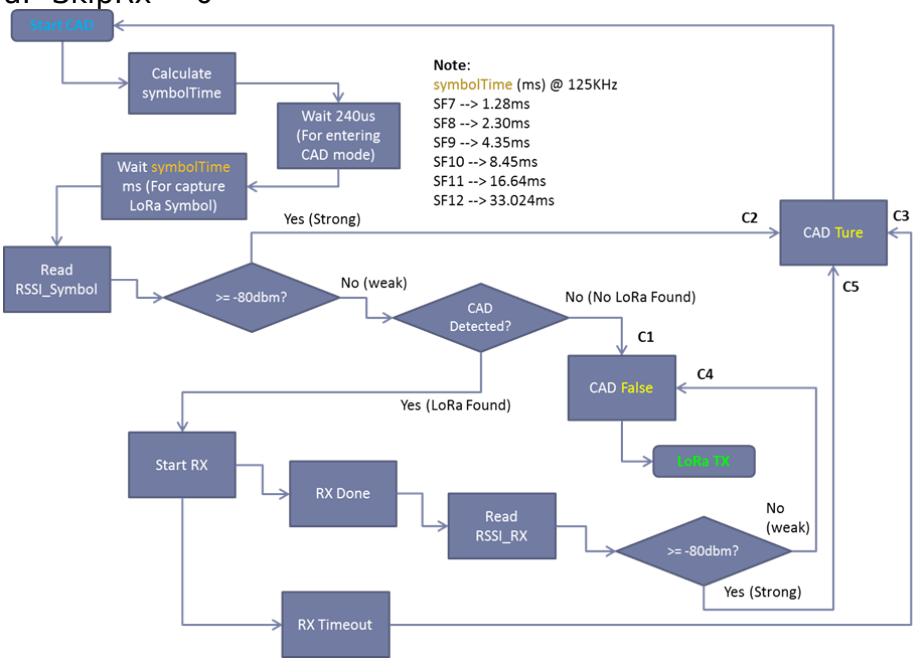
<Frequency>: a decimal string representing operation frequency of specified channel in Hz, it can be from **862000000** to **932000000** (868, 902-924 band); from **137000000** to **525000000** (470 band).
<SF>: a string representing spreading factor used for communication, it can be: **7**, **8**, **9**, **10**, **11** and **12**
<BW>: a string representing signal bandwidth in kHz, it can be: **125**, **250** and **500**.
<SyncWord>: a hexadecimal string representing sync word, it can be from **0** to **FF**
<SkipRX>: a decimal string representing whether it can skip checking RX packet RSSI or not. "**1**" means to skip RX and "**0**" means not to skip RX and try to get RX RSSI value when a RX packet is received.

Response: there are several responses. If parameters are all valid, it would not show "invalid or Too many arguments!" immediately. Instead, the following response used to indicate the CAD result.

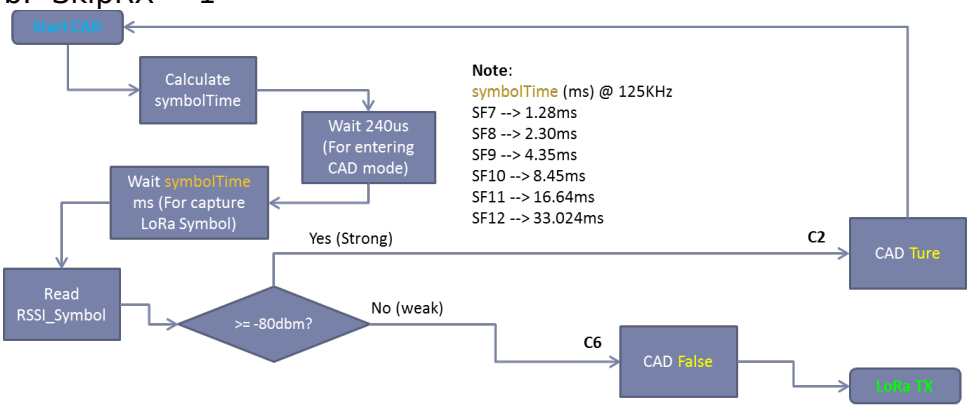
First Response if something goes wrong:
Invalid, if upper string are not valid.
Too many arguments!, if input parameter is too much.
Second Response if RSSI Symbol is strong enough (and large than -80dBm):
>> RSSI Symbol = -59, RSSI RX = -300
(The 1st RSSI symbol means the signal strength of the LoRa signal or other RF signal occupying on this frequency, the 2nd RSSI RX means the RX packet signal strength when the packet is received by LoRa protocol)

>> CAD Result: RSSI Symbol Weak & Skip Rx, C6
>> CAD Result: RSSI Symbol Strong, C2, try CAD again
(The below figures show the possibilities of the different kind of CAD result)

a. SkipRx = 0



b. SkipRX = 1



Example1: (CAD at 922500000, SF10, BW125, SyncWord is 0x12 and SkipRx is 1 and CAD found)
rf cad 922500000 10 125 12 1

>> RSSI Symbol = -59, RSSI RX = -300

>> CAD Result: RSSI Symbol Strong, C2, try CAD again

>> Ok

Example2: (CAD at 922500000, SF10, BW125, SyncWord is 0x12 and SkipRx is 1 and CAD didn't find any signal stronger than -80 dBm)
rf cad 922500000 10 125 12 1

>> CAD Result: RSSI Symbol Weak & Skip Rx, C6

>> Ok

3.4 RM commands

3.4.1rm set_gpio <Direction> <Serial_Number> <Pin_Group> <Pin_Number>

<Direction>: A string representing STM32 GPIO pin mode, it can be assigned as output or input, set "**out"** would let a certain remote mode GPIO pin be output mode and **"in"** let it as input mode.
<Serial_Number>: "**0**" or "**1**", to indicate the serial number of using in remote mode GPIO, to report or control GPIO state.
<Pin_Group>: A string representing STM32 GPIO pin groups, it can be these characters **A, B, C, D, E, F** and **H** (note: no G).
<Pin_Number>: A decimal string representing STM32 GPIO pin number, it can be set from **0** to **15**

Purpose: To assign one of remote mode GPIO pin, to let it be one of STM32 MCU GPIO pin. For example, if <Direction> is "in" & <Serial_Number> is "0", the "IN0" symbol of using in remote mode would be designated a GPIO report pin, and the corresponding pin number would come from these two following parameters: <Pin_Group> and <Pin Number>.

Response: **Ok**, if input arguments are valid.
Invalid, if input argument are not valid or out of range.
Example:
(Set GPIO PC_9 as IN0 using in remote mode)
rm set_gpio in 0 C 9

>> Ok
(Set GPIO PA_8 as OUT1 using in remote mode)
rm set_gpio out 1 A 8

>> Ok

3.4.2rm get_gpio

Response: Four string representing the current setting of remote mode GPIO, it would show the serial number setting (IN0, IN1, OUT0 and OUT1) by sequentially. If it shows "NC", it means there is no setting to the corresponding GPIO serial number setting.

Purpose: To show the current setting that is picked from the result of "rm set_gpio" command.

Example:
(IN0 is set as PC_9, OUT0 is set as PC_4, OUT1 is set as PC_5 and there is no any setting toward IN1)
rm get_gpio

>> PC_9 NC PC_4 PC_5

3.4.3rm set_gpio_swtich <Switch>

<Switch>: A string representing whether remote mode GPIO report & control feature is enabled or disabled, it only allows "**on**" or "**off**" strings. If <Switch> is "**on**", the report or control feature of remote mode is enabled; if <Switch> is **off**, there is no GPIO report or control feature for remote mode even user had already executed "rm set_gpio" command.

Purpose: To enable or disable GPIO report & control remote mode.

Response: **Ok**, if <Switch> string is valid.
Invalid, if <Switch> string is not valid.

Example:
rm set_gpio_switch on

>> Ok

3.4.4rm get_gpio_swtich

Response: A string representing the current setting of remote mode GPIO switch value.

Purpose: To show the current setting that is determined from the result of "rm set_gpio_switch" command.

Example:
rm get_gpio_switch

>> on

3.4.5rm set_adc <Serial_Number> <Switch>

<Serial_Number>: "**0**" or "**1**", to indicate the serial number of using in remote mode ADC, to report ADC state to server.
<Switch>: A string representing whether an independent ADC channel report feature is enabled or disabled, it only allows "**on**" or "**off**" strings. If <Switch> is "**on**", the assigned ADC channel (selected by <Serial_Number> value) report feature of remote mode is enabled; if <Switch> is **off**, there is no ADC report feature for this ADC channel.

Purpose: To assign one of remote mode ADC channel, to let it be one of STM32 MCU ADC fixed pin. For example, if <Serial_Number> is "0", the "ADC0" (ADC channel 0) of using in remote mode would be designated a ADC report pin PA_0; If <Serial_Number> is "1", the "ADC1" (ADC Channel 1) of using in remote mode would be designated a ADC report pin PB_0.

Note: Currently, the corresponding ADC0 & ADC1 channel is using two fixed GPIO pin number which are PA_0 & PB_0 pins individually.

Response: **Ok**, if input arguments are valid.
Invalid, if input argument are not valid or out of range.

Example:

(Enable ADC Channel 0 which its GPIO pin is fixed at PA_0 using in remote mode)
rm set_adc 0 on

>> Ok
(Disable ADC Channel 1 which its GPIO pin is fixed at PB_0 using in remote mode)
rm set_adc 1 off

>> Ok

3.4.6rm get_adc <Serial_Number>

<Serial_Number>: "**0**" or "**1**", to indicate the serial number of using in remote mode ADC, "0" means ADC0 and "1" means ADC1.

Response: A string representing the current setting of ADC channel individual switch value for remote mode. "**on**" means the corresponding ADC switch is enabled; "**off**" means it's disabled.

Purpose: To show the current setting that is determined from the result of "rm set_adc" command.

Example:
rm get_adc 0

>> on
rm get_adc 1

>> off

3.4.7rm set_adc_switch <Swtich>

<Switch>: A string representing whether remote mode ADC report feature is enabled or disabled, it only allows "**on**" or "**off**" strings. If <Switch> is "on", the ADC report feature of remote mode is enabled; if <Switch> is off, there is no ADC report feature for remote mode even if user had already enable a certain ADC channel for remote mode previously set by "rm set_adc" command.

Purpose: To enable or disable overall ADC report feature for remote mode.

Response: **Ok**, if <Switch> string is valid.
Invalid, if <Switch> string is not valid.

Example:
(Enable ADC 2 channls report mode feature, but the individual ADC switch value still controlled by "rm set_adc" command)
rm set_adc_switch on

>> Ok

3.4.8rm get_adc_switch

Response: A string representing the current setting of remote mode ADC switch value.

Purpose: To show the current setting that is determined from the result of “rm set_adc_switch” command.

Example:
rm get_adc_switch

>> on

3.4.9rm set_port_uplink <Port>

<Port>: a decimal string representing port number used for remote mode LoRaWAN uplink, its range can be set from **1** to **223.**

Purpose: To set the LoRaWAN uplink port number when uploading report data up to server under remote mode.

Response: **Ok**, if input arguments are valid.
Invalid, if input argument are not valid or out of range

Exampe:
(To use LoRaWAN port 200 as uplink port)
rm set_port_uplink 200

>> Ok

3.4.10 rm set_port_downlink <Port>

<Port>: A decimal string representing port number used for remote mode LoRaWAN downlink, its range can be set from **1** to **223.**

Purpose: To set the LoRaWAN downlink port number when downloading control data that is sent from server under remote mode.

Response: **Ok**, if input arguments are valid.
Invalid, if input argument are not valid or out of range

Exampe:
(To use LoRaWAN port 201 as downlink port)
rm set_port_uplink 201

>> Ok

3.4.11 rm get_port

Response: Two decimal strings representing the current setting of LoRaWAN uplink & downlink port values using for remote mode.

Purpose: To show the current setting that is determined from the result of “rm set_port_uplink” & “rm set_port_downlink” commands.

Example:
rm get_port

>> 200 201

3.4.12 rm set_mode <Mode>

<Mode>: A string representing the remote mode operating mode is set to either "Cycle", "Trigger" or "Off" mode, so the string can be **cycle**, **trigger** or **off**.

Response: **Ok**, if <Mode> string is valid.
Invalid, if <Mode> string is not valid.

Purpose: The LoRaWAN uplinks for reporting data up to server can be triggered by a fixed time interval or an external GPIO pin changing its state. So this command allows user to use, to set the remote mode operating mode.

Example:
(Set Report Mode as "Cycle" mode, it can report GPIO & ADC data in every LoRaWAN uplinks periodically)
rm set_mode cycle

>> Ok
(Set Report Mode as "Trigger" mode, it can report GPIO & ADC data when detecting Rasing/Falling/Both trigger type ␣ Depends on "rm set_trigger" command)
rm set_mode trigger

>> Ok

3.4.13 rm get_mode

Response: A string representing which remote mode operation mode is assigned; It can be **cycle**, **trigger** or **off.**

Purpose: See "rm set_mode" command.

Example:
rm get_mode

>> cycle

3.4.14 rm set_trigger <Pin_Group> <Pin_Number> <Trigger_Type>

<Pin_Group>: A string representing the remote mode trigger pin group, it can be these characters **A, B, C, D, E, F** and **H** (note: no G).
<Pin_Number>: A decimal string representing the remote mode trigger pin number, it can be set from **0** to **15**
<Trigger_Type>: A string representing the trigger pin detection waveform type, it can be "**rising**", "**falling**" or "**both**".

Purpose: To assigned a GPIO pin as trigger pin under remote mode trigger mode, and also set its detection type.

Response: **Ok**, if input arguments are valid.

Invalid, if input argument are not valid or out of range

Example:
(Set GPIO PC_7 as an external trigger pin that can detect a rising type signal)
rm set_trigger C 7 rising

>> Ok

3.4.15 rm get_trigger

Response: Two strings representing the current setting of trigger pin name and detection type.

Purpose: To show the current setting that is determined from the result of "rm set_trigger" command.

Example:
rm get_trigger

>> PC_7 rising

4. Example

This section gives several complete examples on how to use AcSiP command interface. All examples include many comments followed by double slash. This comments are for clearly explanation and should not be inputted to S76S through command interface

4.1 LoRaWAN™

4.1.1 ABP

// Set channel frequency channel number and frequency depends on server configuration
mac set ch_freq 0 926500000
>> Ok
mac set ch_freq 1 926700000
>> Ok
mac set ch_freq 2 926900000
>> Ok
...

// Set following according to LoRaWAN configuration
mac set_devaddr 00220009
>> Ok
mac set_nwkskey 965F6942F29C9EBE5747E25F07DA5114
>> Ok
mac set_appskey A46847D184323C21C992D8F9EF4B7CE9
>> Ok

// Activation by Personalization
mac join abp
>> Ok
>> accepted

// Send unconfirmed uplink on port 15

mac tx ucnf 15 1234
>> Ok
>> tx_ok

4.1.2 OTAA

// Set channel frequency channel number and frequency depends on server configuration
mac set_ch_freq 0 926500000
>> Ok
mac set_ch_freq 1 926700000
>> Ok
mac set_ch_freq 2 926900000
>> Ok
...

// Set following according to LoRaWAN configuration
mac set_deveui 9c65f9fffeabcd12
>> Ok
mac set_appeui 70B3D57ED000059E
>> Ok
mac set_appkey C1FE94B0F5F6A50E83015B3C45C933A9
>> Ok

// Over-the-Air Activation
mac join otaa
>> Ok
>> accepted

// Send unconfirmed uplink on port 15
mac tx ucnf 15 1234
>> Ok
>> tx_ok

//Auto Join Mode
(The next boot-up, it would try to join by OTAA for three times.)
mac set_auto_join on otaa 3

(The next boot-up, it would try to join by ABP (ABP only needs one-time joining)
mac set_auto_join on abp

(Disable Auot Join behavior)
mac set_auto_join off

(Don't forget to save the last setting above, or it would not take effect after reboot)
mac save

(Reset node and then Auto Join Mode starts)
sip reset

4.1.3 Confirmed Uplink and Downlink

// Send confirmed uplink on port 15
mac tx cnf 15 1234 // Send 0x12, 0x34 to server
>> Ok

```
>> tx_ok

mac tx cnf 15 1234
>> Ok
>> err                    // Fail to get confirm from server

mac tx cnf 15 1234
>> Ok
>> rx 15 6432             // Receive downlink (0x64, 0x32) from server on port 15
```

4.2 Node to Node

```
rf set_sync 12        // Set SyncWord to 0x12
>> Ok
rf set_freq 926500000 // Set frequency to 926500000Hz
>> Ok
rf set_sf 7           // Set spreading factor to 7
>> Ok
rf set_bw 125         // Set bandwidth to 125KHz
>> Ok
...
// Send LoRa packet
rf tx 1234567890
>> Ok
>> radio_tx_ok

// Receive LoRa packet
rf rx 10000   // Open an 10s receive window
>> Ok
>> radio_rx 1234567890 -90 7      // Received data, RSSI and SNR
```

4.3 Remote mode

4.3.1 Report GPIO, ADC Data & Uplink to Server

Report 2 GPIO and 2 ADC upon to Server under Cycle Mode by Fixed TX Interval

(Setting)
rm set_gpio in 0 C 9 ⏎ Set GPIO PC_9 as IN0
rm set_gpio in 1 C 8 ⏎ Set GPIO PC_8 as IN1
rm set_gpio_switch on ⏎ Enable GPIO*2 Report Mode
rm set_adc 0 on ⏎ Enable ADC Channel 0 (Fixed at PA_0)
rm set_adc 1 on ⏎ Enable ADC Channel 1 (Fixed at PB_0)
rm set_adc_switch on ⏎ Enable ADC*2 Report Mode
rm set_port_uplink 200 ⏎ To use LoRaWAN port 200 as uplink port.

(Set operating mode & start)
rm set_mode cycle ⏎ Set Report Mode as “Cycle” mode, it can report GPIO & ADC data in every LoRaWAN uplinks periodically.
mac set_tx_mode cycle ⏎ Set LoRaWAN Uplinks behavior is also cycle mode.
mac set_tx_confirm off ⏎ No need to let server return ACK from downlink.
mac set_tx_interval 6000 ⏎ Upload data by uplinks in every 6000ms.
mac join abp/otaa

…
(User can see uplink's payload in debug log)

```
--> Remote Mode Uplink Port(200) Payload(01 ff ffff 07fe)
01 ㅠ IN1 (PC_8) is 0, IN0 (PC_9) is 1.
ff ㅠ RFU (Reserved For Use)
ffff ㅠ ADC0 (PA_0) is around 3.3V
07fe ㅠ ADC1 (PB_0) is around 1.65V
```

Report 1 GPIO and 1 ADC upon to Server (Trigger Mode by an certain external GPIO pin)

(setting)
rm set_gpio in 0 NC ㅠ Not to use GPIO IN0
rm set_gpio in 1 C 8
rm set_gpio_switch on ㅠ Enable GPIO*1 Report Mode
rm set_adc 0 off ㅠ Not to use ADC Channel 0 (Fixed at PA_0)
rm set_adc 1 on
rm set_adc_switch on ㅠ Enable ADC*1 Report Mode
rm set_port_uplink 150 ㅠ To use LoRaWAN port 150 as uplink port.

(mode & start)
rm set_trigger C 7 rising/falling/both ㅠ Set GPIO PC_7 as an external trigger pin, that can detonate GPIO & ADC reading and upload their data up to server.
rm set_mode trigger ㅠ Set Report Mode as "Trigger" mode, it can report GPIO & ADC data when detecting Rasing/Falling/Both trigger type.
mac set_tx_mode no_cycle ㅠ Set LoRaWAN Uplinks behavior is need to be set at no-cycle mode.
mac join abp/otaa

(Let PC_7 connect with high level signal like 3.3v, to generate a rising signal)
ㅠ user can see uplinks payload in debug log

```
--> Remote Mode Uplink Port(150) Payload(1f ff ffff 07fe)
…
1f ㅠ IN1 (PC_8) is 1, IN0 (PC_9) is not using, so it shows 'f'.
```

Remote Mode Payload

```
/*!
 * Remote Mode Payload Definition
 *
 (Report/Uplink)
 +----------+----------+---------------+-----+--------+--------+
 |  Header  | Data Len | GPIO1 | GPIO0 | N/A |  ADC0  |  ADC1  |
 +----------+----------+---------------+-----+--------+--------+
 | 1st Byte | 2nd Byte |   3rd Byte    | 4th | 5-6th  | 7-8th  |
 +----------+----------+---------------+-----+--------+--------+
 |   0xAC   |  8-bit   | 4-bit | 4-bit | RFU | 12-bit | 12-bit |
 +----------+----------+---------------+-----+--------+--------+
 *
 */
```

1st Byte (Header): Fixed as 0xAC
2nd Byte (Payload Length): 1(GPIO) + 1(RFU) + 4(ADC) = 0x06
3rd Byte (GPIO):
- ㅠ higher nibble: GPIO1 (IN1)
- ㅠ lower nibble: GPIO0 (IN0)

4th Byte (RFU, Reserved For Use)
5th , 6th Bytes (ADC0): 12-bit ADC channel 0 (PA_0) converted raw data.

7th , 8th Bytes (ADC1): 12-bit ADC channel 1 (PB_0) converted raw data.

4.3.2 Downlink from Server & Control GPIO Pins

Control Mode: to control two GPIO output value

(setting)
rm set_gpio out 0 C 4 ⇐ Set GPIO PC_4 as OUT0
rm set_gpio out 1 C 5 ⇐ Set GPIO PC_5 as OUT1
rm set_gpio_switch on
rm set_mode cycle
mac set_tx_mode cycle
mac set_tx_confirm off
rm set_port_downlink 201

(server's downlink payload 0xAC, 0x01, 0x10)

a. 0xAC: Fixed Header
b. 0x01: Payload Length, 1 Byte
c. 0x10: Set 1 (High) to OUT1 (PC_5), Set 0 (Low) to OUT0 (PC_4)