S76S/S78S

Commands Set Reference

Document Name	S76S/S78S Commands Set Reference
Version	v1.6.5
Doc No	
Date	June 13, 2018

Document History

Date	Revised Contents	Revised by	Version
Nov 11 ,2016	Initial version supports EU868, US915 and US915 bands.	Leo Tseng	A
Nov 25, 2016	CN470-510 band added and Typo errors fixed.	Leo Tseng	B
Dec 22, 2016	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	C
Dec 23, 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
Jan, 17, 2017	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, 23, 2017	1 new commands (v1.2.1) sip get_uuid	Leo Tseng	F
Mar, 1, 2017	7 new commands (v1.3.1) rf set_fdev rf get_fdev rf set_cad rf get_cad rf cad mac set_lbt mac get_lbt Change CN470 frequency table	Leo Tseng	G

Apr 24, 2017	6 new commands (v1.4.2)	Leo Tseng	H
	mac set_uplink_dwell		
	mac get_uplink_dwell		
	mac set_downlink_dwell		
	mac get_downlink_dwell		
	mac set_max_erip		
	mac get_max_erip		
	Support AS923 Band & LoRaWAN^{TM} v1.0.2		
June, 9, 2017	5 new commands (v1.4.5)	Leo Tseng	I
	mac set_ch_count		
	mac get_ch_count		
	sip set_storage		
	sip get_storage		
	mac set_keys		
Sep, 11, 2017	6 new commands (v1.5.5)	Leo Tseng	J
	sip set_iap		
	sip set_iap_mode		
	mac set_tx_interval		
	mac get_tx_interval		
	mac set_rx1_freq		
	mac get_rx1_freq		
Oct, 11, 2017	17 new commands (v1.6.0)	Leo Tseng	K
	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		
Mar, 2, 2018		1. Correct typo errors & do some max/min values modification. 2. Remove "IAP" related commands	Leo Tseng	v1.6.3
June, 13, 2018		1. 1 new command added (v1.6.5) mac set_power_index 2. Change "sip set_baudrate" <password> as correct one, to "24399520" from "12345678". 3. Remove CN\ZTE\CLAA related commands 4. Remove rf & rm commands	Leo Tseng	v1.6.5

Index

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.
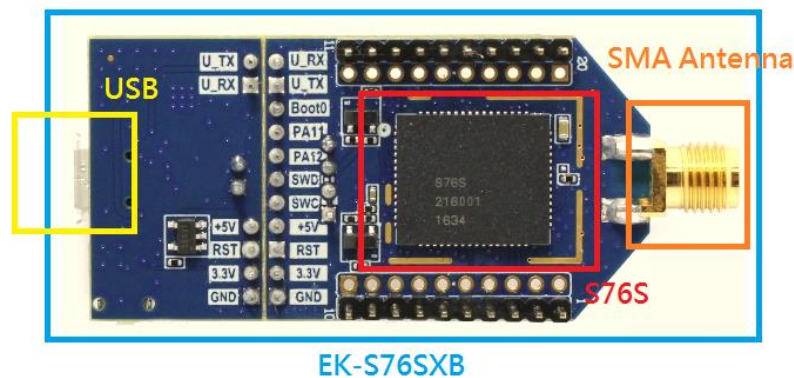
Figure 1.1 (Take S76S as example)

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.
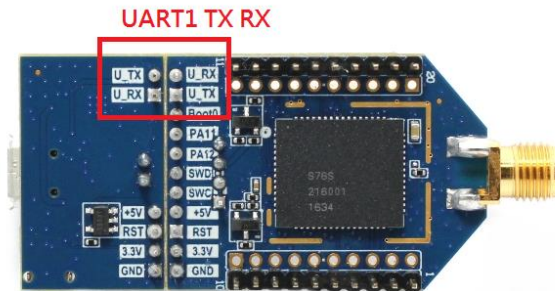
Figure 2.1
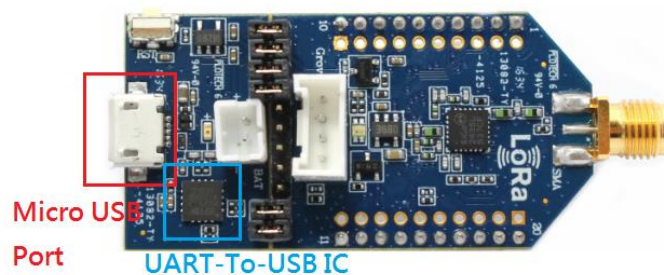
Figure2.2

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).
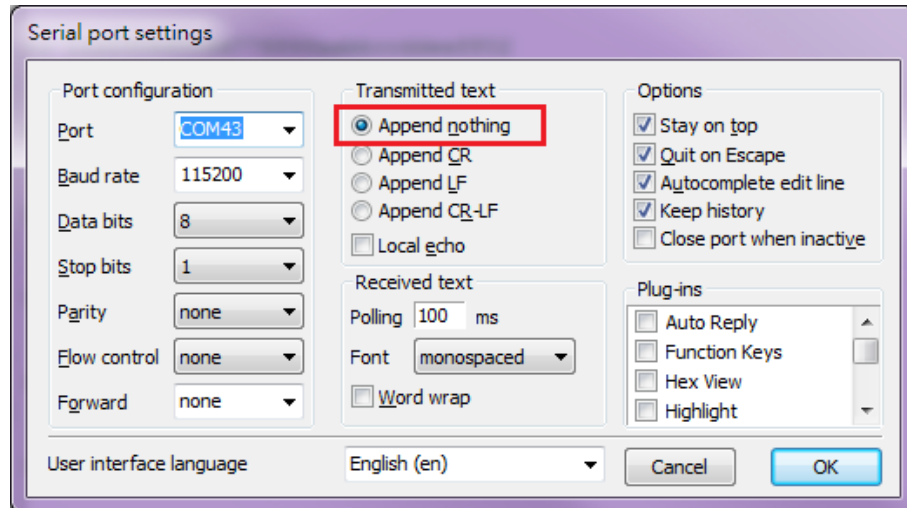
Figure 2.3

3. Commands Set Reference

S76S/S78S commands set can be categorized into 2 types: SIP command and MAC (LoRaWAN™) 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.

The command set interface is readable ASCII string. S76S/S78S starts to accept command if the string starts from "sip" or "mac", 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

3.1.1 sip 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

3.1.2 sip get_ver

Response: A string representing firmware version.

Purpose: Get current firmware version.

Example:

sip get_ver

>> v1.0.8

3.1.3 sip 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

3.1.4 sip get_hw_model

Response: A string representing hardware model.

Purpose: Get hardware model name.

Example:

sip get_hw_model

>> S76S

3.1.5 sip 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

3.1.6 sip 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

3.1.7 sip sleep <Time> <Interruptible>

<Time> A decimal string representing S76S/S78S sleep time in seconds, it can be assigned from **10** to **604800**, and the input value must be the multiple of **10**.

<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:

(It can be triggered by any UART input.)

sip sleep 100 uart_on

>> *sleep 100 sec uart_on (← See this message, it means it starts to sleep.)*

adsfajkl (← Any UART input can wake it up.)

>> *Ok*

(Wake up after 20 seconds)

sip sleep 20 uart_off

>> *sleep 20 sec uart_off*

>> *Ok*

(Not allow to use the value which is not the multiple of 10.)

sip sleep 22 uart_off

>> *Invalid*

3.1.8 sip 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 24399520

>> *Ok*

3.1.9   (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*

3.1.10 (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

3.1.11 (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

3.1.12 (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

3.1.13 (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

3.1.14 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:

3.1.15 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:

Product Name

S76S/S78S Commands Set Reference

Version   v1.6.5
Doc No
Date   June 13, 2018
Page   15 of 48

Termite 3.3 (by CompuPhase) — □ ✕

COM3 115200 bps, 8N1, no handshake   [Settings] [Clear] [About] [Close]

```
sip reset

    ___   ____ _____    ___
   /   | / ___// ___/(_)  __ \
  / /| | \__ \\__ \ / / / / /  Tech Co., LTD
 / ___ |___/ /__/ // / / /     LoRaWAN v1.0.2 Ready
/_/  |_/____//____//_/ /_/        (Class A & C)

>> S76S - v1.4.4 - EU868 - May 22 2017 - 13:08:43

sip get_storage

>> Please copy all the characters below
```

Copy these characters into clipboard

```
APxmFplbhJt3ht+52LM4TzNDJ40FpGSUKkSXfR3MVr+kEnO2XM4YV6d2Xu4LSht
iTB0tUpgdMogMDRGDKaIuqec/uVwZC5u8rLWkEA+jdN/yXuBurHfJHOX4SBX250
s3sdMptwkKhdkVKmQcqE9N/ZgHx5kpFtHyosJXHhnbmLPlgtvIsRmkyLPqnjapd
+YsjLZHZPSDNIm3lrEsabHWvIy2R2T0gzSJt5axLGmx1ryMtkdk9IM0ibeWsSxp
sda8jLZHZPSDNIm3lrEsabHWvIy2R2T0gzSJt5axLGmx1ryMtkdk9IM0ibeWsSx
psda8jLZHZPSDNIm3lrEsabHWvIy2R2T0gzSJt5axLGmx1ryMtkdk9IM0ibeWsS
xpsda8jLZHZPSDNIm3lrEsabHWvIy2R2T0gzSJt5axLGmx1ryMtkdk9IM0ibeWs
Sxpsda8jLZHZPSDNIm3lrEsabHWvIy2R2T0gzSJt5axLGmx1ryMtkdk9IM0ibeW
sSxpsda8p8PIilKvi1BRxIOWS2PzNYy2R2T0gzSJt5axLGmx1rxyaNnvOQ2fI0t
Mysag/lCxEmWyaO/6Jihpl6n033nDEbJ6vMjvmodVB/d2TpFvPakoI4Hv/Msqqa
//xhy1VLrc+IRl7PsHLmGql2PtjahV6GT7ISQcbW6//02uzXIXgNITxMapoP94W
Yo8EqRWJiZ/3J04H/zqDVTVsIeG67LTBNxjnxtvZABeVO0rD9iHWlNQkjXTF5Mb
29NDS5VdnO/hwSfWCRYupyXsQyn+0fdx5ToFjKbEQ0+Jlr4EJrNDtbDyhHS62Z8
alZiKirXuEBV0fiC+nXvCuwMHU0Sej14rg4y2R2T0gzSJt5axLGmx1rwL6YU43I
AhbNkzKMPVyZ1pcKsb5dd
```

[↵]

ACSIP

Product Name	**S76S/S78S Commands Set Reference**
Version	v1.6.5
Doc No	
Date	June 13, 2018
Page	16 of 48

3.2 MAC commands

3.2.1 mac 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

3.2.2 mac 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.

3.2.3 mac save

Response: **Ok**

Purpose: Save LoRaWAN^TM configuration parameters to flash.

Example:

mac save

>> Ok

3.2.4 mac get_join_status

Response: a string representing whether module is joined successfully. Returned string can be: **joined**, **unjoined**.

Purpose: Get join status of LoRaWAN^TM.

Example:

mac get_join_status

>> joined

3.2.5 mac 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

3.2.6 mac set_deveui <DevEUI>

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

Response: **Ok**, if <DevEUI> string is valid

> **Invalid**, if <DevEUI> string is not valid.

Purpose: Set Device EUI used for LoRaWAN™.

Example:

mac set_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, and it would NOT be changed to default value by "sip factory_reset" command.

3.2.7   mac 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

Note: This assigned AppEUI would be stored into EEPROM immediately after it is changed. No need to use "mac save" command to store it, and it would NOT be changed to default value by "sip factory_reset" command.

3.2.8   mac 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 8089b0e2879ff3307a8c95bfbd7416ff

>> Ok

Note: This assigned AppKey would be stored into EEPROM immediately after it is changed. No need to use "mac save" command to store it, and it would NOT be changed to default value by "sip factory_reset" command.

3.2.9   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^TM.

Example:

mac set_devaddr 12345678

>> Ok

Note: This assigned DevAddr would be stored into EEPROM immediately after it is changed. No need to use "mac save" command to store it, and it would NOT be changed to default value by "sip factory_reset" command.

3.2.10 mac set_nwkskey <NwkSessionKey>

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

Response: **Ok**, if <NwkSessionKey> string is valid

Invalid, if <NwkSessionKey> string is not valid.

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

mac set_nwkskey 703af95b0ea9194771f82f3794ccf18f

>> Ok

Note: This assigned NwkSKey would be stored into EEPROM immediately after it is changed. No need to use "mac save" command to store it, and it would NOT be changed to default value by "sip factory_reset" command.

3.2.11 mac set_appskey <AppSessionKey>

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

Response: **Ok**, if <AppSessionKey> string is valid

Invalid, if <AppSessionKey> string is not valid.

Purpose: Set Application Session Key used for LoRaWAN^TM.

Example:

mac set_appskey 262f196d3510eeb304c2a8896545a36d

>> Ok

Note: This assigned AppSKey would be stored into EEPROM immediately after it is changed. No need to use "mac save" command to store it, and it would NOT be changed to default value by "sip factory_reset" command.

3.2.12 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

3.2.13 mac set_dr <Datarate>

<DataRate>: a decimal string representing data rate used for LoRaWAN™, it can be from **0** to **6.** (US915 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

3.2.14 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

3.2.15 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

3.2.16 mac set_rxdelay1 <Delay>

<Delay>: a decimal string representing delay interval in milliseconds used for receive window 1, it can be from **950** to **64000**. 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

3.2.17 mac set_rx2 <DataRate> <Frequency>

<DataRate>: a decimal string representing data rate of second receive window, it can be **0** to **7** (EU868 or AS923), **0** to **5** (CN470), **0** to **15** (US915).

<Frequency>: a decimal string representing operation frequency of specified channel in Hz, it can be from **902000000** to **932000000** (US915); from **470000000** to **510000000** (CN470); from **433000000** to **932000000** (Other regions).

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

3.2.18 mac set_sync <SyncWord>

<SyncWord>: a hexadecimal string representing sync word, it can be from **00** 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

3.2.19 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**; US915 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** (US915); from **470000000** to **510000000** (CN470); from **433000000** to **932000000** (Other regions).

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

3.2.20 mac set_ch_dr_range <ChannelId> <MinDR> <MaxDR>

<ChannelId>: a decimal string representing channel number, its value range depends on different regional band. It can be **0** to **15** (EU868 or AS923), **0** to **95** (CN470) or **0** to **71** (US915).

<MinDR>: a string representing minimum data rate, it can be from **0** when uplink dwell is on, **2** when downlink dwell is off.

<MaxDR>: a string representing maximum data rate, it would be **6** for EU868 or AS923, **5** for CN470 and **4** for US915.

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

3.2.21 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**; US915 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

3.2.22 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.

Note: Used for CE certification.

Example:

mac set_dc_ctl on

>> Ok

3.2.23 mac set_dc_band <BandID> <DutyCycle>

<BandID>: a decimal string representing band number, it can be from **0** to **15** for EU868 or AS923, **0**
 for US915 or CN470.

<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

3.2.24 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

3.2.25   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

3.2.26   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

3.2.27   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

3.2.28   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

3.2.29 mac get_deveui

Response: a hexadecimal string representing Device EUI used for LoRaWAN[TM].

Purpose: Return Device EUI used for LoRaWAN[TM].

Example:

mac get_deveui

>> 000b78ffff000000

3.2.30 mac get_appeui

Response: a hexadecimal string representing Application EUI used for LoRaWAN[TM].

Purpose: Return Application EUI used for LoRaWAN[TM].

Example:

mac get_appeui

>> 0000000000000000

3.2.31 mac get_nwkskey

Response: a hexadecimal string representing Network Session Key used for LoRaWAN[TM].

Purpose: Return Network Session Key used for LoRaWAN[TM].

Note: The middle part of 10 bytes is masked as star characters for safety reason.

Example:

mac get_nwkskey

*>> 703af9*******************ccf18f*

3.2.32 mac get_appskey

Response: a hexadecimal string representing Application Session Key used for LoRaWAN[TM].

Purpose: Return Application Session Key used for LoRaWAN[TM].

Note: The middle part of 10 bytes is masked as star characters for safety reason.

Example:

mac get_appskey

*>> 262f19******************45a36d*

3.2.33 mac get_appkey

Response: a hexadecimal string representing Application Key used for LoRaWAN[TM].

Purpose: Return Application Key used for LoRaWAN[TM].

Note: The middle part of 10 bytes is masked as star characters for safety reason.

Example:

mac get_appkey

*>> 8089b0******************7416ff*

3.2.34 mac get_dr

Response: a decimal string representing data rate used for LoRaWAN^{TM}, it can be from 0 to 6.

Purpose: Return data rate used for LoRaWAN^{TM}.

Example:

mac get_dr

>> 0

3.2.35 mac get_band

Response: a string representing current frequency plan, it can be **470, 868, 915, and 923**.

Purpose: Get current frequency list name.

Example:

mac get_band

>> 915

3.2.36 mac get_power

Response: a decimal string representing transmitting power in dBm.

Purpose: Return transmitting power.

Example:

mac get_power

>> 14

3.2.37 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

3.2.38 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

3.2.39 mac get_rxdelay

Response: <rxdelay1> <rxdelay2>

<rxdelay1> - delay interval in **milliseconds** used for receive window 1 (RX1), it can be from **950** to **64000**.

<rxdelay2> - delay interval in milliseconds used for receive window 2 (RX2), this value would be more 1000 than rxdelay1.

Purpose: Get delay interval of receive window 1 and receive window 2.

Example:

mac get_rxdelay

>> 1000 2000

3.2.40 mac get_rx2

Response: <DR> <freq>

   <DR> - data rate of second receive window.

   <freq> - operation frequency of second receive window in Hz.

Purpose: Get data rate and operation frequency used for second receive window.

Example:

mac get_rx2

>> 0 868000000

3.2.41 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

3.2.42 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; US915 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

3.2.43 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; US915 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

3.2.44 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

3.2.45 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

3.2.46 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 get_join_ch

>> 0 1 2

3.2.47 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

3.2.48 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

3.2.49 mac get_class

Response: class type of LoRaWAN^{TM}, can be **A** or **C**. Default: **A**

Purpose: Get class type of LoRaWAN^{TM}.

Example:

mac get_class

>> A

3.2.50 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

3.2.51 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

3.2.52 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

3.2.53 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

3.2.54 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

3.2.55 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

3.2.56 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)

…

3.2.57 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

>> off

3.2.58 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.**

Purpose: 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.

Note: Only useful for AS923.

Example:

mac set_uplink_dwell on

>> Ok

3.2.59 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

3.2.60 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.**

Purpose: 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.

Note: Only useful for AS923.

Example:

mac set_downlink_dwell on

>> Ok

3.2.61 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

3.2.62 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.

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

Note: Only useful for AS923/EU868/CN470

Example:

mac set_max_eirp 4

>> Ok

3.2.63 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

3.2.64 mac set_ch_count <ChannelsCount> <BW>

<ChannelsCount>: a decimal string representing channel count, it can only be **0~8, 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. US915 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

3.2.65 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

3.2.66 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.

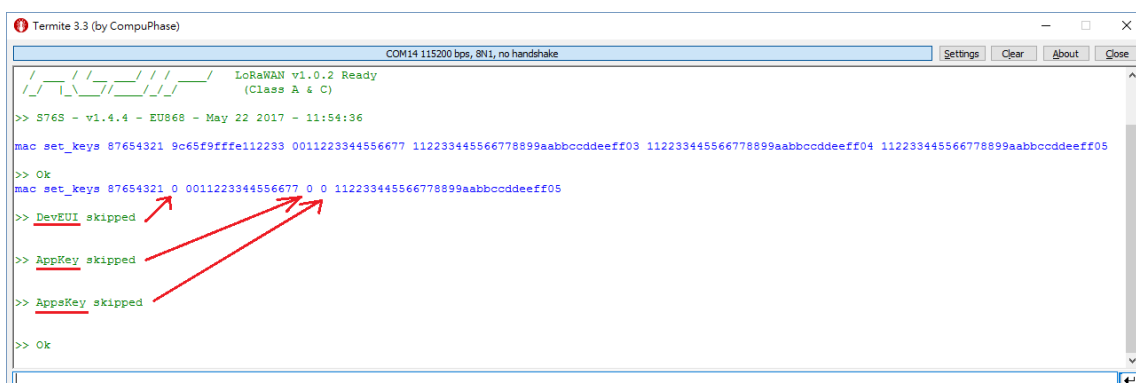**Invalid**, if one of input strings is not valid.

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)
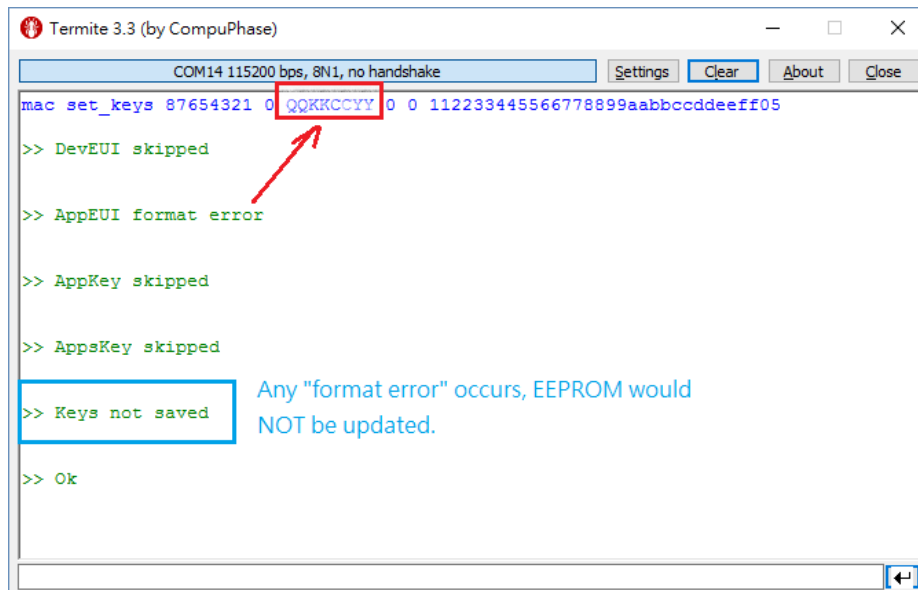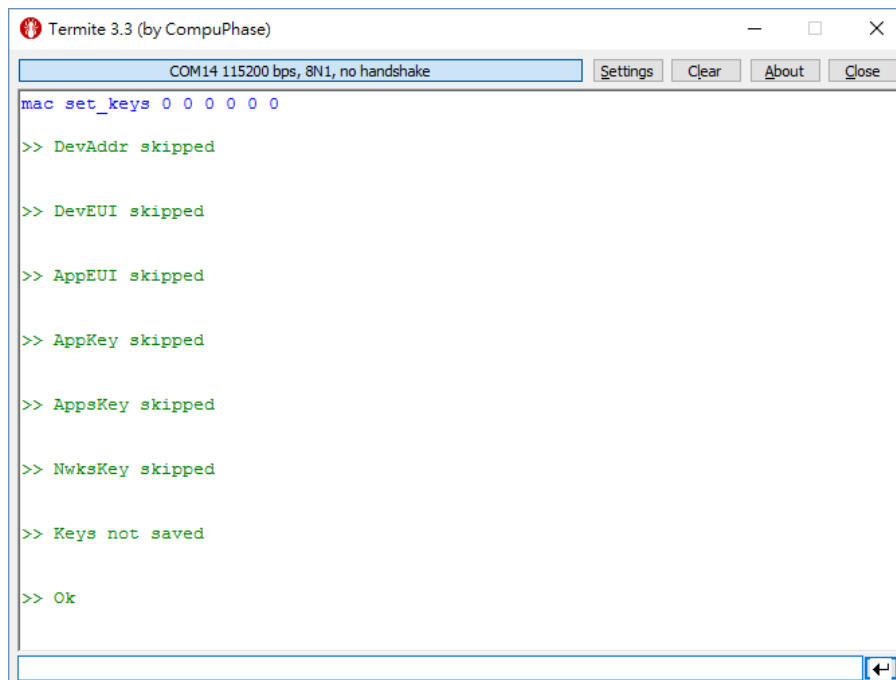

(Incorrect examples)

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

```
Termite 3.3 (by CompuPhase)                    —  □  ×
    COM14 115200 bps, 8N1, no handshake    Settings | Clear | About | Close
mac set_keys 87654321 0 QQKKCCYY 0 0 112233445566778899aabbccddeeff05

>> DevEUI skipped

>> AppEUI format error

>> AppKey skipped

>> AppsKey skipped

>> Keys not saved          Any "format error" occurs, EEPROM would
                           NOT be updated.
>> Ok
```
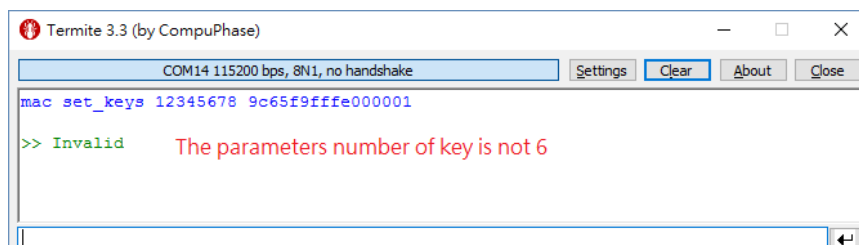


```
Termite 3.3 (by CompuPhase)                    —  □  ×
    COM14 115200 bps, 8N1, no handshake    Settings | Clear | About | Close
mac set_keys 0 0 0 0 0 0

>> DevAddr skipped

>> DevEUI skipped

>> AppEUI skipped

>> AppKey skipped

>> AppsKey skipped

>> NwksKey skipped

>> Keys not saved

>> Ok
```



```
Termite 3.3 (by CompuPhase)                    —  □  ×
    COM14 115200 bps, 8N1, no handshake    Settings | Clear | About | Close
mac set_keys 12345678 9c65f9fffe000001

>> Invalid      The parameters number of key is not 6
```

3.2.67 mac set_tx_interval <Interval>

<Interval>: A decimal string representing TX (LoRaWAN^{TM} Uplink) interval (ms), it can be between **5000** to **86400000.**

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^{TM} Uplinks) when it's under tx cycle mode and the duty cycle control is turned off.

Example:

mac set_tx_mode cycle

>> Ok
mac set_dc_ctl off

>> Ok
mac set_tx_interval 6000

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

3.2.68 mac get_tx_interval

Response: A decimal string representing the current TX intval setting value, its range would be from **5000** to **86400000**.

Purpose: See "mac set_tx_interval" command.

Example:

mac get_tx_interval

>> 6000

3.2.69 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^{TM} 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);

Note: This command is not available for US915 regions.

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

Product Name		**S76S/S78S Commands Set Reference**
	Version	v1.6.5
	Doc No	
	Date	June 13, 2018
	Page	41 of 48

Acsip

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

3.2.70 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

3.2.71 mac set_auto_join <Switch> <Join_Type> <Join_Count>

<Switch>: A string representing whether auto join mode is **on** or **off**.

Response: **Ok**, if < Switch > 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

3.2.72 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

3.2.73 mac set_power_index <Power_Index>

<Power_Index>: a decimal string representing transmitting power index (TXPower), it can be **0 to 7** (non-915 band); **0 to 10** (915 band).

Response: **Ok**, if <Power> string is valid

        Invalid, if <Power> string is not valid.

Purpose: Set transmitting power index value. (See the diagram below)

TXPower	Configuration (EIRP)
0	MaxEIRP
1	MaxEIRP – 2dB
2	MaxEIRP – 4dB
3	MaxEIRP – 6dB
4	MaxEIRP – 8dB
5	MaxEIRP – 10dB
6	MaxEIRP – 12dB
7	MaxEIRP – 14dB
8..15	RFU

Table 5: TX power table

LoRaWAN v1.0.2 EU868

Example:

mac set_power_index 3

>> Ok

3.2.74 mac get_power_index

Response: a decimal string representing transmitting power index value.

Purpose: Return transmitting power index value.

Example:

mac get_power_index

>> 3

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^{TM}

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^{TM} 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^{TM} 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