

# S76G/S78G

## Commands Set Reference



<b>Document Name</b>	<b>S76G/S78G Commands Set Reference</b>
<b>Version</b>	<b>v1.6.6-g11</b>
<b>Date</b>	<b>Sep. 17, 2019</b>



# 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) 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™ v1.0.2	Leo Tseng	H
Jun. 9, 2017	5 new commands (v1.4.5) mac set_ch_count mac get_ch_count sip set_storage sip get_storage mac set_keys	Leo Tseng	I
Sep. 11, 2017	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	Leo Tseng	J
Oct. 11, 2017	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	Leo Tseng	K

Dec. 18, 2017	<p>rm get_trigger</p> <p>11 new commands (v1.6.2-g6)</p> <p>gps set_level_shift</p> <p>gps set_nmea</p> <p>gps set_port_uplink</p> <p>gps set_format_uplink</p> <p>gps set_positioning_cycle</p> <p>gps set_mode</p> <p>gps get_mode</p> <p>gps get_data</p> <p>gps sleep</p> <p>gps get_ttf</p>	Leo Tseng	V1.6.2-g6
Jan. 18, 2017	<ol style="list-style-type: none"> <li>1. Adds more details of GPS format.</li> <li>2. GPS examples modification</li> <li>3. gps set_satellite_system command adds</li> </ol>	Leo Tseng	V1.6.2-g7
Mar. 28, 2018	<ol style="list-style-type: none"> <li>1. Correct typo errors &amp; do some max/min values modification.</li> <li>2. A new command: gps set_start</li> <li>3. "IAP" related commands are removed</li> </ol>	Leo Tseng	V1.6.3-g7
May 15, 2018	<ol style="list-style-type: none"> <li>1. 7 new commands added (v1.6.5) mac set_claa_join_ch mac get_claa_join_ch mac set_realtime_store mac get_realtime_store mac set_csma mac get_csma mac set_endswitchclassacmode</li> <li>2. Change "sip set_baudrate" &lt;password&gt; as correct one, to "24399520" from "12345678".</li> </ol>	Leo Tseng	V1.6.5-g7
Jul. 2, 2018	<ol style="list-style-type: none"> <li>1. 3 new commands added (v1.6.6) sip set_batt_resistor sip get_batt_resistor sip get_batt_volt</li> <li>2. GPS raw data format changed</li> </ol>	Leo Tseng	V1.6.5-g9
Dec. 20, 2018	<ol style="list-style-type: none"> <li>1. One new command added</li> </ol>	Leo Tseng	V1.6.6-g10

	<p>gps set_low_power</p> <ol style="list-style-type: none"> <li>2. sip sleep command is modified</li> <li>3. gps set_satellite_system is modified</li> </ol>		
<p>Aug. 21, 2019</p>	<ol style="list-style-type: none"> <li>1. Delete some unused commands.</li> </ol>	<p>JC</p>	<p>V1.6.6-g10</p>
<p>Sep. 17, 2019</p>	<ol style="list-style-type: none"> <li>1. gps set_satellite_system is modified.</li> <li>2. gps get_mode is modified.</li> </ol>	<p>JC</p>	<p>V1.6.6-g11</p>



# 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 reset](#)

#### [3.1.4 sip get hw model](#)

#### [3.1.5 sip set echo](#)

#### [3.1.6 sip set log](#)

#### [3.1.7 sip sleep](#)

#### [3.1.8 sip baudrate](#)

#### [3.1.9 sip get hw model ver](#)

#### [3.1.10 sip set gpio mode](#)

#### [3.1.11 sip set gpio](#)

#### [3.1.12 sip get gpio](#)

#### [3.1.13 sip get uuid](#)

#### [3.1.14 sip set storage](#)

#### [3.1.15 sip get storage](#)

#### [3.1.16 sip set batt resistor](#)

#### [3.1.17 sip get batt resistor](#)

#### [3.1.18 sip get batt volt](#)

### 3.2 MAC commands

#### [3.2.1 mac tx](#)

#### [3.2.2 mac join](#)

#### [3.2.3 mac save](#)

#### [3.2.4 mac get join status](#)

#### [3.2.5 mac set linkchk](#)

#### [3.2.6 mac set deveui](#)

- [3.2.7 mac set appeui](#)
- [3.2.8 mac set appkey](#)
- [3.2.9 mac set devaddr](#)
- [3.2.10 mac set nwkskey](#)
- [3.2.11 mac set appskey](#)
- [3.2.12 mac set power](#)
- [3.2.13 mac set dr](#)
- [3.2.14 mac set adr](#)
- [3.2.15 mac set txretry](#)
- [3.2.16 mac set rxdelay1](#)
- [3.2.17 mac set rx2](#)
- [3.2.18 mac set sync](#)
- [3.2.19 mac set ch freq](#)
- [3.2.20 mac set ch dr range](#)
- [3.2.21 mac set ch status](#)
- [3.2.22 mac set dc ctl](#)
- [3.2.23 mac set dc band](#)
- [3.2.24 mac set join ch](#)
- [3.2.25 mac set upcnt](#)
- [3.2.26 mac set downcnt](#)
- [3.2.27 mac set class](#)
- [3.2.28 mac get devaddr](#)
- [3.2.29 mac get deveui](#)
- [3.2.30 mac get appeui](#)
- [3.2.31 mac get nwkskey](#)
- [3.2.32 mac get appskey](#)
- [3.2.33 mac get appkey](#)
- [3.2.34 mac get dr](#)
- [3.2.35 mac get band](#)
- [3.2.36 mac get power](#)
- [3.2.37 mac get adr](#)
- [3.2.38 mac get txretry](#)
- [3.2.39 mac get rxdelay](#)
- [3.2.40 mac get rx2](#)
- [3.2.41 mac get sync](#)
- [3.2.42 mac get ch para](#)
- [3.2.43 mac get ch status](#)



[3.2.44 mac get dc ctl](#)  
[3.2.45 mac get dc band](#)  
[3.2.46 mac get join ch](#)  
[3.2.47 mac get upcnt](#)  
[3.2.48 mac get downcnt](#)  
[3.2.49 mac get class](#)  
[3.2.50 mac set tx mode](#)  
[3.2.51 mac get tx mode](#)  
[3.2.52 mac set batt](#)  
[3.2.53 mac get batt](#)  
[3.2.54 mac set tx confirm](#)  
[3.2.55 mac get tx confirm](#)  
[3.2.56 mac set lbt](#)  
[3.2.57 mac get lbt](#)  
[3.2.58 mac set uplink dwell](#)  
[3.2.59 mac get uplink dwell](#)  
[3.2.60 mac set downlink dwell](#)  
[3.2.61 mac get downlink dwell](#)  
[3.2.62 mac set max erip](#)  
[3.2.63 mac get max erip](#)  
[3.2.64 mac set ch count](#)  
[3.2.65 mac get ch count](#)  
[3.2.66 mac set keys](#)  
[3.2.67 mac set tx interval](#)  
[3.2.68 mac get tx interval](#)  
[3.2.69 mac set rx1 freq](#)  
[3.2.70 mac get rx1 freq](#)  
[3.2.71 mac set auto join](#)  
[3.2.72 mac get auto join](#)  
[3.2.73 mac set power index](#)  
[3.2.74 mac get power index](#)

### 3.3 GPS commands

[3.5.1 qps set level shift](#)  
[3.5.2 qps set nmea](#)  
[3.5.3 qps set port uplink](#)  
[3.5.4 qps set format uplink](#)



[3.5.5 qps set positioning cycle](#)

[3.5.6 qps set mode](#)

[3.5.7 qps get mode](#)

[3.5.8 qps get data](#)

[3.5.9 qps sleep](#)

[3.5.10 qps get ttff](#)

[3.5.11 qps set satellite system](#)

[3.5.12 qps set start](#)

[3.5.13 qps set low power](#)

#### 4. Example

##### 4.1 LoRaWAN™

[4.1.1 ABP](#)

[4.1.2 OTAA](#)

[4.1.3 Confirmed Uplink and Downlink](#)

##### 4.2 GPS Mode

[4.2.1 GPS Manual Mode](#)

[4.2.2 GPS Auto Mode](#)

[4.2.3 Enter & Leave GPS Sleep](#)

[4.2.4 Get GPS TTFF Value](#)

[4.2.5 GPS Auto Mode & GPS Auto Join](#)

# 1. Introduction

The S76G/S78G 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), S76G supports global 868 MHz or 915 MHz ISM-Bands. (S78G supports 433MHz or 470 MHz by using SX1278 and the identical MCU with S76G) Capable of 2-way communication and reach over 16 km (10 miles) distance in our field test. Besides, S76G/S78G contains a GPS chip – SONY CXD5603GF which is used to receive GPS/GPS+GLONASS signals for positioning.

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<sup>®</sup> applications.

For faster development, AcSiP provides an EKB named EK-S76GXB. 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. Figure 1.1 describes the related components above.

S76G module provides a commands set interface that can use LoRa<sup>®</sup> and LoRaWAN<sup>™</sup> communication through UART interface. And S76G LoRaWAN<sup>™</sup> protocol has been certificated by LoRaWAN<sup>™</sup> alliance.

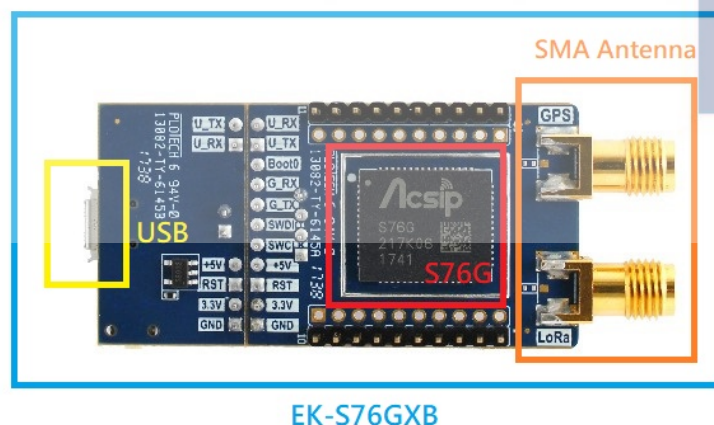


Figure 1.1 (Take S76G as example)

## 2. Configuration

### 2.1 Hardware Configuration

S76G/S78G 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 S76G/S78G.

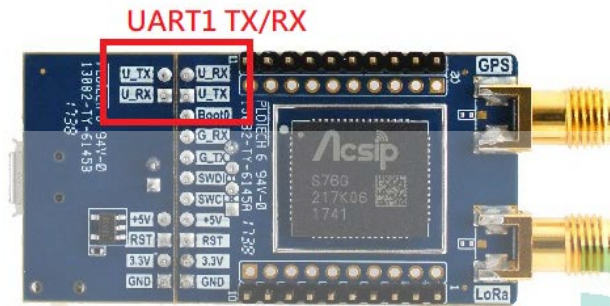


Figure 2.1

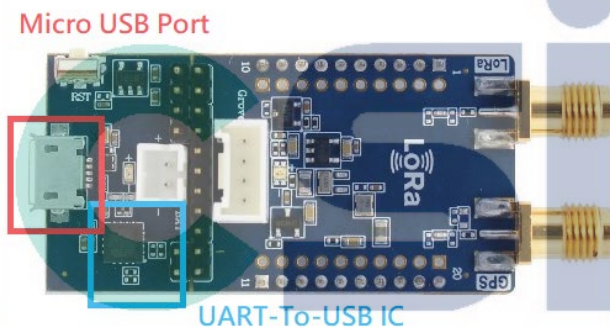


Figure 2.2

### 2.2 Software Configuration

The default baud rate of S76G/S78G 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 S76G/S78G EVB, the 1<sup>st</sup> 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).

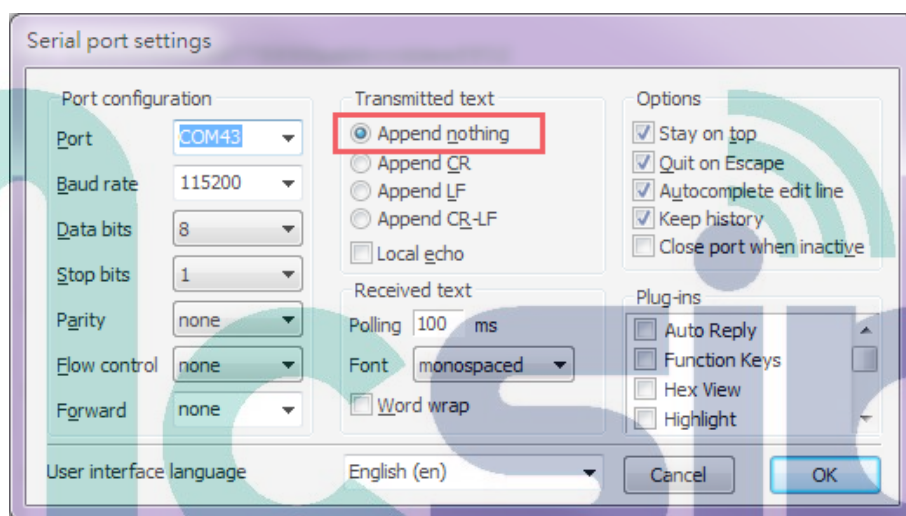


Figure 2.3

## 3. Commands Set Reference

S76G/S78G commands set can be categorized into 4 types: SIP command, 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. GPS (gps mode) commands can control SONY CXD5603GF GPS chip by GPS commands or receive GPS responses.

The command set interface is readable ASCII string. S76G/S78G starts to accept command if the string starts from these “sip”, “mac” and “gps” prefixes, and the response string from S76G/S78G 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 S76G/S78G.

```
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  
>> S76G - 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
>> S76G
```

### 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> <Gps\_sleep\_mode>

<Time> A decimal string representing S76G/S78G 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.

<Gps\_sleep\_mode> It is used to transfer a value of SONY GPS sleep state, it can be assigned as state **gps\_sleep\_0**, **gps\_sleep\_1** or **gps\_sleep\_2**; If <Gps\_sleep\_mode> is not given by caller, the default gps sleep state is set to **gps\_sleep\_1**.

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:

1. (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 ()
```

2. (Wake up only after 20 seconds)

```
sip sleep 20 uart_off
```

```
>> sleep 20 sec uart_off (← Forced to sleep 20s, not able to be waked up by UART input.)
```

```
>> Ok
```

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

```
sip sleep 22 uart_off
```

```
>> Invalid
```

4. (Sip sleep for 20s without UART wake-up, SONY GPS also enters the deepest sleep mode.)

```
sip sleep 20 uart_off gps_sleep_2
```

```
>> sleep 20 sec uart_off (← See this message, it means it starts to sleep, and GPS is using the deepest sleep state, gps_sleep_2)
```

```
>> Ok (← wake up after 20s.)
```

### 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=S76G 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 EEPROM from another S76G/S78G 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 decryption error**, AES execution error occurs.

**Invalid**, if anything is wrong in executing this command.

Example:

```

Termiter 3.3 (by CompuPhase)
COM14 115200 bps, 8N1, no handshake
sip reset

Tech Co., LTD
LoRaWAN v1.0.2 Ready
(Class A & C)

>> S76S - v1.4.4 - EU868 - May 22 2017 - 11:54:36

sip set_storage APxmFplbhJt3ht+52LM4TzNDJ40FpGSUKkSXfR3MvR+kEnO2XM4YV6d2Xu4LSht
iTB0tUpgdMogMDRGDKaIuqec/uVwZC5u8rLWkEA+jdN/yXuBurHfJHOX4SBX250
s3sdMptwkKhdkVKmQcqE9N/ZgHx5kpFtHyosJXHhnbmLP1gtvIsRmkyLPqnjapd
+YsjLZH2PSDNI31rEsabHWvIy2R2T0gzSjt5axLGmx1ryMtkdk9IM0ibeWsSxp
sda8jLZH2PSDNI31rEsabHWvIy2R2T0gzSjt5axLGmx1ryMtkdk9IM0ibeWsSx
psda8jLZH2PSDNI31rEsabHWvIy2R2T0gzSjt5axLGmx1ryMtkdk9IM0ibeWsS
xpsda8jLZH2PSDNI31rEsabHWvIy2R2T0gzSjt5axLGmx1ryMtkdk9IM0ibeWs
sXpsda8p8PIi1Kvi1BRxIOWS2PzNYy2R2T0gzSjt5axLGmx1ryMtkdk9IM0ibeW
Mysag/1CxEmWyaO/6Jihpl6n033nDEbJ6vMjvmodVB/d2TpFvPakoI4Hv/Msqqa
//xhy1VLrc+IR17PsHLMGq12PtjahV6GT7ISQcbW6//02uzXIXgNITxMapoP94W
Yo8EqRWJiZ/3J04H/zqDVTVsIeG67LTBNxjnxvtvZABeVO0rD9iHW1NQkjXTF5Mb
29NDS5VdnO/hwSfWCRyupyXsQyn+0fdx5ToFjKbEQ0+J1r4EJrNDtbDyhHS62Z8
a1ZiKirXuEBV0fiC+nXvCuwMHU0Sej14rg4y2R2T0gzSjt5axLGmx1rwL6YU43I
AhbNkzKMPVyZ1pcKsb5dd

>> Ok

```

### 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 EEPROM (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:



```

Termiter 3.3 (by CompuPhase)
COM3 115200 bps, 8N1, no handshake
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

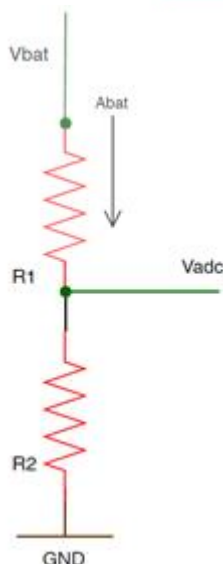
APxmFplbhJt3ht+52LM4TzNDJ40FpGSUKkSXfR3MVr+kEnO2XM4YV6d2Xu4LSht
iTB0tUpgdMogMDRGDKaIuqec/uVwZC5u8rLWkEA+jdN/yXuBurHfJHOX4SBX250
s3sdMptwkKhdKVKmQcqE9N/ZgHx5kpFtHyosJXHhnbmLP1gtvIsRmkyLPqnjapd
+YsjLZH2PSDNIm3lrEsabHWvIy2R2T0gzSJt5axLGmx1ryMtkdk9IM0ibeWsSxp
sda8jLZH2PSDNIm3lrEsabHWvIy2R2T0gzSJt5axLGmx1ryMtkdk9IM0ibeWsSx
psda8jLZH2PSDNIm3lrEsabHWvIy2R2T0gzSJt5axLGmx1ryMtkdk9IM0ibeWsS
xpsda8jLZH2PSDNIm3lrEsabHWvIy2R2T0gzSJt5axLGmx1ryMtkdk9IM0ibeWs
Sxpsda8jLZH2PSDNIm3lrEsabHWvIy2R2T0gzSJt5axLGmx1ryMtkdk9IM0ibeW
sSxpsda8p8PIiIkvi1BRxiOWS2PzNYy2R2T0gzSJt5axLGmx1rxyaNnvOQ2fI0t
Mysag/1CxEmNyaO/6Jihp16n033nDEbJ6vMjvmodVB/d2TpFvPakoI4Hv/Msqqa
//xhy1VLzc+IR17PsHLMGq12PtjahV6GT7ISQcbW6//02uzXIXgNITxMapoP94W
Yo8EqRWJ1Z/3J04H/zqDVTVsIeG67LTBNxjnxvtZABeVO0rD9iHW1NQkxjXTF5Mb
29NDS5VdnO/hwSfWCRYupyXsQyn+0fdx5ToFjKbEQ0+J1r4EJrNDtbDyhHS62Z8
a1ZiKirXuEBV0fiC+nXvCuwMHU0Sej14rg4y2R2T0gzSJt5axLGmx1rwL6YU43I
AhbNkzKMPVY21pcKsb5dd

```

### 3.1.16 sip set\_batt\_resistor <R1> <R2>

<R1> A decimal string representing the resistance value (ohm) of resistor R1 (see the below figure), it must be greater than **0** and less or equal than **2<sup>24</sup>** (16777216, 16M).

<R2> A decimal string representing the resistance value (ohm) of resistor R1 (see the below figure), it must be greater than **0** and less or equal than **2<sup>24</sup>** (16777216, 16M)



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

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

Purpose: User needs to measure the external voltage (e.g. battery voltage) which is higher than 3.3v. The full-charged voltage of some lithium-ion battery might exceed 3.3 volt, this voltage level already exceeds the maximum allowable voltage of S7XX ADC. So user needs to provide an external circuit, voltage divider, to reduce the magnitude of the external voltage (e.g. Vbat). So the reduced voltage (i.e. Vadc) which is equal or low than 3.3v can be measured correctly by S7XX ADC.

Steps for measure Vbat:

- a. To implement a voltage divider circuit. And the recommended resistance values of R1 & R2 are around **100K** & **200K** respectively.
- b. The end-point of Vadc is needed to be connected with PB\_1 of S7XS GPIO.
- c. To measure the resistance value of these two R1 & R2 by a resistance meter.
- d. To let S7X know the measured R1 & R2 values by calling this command “sip set\_batt\_resistor”.

(e.g.)

```
sip set_batt_resistor 98900 199700
```

```
>> Ok
```

### 3.1.17 sip get\_batt\_resistor

Response: Two decimal values representing R1 & R2 values respectively.

Purpose: See “sip set\_batt\_resistor” command.

Example:

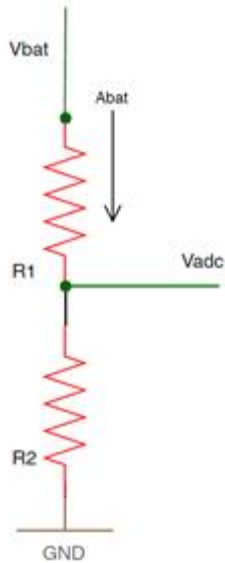
```
sip get_batt_resistor
```

```
>> 98900 199700
```

### 3.1.18 sip get\_batt\_volt

- a. R1 & R2 set

Response: Return ADC voltage (Vadc) & battery voltage (Vbat) if “sip set\_batt\_resistor” had ever set R1 & R2.



Purpose: To tell the battery voltage value (Vbat) & ADC value (Vadc) by given R1 & R2 values. □

Although a voltage divider circuit is used for reducing external voltage (i.e. Vbat), the maximum input voltage value (Vadc) of S7XS ADC can't exceed 3.3V

Example:

```
sip get_batt_volt
```

```
>> adc volt(2807 mv)
```

```
>> battery volt(4197 mv)
```

b. R1 & R2 Not Set.

Response: Return ADC voltage only if R1 & R2 didn't set by calling "sip set\_batt\_resistor".

Purpose: The external voltage can be attached with S7XS GPIO PB\_1 pin directly when the maximum external voltage is not more than 3.3 volt.

Example:

```
sip get_batt_volt
```

```
>> battery volt(2806 mv)
```

## 3.2 MAC commands

### 3.2.1 mac tx <Type> <PortNum> <Data>

<Type>: a string representing type of transmitting message, it 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 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 S76G/S78G in rx <Portnum> <Data> format.

Example:

```
>> mac rx 4 1234abcd
```

### 3.2.2 mac join <Mode>

<Mode>: a string representing join mode of LoRaWAN™, it 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™ 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™.

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™.

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\_appoui <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_appoui 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™.

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™.

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 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™.

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 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, it 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 **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
```

### 3.2.19 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 or 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, it can be from **0** to **65535**.

**0**: means 0%.

**1-65535**: duty cycle is equal to  $1/\langle \text{duty cycle} \rangle$ .

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™.

Purpose: Return Device EUI used for LoRaWAN™.

Example:

```
mac get_deveui  
>> 000b78ffff000000
```

### 3.2.30 mac get\_appmui

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

Purpose: Return Application EUI used for LoRaWAN™.

Example:

```
mac get_appmui  
>> 000000000000000000
```

### 3.2.31 mac get\_nwkskey

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

Purpose: Return Network Session Key used for LoRaWAN™.

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™.

Purpose: Return Application Session Key used for LoRaWAN™.

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™.

Purpose: Return Application Key used for LoRaWAN™.

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™, it can be from 0 to 6.

Purpose: Return data rate used for LoRaWAN™.

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, it can be from **0** to **15**.

Response: **<duty cycle>**, if <BandID> is valid.

<duty cycle> - duty cycle of specified band, it 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 S76G 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, it can be **A** or **C**. Default: **A**

Purpose: Get class type of LoRaWAN™.

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 complied 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 complied 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 S76G 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 S76G 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
>> on
```

### 3.2.56 mac set\_lbt <Switch>

<Switch>: A string representing whether S76G 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 TELEEC request, it needs to detect a channel is using or not before using this channel, so if LBT is on, S76G/S78G can listen before talk (LBT) the channel symbol signal strength before any TX uplink like joining or normal uplinks. If the channel is occupying, S76G/S78G 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 S76G/S78G 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**.

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 channels 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\_appelui" 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:

```

Termite 3.3 (by CompuPhase)
COM14 115200 bps, 8N1, no handshake
sip reset
Tech Co., LTD
LoRaWAN v1.0.2 Ready
(Class A & C)
>> S76S - v1.4.4 - EU868 - May 22 2017 - 11:54:36
mac set_keys 87654321 9c65f9fffe112233 0011223344556677 112233445566778899aabbccddeeff03 112233445566778899aabbccddeeff04 112233445566778899aabbccddeeff05
>> Ok      DevAddr  DeuEUI  AppEUI  AppKey  AppsKey  NwksKey
    
```

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)

```

Termite 3.3 (by CompuPhase)
COM14 115200 bps, 8N1, no handshake
LoRaWAN v1.0.2 Ready
(Class A & C)
>> S76S - v1.4.4 - EU868 - May 22 2017 - 11:54:36
mac set_keys 87654321 9c65f9fffe112233 0011223344556677 112233445566778899aabbccddeeff03 112233445566778899aabbccddeeff04 112233445566778899aabbccddeeff05
>> Ok
mac set_keys 87654321 0 0011223344556677 0 0 112233445566778899aabbccddeeff05
>> DevEUI skipped
>> AppKey skipped
>> AppsKey skipped
>> Ok
    
```

(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
mac set_keys 87654321 0 QQKRCYY 0 0 112233445566778899aabbccddeeff05
>> DevEUI skipped
>> AppEUI format error
>> AppKey skipped
>> AppsKey skipped
>> Keys not saved
>> Ok
    
```

Any "format error" occurs, EEPROM would NOT be updated.

```

Termit 3.3 (by CompuPhase)
COM14 115200 bps, 8N1, no handshake
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

```

```

Termit 3.3 (by CompuPhase)
COM14 115200 bps, 8N1, no handshake
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 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 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 would be uploaded automatically in every 6s)*

### 3.2.68 mac get\_tx\_interval

Response: A decimal string representing the current TX interval 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 mean to 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 1<sup>st</sup> RX1 frequency is beginning from <Rx1\_Freq\_Begin>, the next 2<sup>nd</sup> RX1 would be set at <Rx1\_Freq\_Begin> +

$\langle \text{Rx1\_Step} \rangle * 1$ , the 3<sup>rd</sup> RX1 is  $\langle \text{Rx1\_Freq\_Begin} \rangle + \langle \text{Rx1\_Step} \rangle * 2$ ; So the Nth Rx1 channel frequency would be  $\langle \text{Rx1\_Freq\_Begin} \rangle + \langle \text{Rx1\_Step} \rangle * (\langle \text{Rx1\_Count} \rangle - 1)$ .

- b) After setting TX/RX1 frequency by this command, user can still use “mac set\_ch\_status N on/off”, to control each channel enable/disable individually.

Note: This command is not available for US915 & CLAA regions.

Example:

1. TX(uplinks) & RX1(downlinks) uses different frequency, set RX1 downlink frequency start from 500.3MHz, incremental frequency is 200 KHz, and enables 8 Channels which channel number are 4,5,6,7 and 22,23,24,25 just for demonstration.

```
mac set_rx1_freq 500300000 200000 8
```

```
>> Ok
```

(Set TX frequencies of Channel 22, 23, 24 & 25)

```
mac set_ch_freq 22 474700000
```

```
>> Ok
```

```
mac set_ch_freq 23 474900000
```

```
>> Ok
```

```
mac set_ch_freq 24 475100000
```

```
>> Ok
```

```
mac set_ch_freq 25 475300000
```

```
>> Ok
```

(Enable Channel 22, 23, 24 & 25)

```
mac set_ch_status 22 on
```

```
>> Ok
```

```
mac set_ch_status 23 on
```

```
>> Ok
```

```
mac set_ch_status 24 on
```

```
>> Ok
```

```
mac set_ch_status 25 on
```

```
>> Ok
```

```
(Disable channel 0, 1, 2, 3)
```

```
mac set_ch_status 0 off
```

```
>> Ok
```

```
mac set_ch_status 1 off
```

```
>> Ok
```

```
mac set_ch_status 2 off
```

```
>> Ok
```

```
mac set_ch_status 3 off
```

```
>> Ok
```

(The result of calling the above setting)

	Ch	Freq		Ch	Freq
<b>TX</b>	0	470.3	<b>RX1</b>	0	500.3
	1	470.5		1	500.5
	2	470.7		2	500.7
	3	470.9		3	500.9
	4	471.1		4	501.1
	5	471.3		5	501.3
	6	471.5		6	501.5
	7	471.7		7	501.7
	8	471.9		8	501.9
	9	472.1		9	502.1
	10	472.3		10	502.3
	11	472.5		11	502.5
	12	472.7		12	502.7
	13	472.9		13	502.9
	14	473.1		14	503.1
	15	473.3		15	503.3
	16	473.5		16	503.5
	17	473.7		17	503.7



	18	473.9		18	503.9
	19	474.1		19	504.1
	20	474.3		20	504.3
	21	474.5		21	504.5
	22	474.7		22	504.7
	23	474.9		23	504.9
	24	475.1		24	505.1
	25	475.3		25	505.3
	26	475.5		26	505.5
	27	475.7		27	505.7
	28	475.9		28	505.9
	..	..		..	..

2. (TX & RX1 are using identical frequency setting)

```
mac set_rx1_freq 0
```

```
>> Ok
```

```
mac get_ch_para 0
```

```
>> 470300000 0 5 0 0 (→ The last item, "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
```

(The default 8 channels are enabled and TX/RX1 uses the identical frequency)

	Ch	Freq		Ch	Freq
<b>TX</b>	0	470.3	<b>RX1</b>	0	470.3
	1	470.5		1	470.5
	2	470.7		2	470.7
	3	470.9		3	470.9
	4	471.1		4	471.1
	5	471.3		5	471.3
	6	471.5		6	471.5
	7	471.7		7	471.7

	8	471.9		8	471.9
--	---	-------	--	---	-------

### 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 S76G 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
```

## 3.3 GPS commands

### 3.3.1 gps set\_level\_shift <Switch>

<Switch>: A string representing whether S76G enables the internal level shift IC (converts 3.3v to 1.8v for SONY CXD5603 usage), it can be **on** or **off**.

Response: **Ok**, if <Switch> string is valid.

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

Purpose: When <Switch> is on, it will enable the level shift inside S76G/S78G and initialize UART4 for MCU (STM32L073) sending/receiving commands/responses to/from SONY CXD5603 GPS Chip; When <Switch> is off, it will disable the level shift instead, so UART4/GPS can be controlled by an external UART. <Switch> value would be stored into EEPROM immediately for keeping the next boot-up value without sending "mac save" command.

Example:

```
gps set_level_shift on
```

```
>> Ok
```

*(More demonstration examples, please refer to chapter 4.4)*

### 3.3.2 gps set\_nmea <Sentence>

<Sentence>: A string representing which NMEA sentence S76G is selected, currently on this version, it only can be **rmc**. (**R**ecommended **M**inimum **S**pecific GNSS Data)

Response: **Ok**, if <Sentence> string is valid.

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

Purpose: SONY CXD5603GF GPS chip outputs below sentences of NMEA0183 (ver4.00) compliant sentences. And currently, user can only select RMC as one of NMEA0183 sentences on S76G/S78G. So user might skip this command because the default value is already set at RMC sentence.

CGA: Global Positioning System Fix Data

GLL: Geographic Position – Latitude/Longitude

GNS: GNSS Fix Data

GSA: GNSS DOP and Active Satellites

GSV: GNSS Satellites in View

RMC: Recommended Minimum Specific GNSS Data

VTG: Course Over Ground & Ground Speed

ZDA: Time & Date

<Sentence> value would be stored into EEPROM immediately for keeping the next boot-up value without sending “mac save” command.

Example:

```
gps set_nmea rmc
```

```
>> Ok
```

### 3.3.3 gps set\_port\_uplink <Port>

<Port>: a decimal string representing port number used for GPS uploading data by LoRaWAN TX uplink, its range can be set from **1** to **223**.

Purpose: To set the LoRaWAN uplink port number when uploading GPS UTC, Latitude, Longitude coordinates and other data up to server while S76G is running on TX cycle mode.

Response: **Ok**, if input argument is valid.

**Invalid**, if input argument is not valid or out of range

<Port> value would be stored into EEPROM immediately for keeping the next boot-up value without sending “mac save” command.

Example:

*(To use LoRaWAN port 20 as uplink port)*

```
gps set_port_uplink 20
```

```
>> Ok
```

*(More demonstration examples, please refer to chapter 4.4)*

### 3.3.4 gps set\_format\_uplink <Format>

<Format>: A string representing the format of GPS data used in some particular server that it is required. It can be “**raw**”, “**ipso**”, “**kiwi**” or “**utc\_pos**”.

Purpose: For some particular server, when its capability can show positions on map by given latitude & longitude coordinates. The server would require a specific payload format of LoRaWAN TX uplinks. So S76G/S78G provides these options for user to select one.

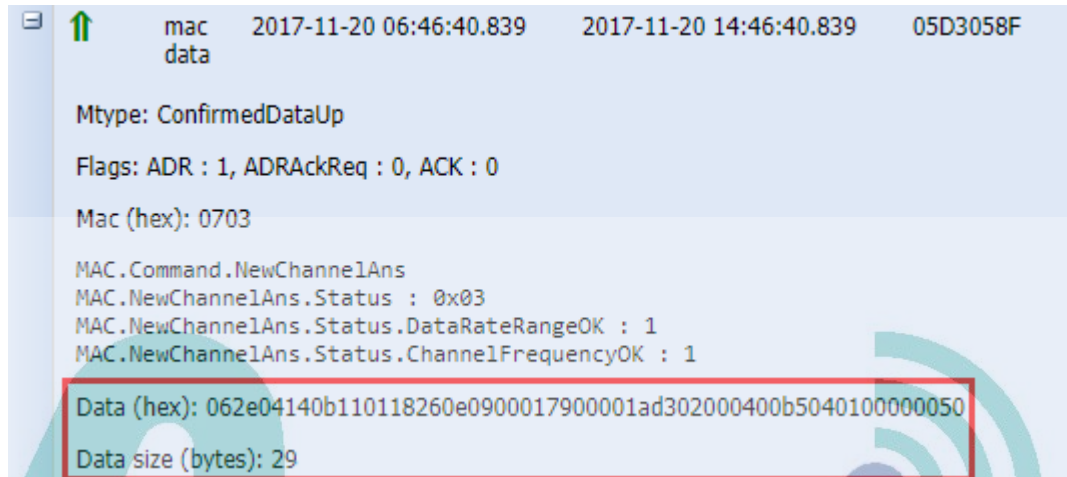
Response: **Ok**, if input argument is valid.

**Invalid**, if input argument is not valid.

<Format> is “**raw**”:

The uploaded data length is 29 bytes.

(e.g. The below figure is a snapshot from Activity\ThingPark LoRaWAN server)



```

mac      2017-11-20 06:46:40.839      2017-11-20 14:46:40.839      05D3058F
data

Mtype: ConfirmedDataUp
Flags: ADR : 1, ADRAckReq : 0, ACK : 0
Mac (hex): 0703

MAC.Command.NewChannelAns
MAC.NewChannelAns.Status : 0x03
MAC.NewChannelAns.Status.DataRateRangeOK : 1
MAC.NewChannelAns.Status.ChannelFrequencyOK : 1

Data (hex): 062e04140b110118260e0900017900001ad302000400b5040100000050
Data size (bytes): 29
  
```

Data Format:

(e.g.) 062e04 140b11 01 18 260e0900 01 79 0000 1ad30200 0400 b504 01000000 50

- a) 062e04: UTC Time 6 hour 46 min 4 sec ==> UTC+8 (@Taiwan) is 2pm 46min 4sec
- b) 140b11: UTC Date 0x11 ==> 2017, 0x0b ==> November, 0x14 ==> 20th; 2017/11/20
- c) 01: North is 0x01, South is 0x00.
- d) 18 260e0900: 0x18 ==> **24**, 0x00090e26 ==> **593446**. So Latitude is **24 59.3446** (GPS RMC Data, not DD or DMS format)
- e) 01: East is 0x01, West is 0x00.
- f) 79 0000 1ad30200: 0x79 ==> **121**, 0x0000 ==> Not Used, 0x0002d31a ==> **185114**. So Longitude is **121 18.5114** (GPS RMC Data, not DD or DMS format)
- g) 0400: 0x0004 ==> Speed over ground is 0.4 knot
- h) b504: 0x04b5 ==> Course over ground is 120.5
- i) 01000000: 0x00000001 ==> Status is 1 (valid, OK), 0 (GPS not ready yet)
- j) 50: 0x50 ==> 80, battery encoded value is 80.
  - i. If “sip set\_batt\_resistor” is not getting called:  
The battery voltage is  $80 * 20 = 1600$  mV.
  - ii. If “sip set\_batt\_resistor” is getting called ever:  
The battery voltage is  $3000 + 80 * 10 = 3800$  mV.

Note:

- 1) Raw to DD

**24 59.3446** (GPS RMC Data)

$59.3446 / 60 = 0.989076$

So GPS DD value is **24.989076**

2) Raw to DMS

**24 59.3446** (GPS RMC Data)

$3446 * ( 60 / 10000 ) = 20.67$

So GPS DMS value is **24°59'20.67"**

<Format> is "**ipso**":

The uploaded data length is 11 bytes.

Reference link: <https://mydevices.com/cayenne/docs/lora/>

#### Device with GPS

<b>Payload (Hex)</b>	01 88 06 76 5f f2 96 0a 00 03 e8	
<b>Data Channel</b>	<b>Type</b>	<b>Value</b>
01 ⇒ 1	88 ⇒ GPS	Latitude: 06765f ⇒ 42.3519
		Longitude: F2960a ⇒ -87.9094
		Altitude: 0003E8 ⇒ 10 meters

<Format> is "**kiwi**":

The uploaded data length is 17 bytes and its format is not allowed to be disclosure here.

But S76G v1.6.2-g6 is already verified and compatible with Kiwi LoRaWAN Server.

Example:

```
gps set_format_uplink raw
```

>> Ok

<Format> is "**utc\_pos**":



Last packets

		UTC Timestamp	Local Timestamp	DevAddr	DevEUI
↑	data	2018-01-18 05:29:40.244	2018-01-18 13:29:40.244	05FD5011	9C65F9F

Mtype: ConfirmedDataUp  
 Flags: ADR : 1, ADRAckReq : 0, ACK : 0  
 Mac (hex): -  
 Data (hex): **cbce48c69ba8ce625b3bc8**  
 Data size (bytes): 11  
 AirTime (s): 0.056576

LRR	RSSI	SNR	ESP	CHAINS timestamp {GPS NTP LOCAL GPS_RADIO}
004A12BD	-77.0	10.25	-77.391785	CHAIN[0]:2018-01-18T06:29:40.244+01:00 {-}

Device [Lat (solv): - Lon (solv): - Lon: - Loc radius: - Loc time: - Alt: - Alt radius: - Acc: - North Velocity: - E:  
 Reporting Status: On time

### LoRaWAN uplinks payload format:

UTC (Date & Time)	Latitude	Longitude
<b>cbce48</b>	<b>c69ba8ce</b>	<b>625b3bc8</b>

- Calculate UTC as Date and Time: (e.g. **cbce48**, 0xcb is 1<sup>st</sup> byte of the payload)
  - Change endian as below:
    - ⇒ cb ce 48 → 0x48cecb
    - ⇒ 0x48cecb equals "0100 1000 1100 1110 1100 1011".
    - ⇒ Divide them as two part: The first 6 bits and others.
    - ⇒ "0100 10" & "00 1100 1110 1100 1011".
  - Find UTC Date out:
    - ⇒ Add 0 in front of "0100 10", it becomes "00010010".
    - ⇒ "00010010" is 0x12 (hex value).
    - ⇒ 0x12 (hex) is 18 (decimal), so the Day of date is **18**.
    - ⇒ Year & Month is not supporting.
  - Find UTC Time out:
    - ⇒ Add 0 in front of "00 1100 1110 1100 1011", it becomes "0000 1100 1110 1100 1011".
    - ⇒ "0000 1100 1110 1100 1011" is 0x0cecd (hex value).
    - ⇒ 0x0cecb (hex) is 52939
    - ⇒ 52939 means "5 29 39"
    - ⇒ **5(hour):29(minute):39(sec)** (UTC Time)
    - ⇒ Taiwan Time (UTC+8) is 13:29:39.
- Calculate Latitude: (e.g. **c69ba8ce**, 0xc6 is 4<sup>th</sup> bytes of the payload, 0xce is 7<sup>th</sup>)
  - Change endian as below:
    - ⇒ ce a8 9b c6 → 0xcea89bc6 (hex) → "1100 1110 1010 1000 1001 1011 1100 0110"

- b) North or South:  
 ⇒ **Bit31** is fixed at **1**, **Bit30** indicates it's showing North or South (**1**: N, **0**: S).
- c) Calculate Latitude as DD (decimal degree) format.  
 ⇒ So change Bit31 & Bit30 as 0 → "0000 1110 1010 1000 1001 1011 1100 0110"  
 ⇒ 0xcea89bc6 becomes 0x0ea89bc6, and its decimal value is 245930950  
 ⇒  $245930950 / 10^7 = 24$  (deg)  
 ⇒  $245930950 \text{ Modulo } 10^7 = 5930950$ ,  $5930950 / 100000 = 59.30950$  (minute)  
 ⇒  $59.30950$  (minute) / 60 = 0.988492 (deg)  
 ⇒ So the latitude is **North, 24.988492** (deg), and it's showing by DD format.
3. Calculate Latitude: (e.g. **625b3bc8**, 0x62 is 8<sup>th</sup> bytes of the payload, 0xc8 is 11<sup>th</sup>)
- a) Change endian as below:  
 ⇒ c8 3b 5b 62 → 0xc83b5b62 (hex) → "1100 1000 0011 1011 0101 1011 0110 0010"
- b) East or West:  
 ⇒ **Bit31** indicates it's showing East or West (**1**: E, **0**: W).
- c) Calculate Latitude as DD (decimal degree) format.  
 ⇒ So change Bit31 & Bit30 as 0 → "0000 1110 1010 1000 1001 1011 1100 0110"  
 ⇒ 0xc83b5b62 becomes 0x483b5b62, and its decimal value is 1211849570  
 ⇒  $1211849570 / 10^7 = 121$  (deg)  
 ⇒  $1211849570 \text{ Modulo } 10^7 = 1849570$ ,  $1849570 / 100000 = 18.49570$  (minute)  
 ⇒  $18.49570$  (minute) / 60 = 0.308261 (deg)  
 ⇒ So the longitude is **East, 121.308261** (deg), and it's showing by DD format.

Example:

```
gps set_format_uplink utc_pos
```

```
>> Ok
```

### 3.3.5 gps set\_positioning\_cycle <Time>

<Time>: A decimal string representing the SONY CXD5603GF GPS positioning cycle time, it can be **1000** to **600000** milliseconds.

Response: **Ok**, if <Interval> string is valid.

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

Purpose: Set a GPS cycle time that is used to tell how soon GPS will update its coordinates.

Example:

```
gps set_positioning_cycle 3000
```

```
>> Ok
```

### 3.3.6 gps set\_mode <Mode>

<Mode>: A string representing S76G/S78G GPS mode is set to either “Auto”, “Manual” or “Idle” mode, the string can be **auto**, **manual** or **idle**.

Response: **Ok**, if <Mode> string is valid.

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

Purpose:

- Auto mode or manual mode, that is used to decide whether it would upload GPS data by LoRaWAN TX uplink automatically or user needs to get GPS data manually and then put it into LoRaWAN TX uplinks manually.
- No matter “auto” or “manual” mode is called, it would start to try to initialize GPS by the given NMEA sentence and positioning cycle time that set before.
- So it is recommended to call “gps set\_nmea” & “gps set\_positioning\_cycle” before calling this command, or it would just return “invalid” response.
- When “Idle” mode is assigned, it lets GPS stop positioning and back to GPS IDLE mode.

Example:

```
gps set_mode auto
```

```
>> Ok
```

```
gps set_iap_mode manual
```

```
>> Ok
```

*(More demonstration examples, please refer to chapter 4.4)*

### 3.3.7 gps get\_mode

Purpose: Return GPS related setting values that set before.

<Mode>: it follows “gps set\_mode” command input format. (off/auto/manual)

<Start\_Type>: it follows “gps set\_start” command input format. (hot/cold/warm)

<Port>: it follows “gps set\_port\_uplink” command input format.

<Time>: it follows “gps set\_positioning\_cycle” command input format.

<Format>: it follows “gps set\_format\_uplink” command input format. (raw/ipso/kiwi/utc\_pos)  
<Satellite\_System>: it follows “gps set\_satellite\_system” command input format. (gps/glonass/gps\_glonass/beidou/gps\_beidou)

Example:

```
gps get_mode
```

```
>> off hot 20 3000 ipso gps
```

### 3.3.8 gps get\_data <Type>

<Type>: a string representing type of how CXD5603GF returns the UTC and coordinates and positioning time, it can be **raw**, **dd** or **dms**.

Purpose: When GPS is running at “Manual” mode by setting “gps set\_mode manual”, user can get GPS related data by this command.

Response:

- a) There are several kinds of responses after entering this command. When GPS is not initialized yet or entered GPS IDLE mode.

Example:

```
gps get_data dd
```

```
>> gps_not_init
```

```
gps set_mode manual
```

```
>> Ok
```

```
gps set_mode idle
```

```
>> Ok
```

```
gps get_data dd
```

```
>> gps_in_idle
```

- b) When GPS starts to position but not positioned well yet, it would just return the spending time when GPS is positioning

Example:

```
gps get_data dd
```

```
>> POSITIONING ( 14.8s )
```

c) When GPS is positioned well, this command can show UTC value, Latitude, Longitude and the last positioning time.

Example:

**To get GPS RAW Data directly from SONY CXD5603GF without any conversion.**

```
gps get_data raw
```

```
>> RAW UTC( 2017/11/23 06:57:16 ) LAT( 2459.3351 N ) LONG( 12118.4978 E ) POSITIONING( 0.8s )
```

**To get GPS DD (Decimal Degree) data**

```
gps get_data dd
```

```
>> DD UTC( 2017/11/23 06:59:06 ) LAT( 24.988825 N ) LONG( 121.308326 E ) POSITIONING( 2.9s )
```

```
gps get_data dms
```

**To get GPS DMS (Degree Minute Second) data**

```
>> DMS UTC( 2017/11/23 06:59:51 ) LAT( 24*59'1990" N ) LONG( 121*18'2992" E ) POSITIONING( 4.6s )
```

d) When GPS is not able to positioning by GPS signal is gone, it would tell GPS is not able to position.

Example:

```
gps get_data dms
```

```
>> gps_not_positioning
```

*(More demonstration examples, please refer to chapter 4.4)*

### 3.3.9 gps sleep <Switch> <Sleep\_Level>

<Switch>: A string representing whether the Sony GPS chip inside S76G/S78G will enter sleep/deep sleep mode or not, the string can be **on** or **off**.

<Sleep\_Level>: To decide which sleep mode S76G/S78G will enter, the value of <Sleep\_Level> can be

0 (Sleep Mode) or 1 (Deep Sleep Mode).

Response: **Ok**, if <Switch> string is valid.

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

Purpose: “gps sleep” only let Sony GPS chip enter sleep mode, MCU (STM32L073) inside S76G/S78G would still be active.

Example:

```
gps set_mode idle
```

```
>> Ok
```

```
gps sleep on 0
```

```
>> Ok
```

```
gps sleep off
```

```
>> Ok
```

### 3.3.10 gps get\_ttff

Purpose: To show the TTFF (Time To First Fix) value of the last GPS positioning. And this value would be reset as “0.0s” when S76G/S78G boots up, reset by “sip reset” or enter GPS Idle mode by setting “gps set\_mode idle”.

Example:

```
gps get_ttff
```

```
>> 5.6s
```

*(More demonstration examples, please refer to chapter 4.4)*

### 3.3.11 gps set\_satellite\_system <satellite\_system>

<satellite\_system>: A string representing S76G/S78G GPS satellite system is set to either “GPS”, “GLONASS”, “GPS+GLONASS”, “BEIDOU”, “GPS+BEIDOU”, so the string can be **gps**, **glonass**, **gps\_glonass**, **beidou**, **gps\_beidou**.

If the GPS firmware supported the GPS and GLONASS, then the input string can be **gps**, **glonass**,

**gps\_glonass.**

If the GPS firmware supported the GPS and BEIDOU, then the input string can be **gps**, **beidou**, **gps\_beidou**

Response: **Ok**, if <satellite\_system> string is valid.

**Invalid**, if <satellite\_system> string is not valid.

**“GPS just support gps, glonass or gps+glonass.”**, if <satellite\_system> string is not valid.

And the GPS firmware supported the GPS and GLONASS.

**“GPS just support gps, beidou or gps+beidou.”**, if <satellite\_system> string is not valid.

And the GPS firmware supported the GPS and BEIDOU

Purpose:

a) “GPS” is selected by default.

b) The setting value of this command saves into EEPROM immediately after this command executed

Example:

```
gps get_mode
```

```
>> off hot 20 3000 kiwi gps
```

```
gps set_satellite_system gps_glonass
```

```
>> Ok
```

```
gps get_mode
```

```
>> off hot 20 3000 kiwi gps_glonass
```

### 3.3.12 gps set\_start <start\_type>

<start\_type>: A string representing the way of how S76G/S78G starts to get position, so the string can be **hot**, **warm** or **cold**.

Response: **Ok**, if <start\_type> string is valid.

**Invalid**, if <start\_type> string is not valid.

Purpose:

- There're 3 types for GPS/GPS\_GLONASS starting and they are called as hot, warm and cold.
- If user didn't use this command after "sip factory\_reset", the default value is set at hot start.
- After calling this command, the setting value would be stored into EEPROM immediately

Example:

```
gps set_start cold
```

```
>> Ok
```

```
gps get_mode
```

```
>> off cold 20 3000 kiwi gps
```

### 3.3.13 gps set\_low\_power <switch>

<switch>: A string representing whether S7XG enables SONY Low Power Mode feature, it can be **on** or **off**.

Response: **Ok**, if <start\_type> string is valid.

**Invalid**, if <start\_type> string is not valid.

Default value is **on**.

Purpose: S7XG could be benefited from reducing SONY GPS power consumption while some satellites are tracked (positioning is fixed) and positioning cycle setting is over 30 seconds.

Example:

```
gps set_low_power on
```

```
>> Ok
```



## 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 S76G 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_appewi 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 GPS commands example

### 4.2.1 GPS Manual Mode

sip reset

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

>> S76G - v1.6.1-g6 - AS923 - Nov 20 2017 - 10:47:10

gps set\_level\_shift on

>> Ok

gps set\_start hot

>> Ok

gps set\_satellite\_system gps

>> Ok

gps set\_positioning\_cycle 5000

>> Ok

gps set\_port\_uplink 20

>> Ok

gps set\_format\_uplink ipso

>> Ok

gps set\_mode manual

>> Ok

gps get\_mode

```
>> manual hot 20 5000 ipso gps
```

```
gps get_data dms
```

```
>> POSITIONING ( 17.5s )
```

```
gps get_data dms
```

```
>> POSITIONING ( 30.2s )
```

```
gps get_data dms
```

```
>> POSITIONING ( 44.1s )
```

```
gps get_data dms
```

```
>> POSITIONING ( 68.4s )
```

```
gps get_data dms
```

```
>> POSITIONING ( 81.3s )
```

```
gps get_data dms
```

```
>> POSITIONING ( 100.6s )
```

```
gps get_data dms
```

```
>> DMS UTC( 2017/12/27 02:29:25 ) LAT( 24*59'20.50" N ) LONG( 121*18'30.48" E )  
POSITIONING( 0.3s )
```

```
gps get_data dms
```

```
>> DMS UTC( 2017/12/27 02:29:25 ) LAT( 24*59'20.50" N ) LONG( 121*18'30.48" E )  
POSITIONING( 1.9s )
```

```
gps get_ttf
```

```
>> 103.4s
```

#### Note: GPS UTC & DMS Format

```
">> DMS UTC( 2017/12/27 02:29:25 ) LAT( 24*59'20.50" N ) LONG( 121*18'30.48" E )  
POSITIONING( 1.9s )" "
```

UTC: 2Hour 29 Min 25 Sec (UTC+8: 10:29:25@Taiwan)

LAT & LONG: Latitude & Longitude defined as DMS coordinate system (121° 18' 30.48", DMS format)

POSITIONING: The current time - The last GPS positioning time → "Positioning Time"

## 4.2.2 GPS Auto Mode

sip reset

```
_____  
/ | ___/___/___ \\  
// | | ___ ^ ___ V // // Tech Co., LTD  
/ ___//___//___/ LoRaWAN v1.0.2 Ready  
// | \___//___// // (Class A & C)
```

>> S76G - v1.6.1-g6 - AS923 - Nov 20 2017 - 10:47:10

gps set\_level\_shift on

>> Ok

gps set\_start hot

>> Ok

gps set\_satellite\_system gps

>> Ok

gps set\_positioning\_cycle 5000

>> Ok

gps set\_format\_uplink ipso

>> Ok

gps set\_port\_uplink 20

>> Ok

gps set\_mode auto

>> Ok

mac set\_tx\_mode cycle

```
>> Ok  
mac set_tx_confirm off
```

```
>> Ok  
mac set_tx_interval 8000
```

```
>> Ok  
mac save
```

```
>> Ok  
mac join otaa
```

```
...
```

(It will upload GPS data by LoRaWAN port 20 when GPS is already positioned well)

### 4.2.3 Enter & Leave GPS Sleep

`gps set_mode idle` ==> GPS enters IDLE mode and stop positioning.

```
>> Ok  
gps sleep on 0 ==> GPS enters Sleep mode.
```

```
>> Ok  
gps sleep off
```

```
>> Ok  
gps sleep on 1 ==> GPS enters Deep Sleep mode.
```

```
>> Ok  
gps sleep off
```

```
>> Ok
```

### 4.2.4 Get GPS TTFF Value

```
gps set_level_shift on
```

```
>> Ok
```

```
gps set_mode manual
```

```
>> Ok
```

```
gps get_data dms
```

```
>> POSITIONING ( 8.4s )
```

```
gps get_data dms
```

```
>> POSITIONING ( 12.3s )
```

```
gps get_data dms
```

```
>> POSITIONING ( 26.8s )
```

```
gps get_data dms
```

```
>> POSITIONING ( 55.7s ) ==> Still positioning...
```

```
gps get_data dms
```

```
>> DMS UTC( 2017/12/08 01:34:48 ) LAT( 24*59'21.50" N ) LONG( 121*18'31.00" E )  
POSITIONING( 0.7s )
```

```
gps get_data dms
```

```
>> DMS UTC( 2017/12/08 01:34:51 ) LAT( 24*59'20.27" N ) LONG( 121*18'30.40" E )  
POSITIONING( 0.1s )
```

```
gps get_ttf
```

```
>> 62.2s ==> Shows the total positioning time from sending the command, "gps set_mode manual".
```

```
gps set_mode idle
```

```
>> Ok ==> GPS enters IDLE mode and stop positioning.
```

```
gps set_mode manual
```

```
>> Ok ==> GPS starts to position
```

```
gps get_ttf
```

```
>> 0.0s
```

```
gps get_ttf
```



```
>> 0.0s
```

```
gps get_data dms
```

```
>> DMS UTC( 2017/12/08 01:38:57 ) LAT( 25*2'16.86" N ) LONG( 121*19'43.20" E )  
POSITIONING( 2.1s )
```

```
gps get_tff
```

```
>> 5.6s ==> Shows the total positioning time from sending the command, "gps set_mode manual".
```

## 4.2.5 GPS Auto Mode & GPS Auto Join

(Take Kiwi Server as demonstration example)

```
sip factory_reset
```

```
>> v1.6.2-g7.rc1
```

```
mac save
```

```
>> Ok
```

```
sip reset
```

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

```
>> S76G - v1.6.2-g7.rc2 - AS923 - Jan 3 2018 - 13:11:55
```

```
gps set_level_shift on
```

```
>> Ok
```

```
gps set_format_uplink kiwi
```

```
>> Ok
```

```
gps set_port_uplink 3
```

```
>> Ok
```

```
gps set_start hot
```

```
>> Ok
gps set_satellite_system gps

>> Ok
gps set_positioning_cycle 3000

>> Ok
gps get_mode

>> off hot 3 3000 kiwi gps
gps set_mode auto

>> Ok
mac set_devaddr 1e000001

>> Ok
mac set_appei 0000ac0000000002

>> Ok
mac set_tx_mode cycle

>> Ok
mac set_tx_confirm off

>> Ok
mac set_adr off

>> Ok
mac set_ch_freq 2 923600000

>> Ok
mac set_ch_freq 3 923800000

>> Ok
mac set_ch_freq 4 924000000
```

>> Ok

```
mac set_ch_freq 5 924200000
```

>> Ok

```
mac set_ch_freq 6 924400000
```

>> Ok

```
mac set_ch_freq 7 924600000
```

>> Ok

```
mac set_auto_join on otaa 3
```

>> Ok

```
gps set_mode auto ==> The setting value would be stored into EEPROM automatically.
```

>> Ok

```
mac save ==> This "mac save" is used for "mac set_auto_join" and other "mac" commands.
```

>> Ok

```
sip reset
```

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

>> S76G - v1.6.2-g7.rc2 - AS923 - Jan 3 2018 - 13:11:55

>> gps\_in\_auto

>> join by otaa (3)

>> accepted

```
gps get_ttf
```

```
>> 102.6s
```

(... Starts to upload GPS coordinates data after positioning)

(To disable GPS Auto Mode & Auto Join Mode)

```
gps set_mode off
```

```
>> Ok
```

```
mac set_auto_join off
```

```
>> Ok
```

```
mac save
```

```
>> Ok
```

```
sip reset
```

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

```
>> S76G - v1.6.2-g7.rc2 - AS923 - Jan 3 2018 - 13:11:55
```

