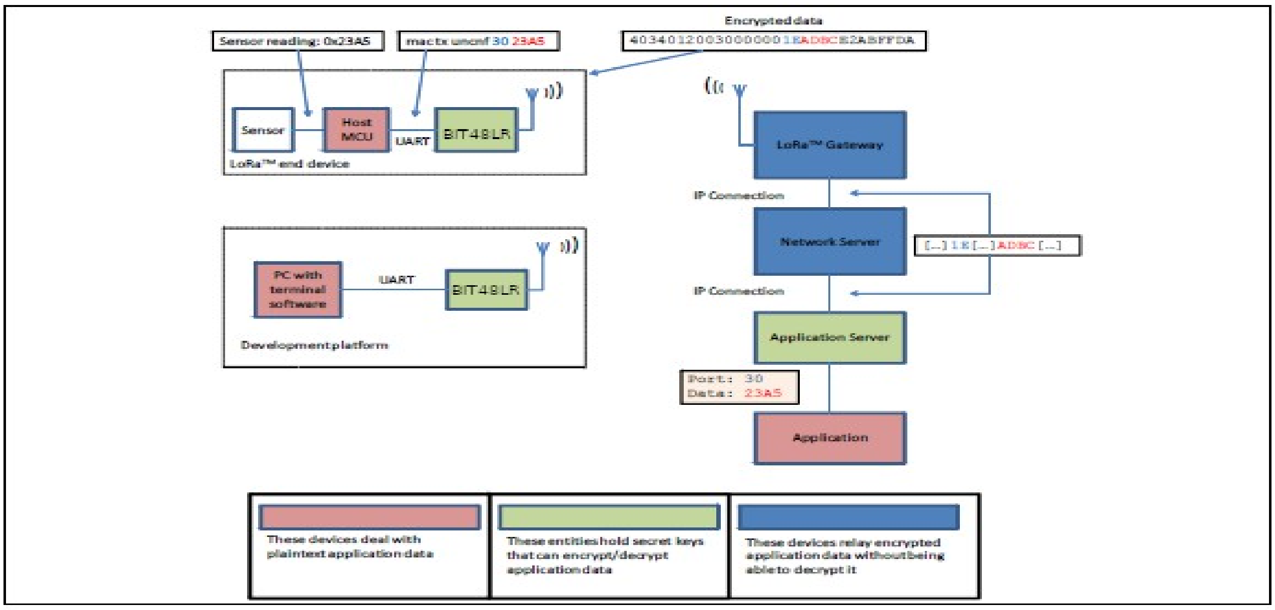
1 *Introduction*

1.1 Overview

The BIT48LR module provides LoRaWAN™ protocol connectivity using a simple UART interface. This module handles the LoRaWAN Class A and Class C protocols and provides an optimized text command/response interface to the host system. This document is intended to describe an implementation of the LoRaWAN Class A and Class C protocols. LoRaWAN protocol terms are described in more detail in the *LoRaWAN™ Specification V1.1* available from the LoRa Alliance (http://www.lora-alliance.org). Thus, it is recommended to review the *LoRaWAN™ Specification V1.1* before using the BIT48LR module.

FIGURE 1-1: SIMPLE LoRa® TECHNOLOGY NETWORK DIAGRAM


Sensor reading: 0x23A5
mac tx uncnf 30 23A5
Encrypted data
40340120030000001EADBCE2ABFFDA
Sensor
Host MCU
UART
BIT48LR
LoRa™ end device
PC with terminal software
UART
BIT48LR
Development platform
LoRa™ Gateway
IP Connection
Network Server
[...] 1E [...] ADBC [...]
IP Connection
Application Server
Port: 30
Data: 23A5
Application
These devices deal with plaintext application data
These entities hold secret keys that can encrypt/decrypt application data
These devices relay encrypted application data without being able to decrypt it

The required configuration for accessing a LoRa technology network is minimal and can be stored in the module's EEPROM, allowing for factory configuration of these parameters, lowering the requirements for the host system while also increasing system security. The module also features GPIO pins that can be configured through the UART interface.
A simple use case is described in FIGURE 1-1 where an end device, containing a host MCU which reads a sensor, commands the BIT48LR to transmit the sensor reading over the LoRa network. Data are encrypted by the BIT48LR and the radio packet is received by one or multiple gateways which forward it to the network server. The network server sends the data to the application server which has the key to decrypt the application data. Similarly, a development platform may consist of an BIT48LR directly connected over UART to a PC which becomes the host system in this case.
Users can then type commands into the module using a terminal program.

1.2 Features

- LoRaWAN Class A and Class C protocol compliance

- Integrated FSK, GFSK and LoRa technology transceiver allowing the user to
- transmit custom packets using these protocols
- Globally unique 64-bit identifier (EUI-64™)
- Configurable GPIOs
- Intelligent Low-Power mode with programmable/on-demand wake up
- All configuration and control done over UART using simple ASCII commands

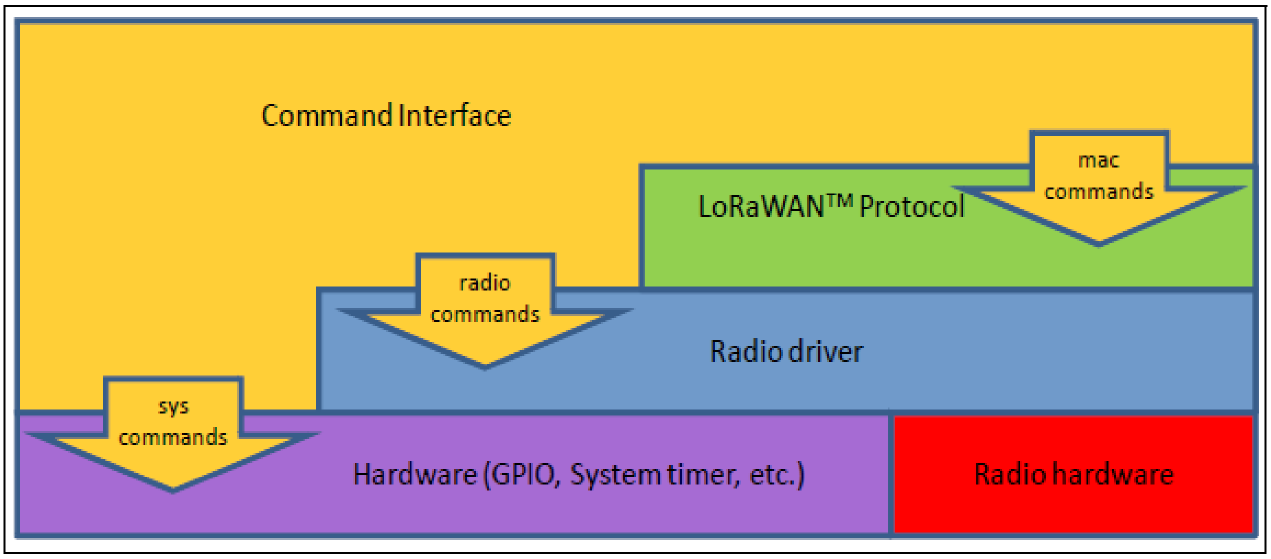
Refer to the ***BIT48LR_LoRa®TechnologyTransceiverModule_ds*** for details on the hardware specifications of the module.

1.3 Configuration

The BIT48LR module's architecture is described in FIGURE 1-2 from the command interface point of view. There are three types of commands that can be used, and each allows access to different module functions:

- LoRaWAN Class A and Class C configuration and control, using the mac group of commands
- Low-level radio configuration and control, using the radio group of commands
- Other module functions, using the sys group of commands

FIGURE 1-2: BIT48LR COMMAND INTERFACE (YELLOW) AND ITS RELATIONSHIP TO THE MODULE'S INTERNAL COMPONENTS



The available commands can be used to configure and control the LoRaWAN protocol layer, the radio driver and some system peripherals.
In order to communicate with a LoRa network, a specific number of parameters need to be configured. Since two distinctive methods are offered for a device to become part of the network, each of these requires different parameters:

- Over-the-Air Activation (OTAA), where a device negotiates network encryption keys at the time it joins the network. For this, the device EUI, application EUI and application key need to be configured and then the OTAA procedure can start.
- Activation by Personalization (ABP) where the device already contains the network

keys and can directly start communication with the network. Configuring the device address, network session key and application session key is sufficient for this type of initialization.

For increased security, these parameters can be configured and stored in the module’s EEPROM during manufacturing of devices requiring LoRaWAN connectivity. Thus, the keys do not need to be sent over the UART interface by the host system every time the device powers up.

1.4 Recommended Reading

This command reference user’s guide describes how to configure the BIT48LR module. The module-specific data sheet contains current information on the module specifications. Other useful documents are listed below. The following documents are available and recommended as supplemental reference resources:

BIT48LR_LoRa®TechnologyTransceiverModule_ds
This data sheet provides detailed specifications for the BIT48LR module.

LoRa® Alliance: LoRaWAN™ Specification V1.1
This document describes the LoRaWAN Class A protocol, which is optimized for battery-powered end devices. This specification is available from the LoRa Alliance at *https://lora-alliance.org*.

1.5 UART Interface

All of the BIT48LR module’s settings and commands are transmitted over UART using the ASCII interface.

All commands need to be terminated with <CR><LF> and any replies they generate will also be terminated by the same sequence.

The settings for the UART interface are 115200 bps, 8 bits, no parity, 1 Stop bit, no flow control.

2 Command Reference

The BIT48LR LoRa technology module supports a variety of commands for configuration. This section describes these commands in detail and provides examples.

2.1 Command Syntax

To issue commands to the BIT48LR module, the user sends keywords followed by optional parameters. Commands (keywords) are case-sensitive, and spaces must not be used in parameters. Hex input data can be uppercase or lowercase. String text data, such as OTAA used for the join procedure, can be uppercase or lowercase.

The use of shorthand for parameters is *NOT* supported.

Depending on the command, the parameter may expect values in either decimal or hexadecimal form; refer to the command description for the expected form. For example, when configuring the frequency, the command expects a decimal value in Hertz such as 868100000 (868.1 MHz). Alternatively, when configuring the LoRaWAN device address, the hex value is entered into the parameter as aabbccdd. To enter a number in hex form, use the value directly. For example, the hex value 0xFF would be entered as FF.

2.2 Command Organization

There are three general command categories, as shown in Table 2-1.

TABLE 2-1: COMMAND TYPES

Command Type	**Keyword**	**Description**
System	`<sys>`	Issues system level behavior actions, gathers status information on the firmware and hardware version, or accesses the module user EEPROM memory.
LoRaWAN™ Class A and Class C Protocols	`<mac>`	Issues LoRaWAN Class A and Class C protocols network communication behaviors, actions and configuration commands.
Transceiver commands	`<radio>`	Issues radio specific configurations, directly accessing and updating the transceiver setup.

After configuring the LoRaWAN protocol settings, the user must save them to EEPROM with the `sys save` command. Once the settings have been saved, they will be retained after a reboot or Reset.

Note: Upon successful reception a command, based on the specific command, the module will respond with one of the following:

- `ok`
- `busy`
- `fram_counter_err_rejoin_needed`
- `invalid_class`
- `invalid_data_len`
- `invalid_param`
- `keys_not_init`
- `mac_paused`
- `multicast_keys_not_set`
- `no_free_ch`
- `not_joined`
- `silent`
- `err`

Note: To facilitate the sharing of the radio between user custom applications and the LoRaWAN MAC, refer to the `mac pause` and `mac resume` commands. Since no sharing exists between `sys` and other types of commands, there is no need for additional `pause` commands.

2.3 SYSTEM COMMANDS

System commands begin with the system keyword `<sys>` and include the categories shown in TABLE 2-2, TABLE 2-3, TABLE 2-4.

TABLE 2-2: SYSTEM COMMANDS

Parameter	Description
`sleep`	Puts the system in sleep for a finite number of milliseconds.
`reset`	Resets and restarts the BIT48LR module.
`FactoryRESET`	Resets the BIT48LR module's configuration data and user EEPROM to factory default values and restarts the BIT48LR module.
`set` (1)	Sets specified system parameter values.
`get` (1)	Gets specified system parameter values.
`save`	Save all system set configurations

Note 1: Refer to TABLE 2-3 for `system <set>` and TABLE 2-4 for `system <get>` command summaries.

2.3.1 `sys sleep <length>`

`<length>`: decimal number representing the number of milliseconds the system is put to sleep, from 100 to 4294967296.
Response: `invalid_param` if the length is not valid
`ok` immediately if syntax is correct
`wakeup` after the system gets back from Sleep mode
This command puts the system to sleep for the specified number of milliseconds. The module can be forced to exit from sleep by sending the UART a Break condition followed by a `0x55` character. Forcing the module from sleep in the manner also triggers the UART auto baud detection. The module will adjust the UART baud rate to match the baud rate at which the `0x55` character was sent. Refer to the note box in 1.5 UART Interface
Example: `sys sleep 120` // Puts the system to sleep for 120 ms.

2.3.2 `sys reset`

Response: `BIT48LR X.Y.Z`, where X.Y.Z is firmware version.
This command resets and restarts the BIT48LR module; stored LoRaWAN protocol settings will be loaded automatically upon reboot.
Example: **`sys reset`** // Resets and restarts the BIT48LR module.

2.3.3 `sys factoryRESET`

Response: `BIT48LR X.Y.Z`, where X.Y.Z is firmware version.
This command resets the module's configuration data and user EEPROM to factory default values and restarts the module. After `factoryRESET`, the BIT48LR module will automatically reset and all configuration parameters are restored to factory default values.

All LoRaWAN protocol settings set by the user will be lost.
Example: **`sys factoryRESET`** // Restores factory default values.

2.3.4 System Set Commands

TABLE 2-3: SYSTEM SET COMMANDS

Parameter	Description
`nvm`	Stores `<data>` to a location `<address>` of user EEPROM.
`pindig`	Allows user to set and clear available digital pins.
`pinmode`	Allows user to set the functionality of a pin to either digital input, digital output or analog input (if available).
`mode`	Set the behaviour of BIT48LR: LoRa mode or transparent Radiomodem

2.3.4.1 `sys set nvm <address> <data>`

`<address>`:	hexadecimal number representing user EEPROM address, from 00 to 3F
`<data>`:	hexadecimal number representing data, from 00 to FF
Response:	`ok` if the parameters (address and data) are valid
	`invalid_param` if the parameters (address and data) are not valid

This command allows the user to modify the user EEPROM at `<address>` with the value supplied by `<data>`. Both `<address>` and `<data>` must be entered as hex values. The user EEPROM memory is located inside the MCU on the module.
Example: **`sys set nvm 30 A5`** // Stores `0xA5` at user EEPROM address `0x30`.

2.3.4.2 `sys set pindig <pinname> <pinstate>`

`<pinname>`:	string representing the pin. Parameter can be: `GPIO0 - GPIO7`
`<pinstate>`:	decimal number representing the state. Values can be: 0 or 1.
Response:	`ok` if the parameters (`<pinname>`, `<pinstate>`) are valid
	`invalid_param` if the parameters (`<pinname>`, `<pinstate>`) are not valid

This command allows the user to modify the unused pins available for use by the module. The selected `<pinname>` is driven high or low depending on the desired `<pinstate>`.
Default: `GPIO0-GPIO7` are driven low (value 0).
Example: **`sys set pindig GPIO5 1`** // Drives GPIO5 high 1, VDD.

> **Note:** In order for the pin to be driven to a value, make sure you have first configured the pin to be a digital output using the command `sys set pinmode <pinname> digout`.

2.3.4.3 sys set pinmode <pinname> <pinmode>

`<pinname>`:	string representing the pin. Parameters can be: `GPIO0` - `GPIO7`
`<pinmode>`:	string representing the functional mode of the pin. Parameters can be:

	`digout`, `digin` or `ana`.
Response:	`ok` if all the parameters are valid
	`invalid_param` if any of the parameters are not valid

This command allows the user to configure the functional mode of a pin. A pin can be configured as digital output by using the `digout` parameter. A pin can be configured as digital input by using the `digin` parameter. A pin can be configured as analog input by using the `ana` parameter.

Note: Not all pins have analog input functionality. Only `GPIO0` and `GPIO1` can be set in analogic mode

Example: `sys set pinmode GPIO0 ana` //Configures GPIO0 as analog input

Note: This command must be called prior to reading or setting the value of a pin in order to have correct behavior.

2.3.4.4 `sys set mode <mode>`

`<mode>`:	string representing the behaviour of BIT48LR. Parameters can be: `lorawan` or `bitrm`
Response:	`ok` if all the parameters are valid
	`invalid_param` if any of the parameters are not valid

This command allows the user to configure the functional mode of the BIT48LR.

2.3.5 System Get Commands

TABLE 2-4: SYSTEM GET COMMANDS

Parameter	Description
`ver`	Returns the information on hardware platform, firmware version, release date.
`hweui`	Returns the preprogrammed EUI node address.
`pindig`	Returns the state of a digital input.
`nvm`	Returns data from the requested user EEPROM `<address>`.
`pinana`	Returns the state of an analog input.
`mode`	Returns the state of BIT48LR mode

2.3.5.1 `sys get ver`

Response: `BIT48LR X.Y.Z`, where X.Y.Z is firmware version.

This command returns the information related to the hardware platform, firmware version, release date and time stamp on firmware creation.

Example: **`sys get ver`** // Returns version-related information.

2.3.5.2 `sys get hweui`

Response: hexadecimal number representing the preprogrammed EUI node address

This command reads the preprogrammed EUI node address from the BIT48LR module. The value returned by this command is a globally unique number.

Example: **`sys get hweui`** // Reads the preprogrammed EUI node address.

> **Note:** The preprogrammed EUI node address is a read-only value and cannot be changed or erased. This value can be used to configure the device EUI using the `mac set deveui` command (see 2.4.6.1)

2.3.5.3 `sys get pindig <pinname>`

<`pinname`>: string representing the pin. Parameters can be: `GPIO0 - GPIO7`

Response: decimal number representing the state (either 0 or 1).

This command allows the user to read the state of a digital input. To be used as a digital input, a pin needs to be configured using the `sys set pinmode` command.

Example: **`sys get pindig GPIO0`** //Reads the state of the GPIO0 //digital input

> **Note:** The `sys set pinmode <pinname> digin` command must be called to configure the function of the pin prior to reading its digital input value.

2.3.5.4 `sys get nvm <address>`

`<address>`: hexadecimal number representing user EEPROM address, from 00 to 3F

Response: 00 – FF (hexadecimal value from 00 to FF) if the address is valid
`invalid_param` if the address is not valid

This command returns the data stored in the user EEPROM of the BIT48LR module at the requested `<address>` location.

Example: **`sys get nvm 30`** // Returns the 8-bit hex value stored at 30.

2.3.5.5 `sys get pinana <pinname>`

`<pinname>`: string representing the pin. Parameters can be: `GPIO0` - `GPIO1`

Response: decimal number representing the result of the conversion, from 0 to 1023, where 0 represents 0V and 1023 is VDD, the supply voltage of the module.

This command allows the user to read the state of an analog input. To be used as an analog input, a pin needs to be configured using the `sys set pinmode` command.

Example: **`sys get pinana GPIO0`** //Reads the state of the GPIO0 analog input

> **Note:** The `sys set pinmode <pinname> ana` command must be called to configure the function of the pin prior to reading its analog input value.

2.3.5.6 `sys get mode`

Response: string representing the operating mode of BIT48LR. Values can be `lorawan` or `bitrm`

This command allows the user to get the functional mode of the BIT48LR.

Default: `lorawan`

Example: **`sys get mode`**

2.3.6 `sys save`

Response: `ok`

The `sys save` command must be issued after configuration parameters have been appropriately entered from the `sys set <cmd>`, `mac set <cmd>` and `radioset <cmd>` commands. This command will save BIT48LR mode and LoRaWAN Class A protocol configuration parameters to the user EEPROM. When the next `sys reset` command is issued, the configuration will be initialized with the last saved parameters.

Example: **`sys save`** // Saves the sys set configuration parameters to // the user EEPROM.

2.4 MAC COMMANDS

LoRaWAN protocol commands begin with the system keyword `mac` and include the categories shown in TABLE 2-5.

TABLE 2-5: MAC COMMANDS

Parameter	**Description**
`reset`	Resets the BIT48LR module to a specific frequency band.
`tx`	Sends the data string on a specified port number and sets default values for most of the LoRaWAN™ parameters.
`join`	Informs the BIT48LR module to join the configured network.
`pause`	Pauses LoRaWAN stack functionality to allow transceiver (radio) configuration.
`resume`	Restores the LoRaWAN stack functionality.
`set`	Accesses and modifies specific MAC related parameters.
`get`	Reads back current MAC related parameters from the module.

2.4.1 `mac reset <band>`

`<band>`: string representing the frequency band; Parameters can be: `EU868, NA915, AU915, AS923, JPN923, KR920, IND865`

Response: `ok` if band is valid
`invalid_param` if band is not valid

This command will automatically reset the software LoRaWAN stack and initialize it with the default parameters.

Example: `mac reset EU868` // Sets the default values and selects the // 868 default band.

Note:	This command will set default values for most of the LoRaWAN parameters. Everything set prior to this command will lose its set value.

2.4.2 `mac tx <type> <portno> <data>`

`<type>`:	string representing the uplink payload type, either `cnf` or `uncnf` (`cnf` – confirmed, `uncnf` – unconfirmed)
`<portno>`:	decimal number representing the port number, from 1 to 223
`<data>`:	hexadecimal value. The length of `<data>` bytes capable of being transmitted are dependent upon the set data rate (for further details, refer to the *LoRaWAN™ Specification V1.1*).
Response:	this command may reply with two responses. The first response will be received immediately after entering the command. In case the command is valid (`ok` reply received), a second reply will be received after the end of the uplink transmission. For further details, refer to the *LoRaWAN™ Specification V1.1*.

Response after entering the command:

- `ok` – if parameters and configurations are valid and the packet was forwarded to the radio transceiver for transmission
- `invalid_param` – if parameters (`<type> <portno> <data>`) are not valid
- `not_joined` – if the network is not joined
- `no_free_ch` – if all channels are busy
- `silent` – if the module is in a Silent Immediately state
- `frame_counter_err_rejoin_needed` – if the frame counter rolled over
- `busy` – if MAC state is not in an Idle state
- `mac_paused` – if MAC was paused and not resumed back
- `invalid_data_len` if application payload length is greater than the maximum application payload length corresponding to the current data rate

Response after the first uplink transmission attempt:

- `mac_tx_ok` if uplink transmission was successful and no downlink data was received back from the server;
- `mac_rx <portno> <data>` if transmission was successful, `<portno>`: port number, from 1 to 223; `<data>`: hexadecimal value that was received from the server;
- `mac_err` if transmission was unsuccessful, ACK not received back from the server
- `invalid_data_len` if application payload length is greater than the maximum application payload length corresponding to the current data rate. This can occur after an earlier uplink attempt if retransmission back-off has reduced the data rate.

A confirmed message will expect an acknowledgment from the server; otherwise, the message will be retransmitted by the number indicated by the command `mac set retx <value>`, whereas an unconfirmed message will not expect any acknowledgment back

from the server. For further details, refer to the *LoRaWAN™ Specification V1.1*.
The port number allows multiplexing multiple data streams on the same link. For example, the end device can send measurements on one port number and configuration data on another. The server application can then distinguish the two types of data based on the port number.
Example: **`mac tx cnf 4 5A5B5B`** // Sends a confirmed frame on port 4 with // application payload 5A5B5B.
If the automatic reply feature is enabled and the server sets the Frame Pending bit or initiates downlink confirmed transmissions, multiple responses will be displayed after each downlink packet is received by the module. A typical scenario for this case would be (prerequisites: free LoRaWAN channels available and automatic reply enabled):

- The module sends a packet on port 4 with application payload `0xAB`
- Radio transmission is successful and the module will display the first response: `ok`
- The server needs to send two separate downlink confirmed packets back on port 1 with the following data: `0xAC`, then `0xAF`. First it will transmit the first one (`0xAC`) and will set the Frame Pending bit. The module will display the second response `mac_rx 1 AC`
- The module will initiate an automatic uplink unconfirmed transmission with no application payload on the first free channel because the Frame Pending bit was set in the downlink transmission
- The server will send back the second confirmed packet (`0xAF`). The module will display a third response `mac_rx 1 AF`
- The module will initiate an automatic unconfirmed transmission with no application payload on the first free channel because the last downlink transmission was confirmed, so the server needs an ACK
- If no reply is received back from the server, the module will display the fourth response after the end of the second Receive window: `mac_tx_ok`
- After this scenario, the user is allowed to send packets when at least one enabled channel is free

Based on this scenario, the following responses will be displayed by the module:

- `mac tx cnf 4 AB`
- `ok`
- `mac_rx 1 AC`
- `mac_rx 1 AF`
- `mac_tx_ok`

2.4.3 `mac join <mode>`

`<mode>`:	string representing the join procedure type (case-insensitive), either `otaa` or `abp` (`otaa` – over-the-air activation, `abp` – activation by personalization).
Response:	this command may reply with two responses. The first response will be received immediately after entering the command. In case the command is valid (`ok` reply received) a second reply will be received after the end of the join procedure. For further details, refer to *LoRaWAN™ Specification V1.1*.

Response after entering the command:

- `ok` – if parameters and configurations are valid and the join request packet was forwarded to the radio transceiver for transmission

- `invalid_param` – if `<mode>` is not valid
- `keys_not_init` – if the keys corresponding to the Join mode (`otaa` or `abp`) were not configured
- `no_free_ch` – if all channels are busy
- `silent` – if the device is in a Silent Immediately state
- `busy` – if MAC state is not in an Idle state
- `mac_paused` – if MAC was paused and not resumed back

Response after the join procedure:

- `denied` if the join procedure was unsuccessful (the module attempted to join the network, but was rejected);
- `accepted` if the join procedure was successful;

This command informs the BIT48LR module it should attempt to join the configured network. Module activation type is selected with `<mode>`. Parameter values can be `otaa` (over-the-air activation) or `abp` (activation by personalization). The `<mode>` parameter is not case sensitive. Before joining the network, the specific parameters for each activation type should be configured (for over the air activation: device EUI, application EUI, application key; for activation by personalization: device address, network session key, application session key).

Example: **`mac join otaa`** // Attempts to join the network using over-the-air activation.

2.4.4 `mac pause`

Response:	0 – 4294967295 (decimal number representing the number of milliseconds the mac can be paused)

This command pauses the LoRaWAN stack functionality to allow transceiver (radio) configuration. Through the use of `mac pause`, radio commands can be generated between a LoRaWAN Class A protocol uplink application (`mac tx` command), and the LoRaWAN Class A protocol Receive windows (second response for the `mac tx` command). This command will reply with the time interval in milliseconds that the transceiver can be used without affecting the LoRaWAN functionality. The maximum value (4294967295) is returned whenever the LoRaWAN stack functionality is in Idle state and the transceiver can be used without restrictions. '`0`' is returned when the LoRaWAN stack functionality cannot be paused.

For example, when operating in LoRaWAN Class C mode, the receiver is continuously in receive. The `mac pause` command will return '`0`' indicating that the LoraWAN stack cannot be paused.

After the radio configuration is complete, the `mac resume` command must be used to return to LoRaWAN protocol commands.

Example:	**`mac pause`**	// Pauses the LoRaWAN stack functionality if the // response is different from 0.

Note:	If already joined to a network, this command MUST be called BEFORE configuring the radio parameters, initiating radio reception, or transmission.

2.4.5 `mac resume`

Response: `ok`

This command resumes LoRaWAN stack functionality, in order to continue normal functionality after being paused.

Example: **`mac resume`** // Resumes the LoRaWAN stack functionality.

Note:	This command MUST be called AFTER all radio commands have been issued and all the corresponding asynchronous messages have been replied.

2.4.6 MAC Set Commands

TABLE 2-6: MAC SET COMMANDS

Parameter	**Description**
`deveui`	Sets the globally unique identifier for the BIT48LR module.
`appeui`	Sets the application identifier for the end device.
`devaddr`	Sets the unique network device address for the BIT48LR module.
`appkey`	Sets the application key for the BIT48LR module.
`nwkskey`	Sets the network session key for the BIT48LR module.
`appskey`	Sets the application session key for the BIT48LR module.
`class`	Sets the LoRaWAN operating class.
`mcast`	Sets the Multicast state to on, or off.
`mcastappskey`	Sets the multicast application session key.
`mcastdevaddr`	Sets the multicast network device address.
`mcastnwkskey`	Sets the multicast network session key.
`adr`	Sets the state of the adaptive data rate mechanism.
`adrackdelay`	Sets the ADR Acknowledgment Delay - Refer LORa Regional parameters spec for more information on default value
`adracklimit`	Sets the ADR Acknowledgment Limit - Refer LORa Regional parameters spec for more information on default value
`ar`	Sets the state of the automatic reply.
`ch`	Allows modification of channel related parameters.
`dr`	Sets the data rate to be used for the next transmissions.
`lbt`	Sets Lorawan Listen Before Talk parameters
`pwridx`	Sets the output power to be used on the next transmissions.

`sync`	Sets the synchronization word for the LoRaWAN communication.
`upctr`	Sets the value of the uplink frame counter that will be used for the next uplink transmission.
`dnctr`	Sets the value of the downlink frame counter that will be used for the next downlink reception.
`rxdelay1`	Sets the value used for the first Receive window delay.
`acktimeout`	Sets Acknowledgment Timeout - Refer LORa Regional parameters spec for more information on default value
`joinacceptdelay1`	Sets JoinAcceptWindow1 Delay - Delay between the transmission of Join req message and first reception window. Refer LORa Regional parameters spec for more information on default value
`joinacceptdelay2`	Sets JoinAcceptWindow2 Delay - Delay between the transmission of Join req message and second reception window. Refer LORa Regional parameters spec for more information on default value
`rx2`	Sets the data rate and frequency used for the second Receive window.
`retxcnf`	Sets the number of retransmissions to be used for an uplink confirmed packet.
`retxuncnf`	Sets the number of retransmissions to be used for an uplink unconfirmed packet.
`linkchk`	Sets the time interval for the link check process to be triggered.
`maxfcntgap`	Sets Maximum Frame Counter Gap - Refer LORa Regional parameters spec for more information on default value
`testmode`	Sets Enables Test Mode which Disable Duty Cycle for a Device, Override Features Supported by Regulatory

Note(*): If any of these parameters was previously saved to user EEPROM by issuing the `sys save` command, after modifying its value, the `sys save` command should be called again.

2.4.6.1 **`mac set deveui <devEUI>`**

`<devEUI>`: 8-byte hexadecimal number representing the device EUI
Response: `ok` if address is valid
`invalid_param` if address is not valid

This command sets the globally unique device identifier for the module. The identifier must be set by the host MCU. The module contains a pre-programmed unique EUI and can be retrieved using the `sys get hweui` command (see 2.3.5.2) or user provided EUI can be configured using the `mac set deveui` command.

Example: **`mac set deveui 0004A30B001A55ED`**

2.4.6.2 `mac set appeui <appEUI>`

`<appEUI>`:	8-byte hexadecimal number representing the application EUI
Response:	`ok` if address is valid
	`invalid_param` if address is not valid

This command sets the application identifier for the module. The identifier must be set by the host MCU.

Example: **`mac set appeui 0001020304050607`**

2.4.6.3 `mac set devaddr <address>`

`<address>`:	4-byte hexadecimal number representing the device address, from 00000000 – FFFFFFFF
Response:	`ok` if address is valid
	`invalid_param` if address is not valid

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

Example: **`mac set devaddr ABCDEF01`**

2.4.6.4 `mac set appkey <appKey>`

`<appKey>`:	16-byte hexadecimal number representing the application key
Response:	`ok` if key is valid
	`invalid_param` if key is not valid

This command sets the application key for the module. The application key is used to derive the security credentials for communication during over-the-air activation.

Example: **`mac set appkey 00112233445566778899AABBCCDDEEFF`**

2.4.6.5 `mac set nwkskey <nwkSessKey>`

`<nwkSessKey>`:	16-byte hexadecimal number representing the network session key
Response:	`ok` if key is valid
	`invalid_param` if key is not valid

This command sets the network session key for the module. This key is 16 bytes in length, and provides security for communication between the module and network server.

Example: **`mac set nwkskey 1029384756AFBECD5647382910DACFEB`**

2.4.6.6 `mac set appskey <appSesskey>`

`<appSessKey>`:	16-byte hexadecimal number representing application session key
Response:	`ok` if key is valid
	`invalid_param` if key is not valid

This command sets the application session key for the module. This key provides security for communication between module and application server.

Example: **`mac set appskey AFBECD56473829100192837465FAEBDC`**

2.4.6.7 `mac set class <class>`

`<class>`: A letter representing the LoRaWAN device class, either a or c.
Response: `ok` if class is valid
`invalid_param` if the class is not valid

This command sets the end device LoRaWAN operating class. The default end device class is Class A. When the class is configured as Class C, the end device will enter Class C continuous receive mode after the next uplink message is sent. The LoRaWAN network server must also configure this node as a Class C node. The network server configuration is performed out of band from LoRaWAN communications. For more information on the description of operating in Class C mode, refer to the *LoRaWAN™ Specification V1.1*.

Example: **`mac set class c`**

2.4.6.8 `mac set mcast <state>`

`<state>`: string value representing the state, either on or off.
Response: `ok` if state is valid
`invalid_param` if the state is not valid

This command sets the end device Multicast state (mcast) to either be enabled or disabled. When multicast is enabled, and the device is operating in Class C continuous receive mode, the end device can receive multicast messages from the server. For more information on the description of multicast operation, refer to the *LoRaWAN™ Specification V1.1*.

Example: **`mac set mcast on`**

2.4.6.9 `mac set mcastappskey <mcastApplicationSessionkey>`

`<mcastApplicationSessionkey>`: 16-byte hexadecimal number representing the application session key.
Response: `ok` if key is valid
`invalid_param` if the key is not valid

This command sets the multicast application session key for the module. This key identifies the multicast application session key used when the network sends a multicast message from an application.

Example: **`mac set mcastappskey 29100192AFBECD564738837465FAEBDC`**

2.4.6.10 `mac set mcastdevaddr <mcastAddress>`

`<mcastAddress>`: 4-byte hexadecimal number representing the device multicast address, from 00000000 - FFFFFFFF.
Response: `ok` if address is valid
`invalid_param` if the address is not valid

This command configures the module with a 4-byte multicast network device address `<address>`. The `<address>` MUST match the multicast address on the current network. This must be directly set for multicast devices.

Example: **`mac set mcastdevaddr 54ABCDEF`**

2.4.6.11 `mac set mcastnwkskey <mcastNetworkSessionkey>`

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

Response: `ok` if key is valid
`invalid_param` if the key is not valid

This command sets the multicast network session key for the module. This key is 16 bytes in length, and provides security for communication between the module and multicast network server.

Example: **`mac set mcastnwkskey 6AFBECD1029384755647382910DACFEB`**

2.4.6.12 `mac set adr <adrState>`

`<adrState>`: string representing the state of the adaptive data rate mechanism, either `on` or `off`.

Response: `ok` if parameter is valid
`invalid_param` if the parameter is not valid

This command sets the state of the adaptive data rate mechanism.
LoRa network allows the end-devices to individually use any of the possible data rates, this is referred to as Adaptive Data Rate (ADR). If the ADR is set, the network will control the data rate of the end-device through the appropriate MAC commands. If the ADR is not set, the network will not attempt to control the data rate of the end-device regardless of the received signal quality.

Example: **`mac set adr on`**

2.4.6.13 `mac set adrackdelay <adrAckDelay>`

`<adrAckDelay>`: decimal number representing the ADR Acknowledgment Delay , from 0 to 15

Response: `ok` if parameter is valid
`invalid_param` if parameter is not valid

This command sets the ADR Acknowledgment Delay. Refer to the *LoRaWAN™ Specification V1.1*.

Example: **`mac set adrackdelay 5`**

2.4.6.14 `mac set adracklimit <adrAckLimit>`

`<adrAckLimit>`: decimal number representing the ADR Acknowledgment Limit , from 0 to 15

Response: `ok` if parameter is valid
`invalid_param` if parameter is not valid

This command sets the ADR Acknowledgment Limit. Refer to the *LoRaWAN™ Specification V1.1*.

Example: **`mac set adracklimit 5`**

2.4.6.15 `mac set ar <state>`

`<state>`:	string value representing the state, either `on` or `off`.
Response:	`ok` if state is valid
	`invalid_param` if state is not valid

This command sets the state of the automatic reply. By enabling the automatic reply, the module will transmit a packet without a payload immediately after a confirmed downlink is received, or when the Frame Pending bit has been set by the server. If set to OFF, no automatic reply will be transmitted.

Example: **`mac set ar on`** // Enables the automatic reply process
// inside the module.

Note: The BIT48LR module implementation will initiate automatic transmissions with no application payload if the automatic reply feature is enabled and the server sets the Frame Pending bit or initiates a confirmed downlink transmission. In this case, if all enabled channels are busy due to duty cycle limitations, the stack will wait for the first channel that will become free to transmit. The user will not be able to initiate uplink transmissions until the automatic transmissions are done.

2.4.6.16 MAC SET CHANNEL COMMANDS

TABLE 2-7: MAC SET CHANNEL COMMANDS

Parameter	Description
`freq`	Sets the module operation frequency on a given channel ID.
`dcycle`	Sets the module operation duty cycle on a given channel ID.
`drrange`	Sets the module allowed data rate range (min.- max.) allowed on a given channel ID.
`status`	Sets the use of the specified channel ID.

2.4.6.16.1 `mac set ch freq <channelID> <frequency>`

`<channelID>`:	decimal number representing the channel number, from 3 to 15.
`<frequency>`:	decimal number representing the frequency, from 863000000 to 870000000 in Hz.
Response:	`ok` if parameters are valid
	`invalid_param` if parameters are not valid

This command sets the operational frequency on the given channel ID. The default channels (0-2) cannot be modified in terms of frequency.

Example: **`mac set ch freq 13 864000000`** // Define frequency for
// channel 13 to be 864 MHz.

2.4.6.16.2 `mac set ch dcycle <channelID> <dutyCycle>`

`<channelID>`: decimal number representing the channel number, from 0 to 15.

`<dutyCycle>`:	decimal number representing the duty cycle, from 0 to 65535.
Response:	`ok` if parameters are valid
	`invalid_param` if parameters are not valid

This command sets the duty cycle used on the given channel ID on the module. The `<dutyCycle>` value that needs to be configured can be obtained from the actual duty cycle X (in percentage) using the following formula: `<dutyCycle>` = (100/X) – 1. The default settings consider only the three default channels (0-2), and their default duty cycle is 0.33%. If a new channel is created either by the server or by the user, all the channels (including the default ones) must be updated by the user in terms of duty cycle to comply with the ETSI regulations.

Example: **`mac set ch dcycle 13 9`** // Defines duty cycle for channel 13 to be
// 10%. Since (100/10) – 1 = 9, the
// parameter that gets configured is 9.

2.4.6.16.3 `mac set ch drrange <channelID> <minRange> <maxRange>`

`<channelID>`:	decimal number representing the channel number, from 0 to 15
`<minRange>`:	decimal number representing the minimum data rate, from 0 to 7
`<maxRange>`:	decimal number representing the maximum data rate, from 0 to 7
Response:	`ok` if parameters are valid
	`invalid_param` if parameters are not valid

This command sets the operating data rate range, min. to max., for the given `<channelID>`. By doing this the module can vary data rates between the `<minRange>` and `<maxRange>` on the specified `<channelID>`. For the actual values of the data rates and the corresponding spreading factors (SF), refer to the *LoRaWAN™ Specification*.

Example: **`mac set ch drrange 13 0 2`** // Using EU863-870 band: on
// channel 13 the data rate can range
// from 0 (SF12/125 kHz) to 2
// (SF10/125 kHz) as required.

2.4.6.16.4 `mac set ch status <channelID> <status>`

`<channelID>`:	decimal number representing the channel number, from 0 to 15.
`<status>`:	string value representing the state, either `on` or `off`.
Response:	`ok` if parameters are valid
	`invalid_param` if parameters are not valid

This command sets the operation of the given `<channelID>`.

Example: **`mac set ch status 4 off`** // Channel ID 4 is disabled from use.

2.4.6.17 `mac set dr <dataRate>`

`<dataRate>`:	decimal number representing the data rate, from 0 and 7, but within the limits of the data rate range for the defined channels.

Response:	`ok` if data rate is valid `invalid_param` if data rate is not valid

This command sets the data rate to be used for the next transmission. For the description of data rates and the corresponding spreading factors, refer to the *LoRaWAN™ Specification V1.1*.

Example: **`mac set dr 5`** // On EU863-870; SF7/125 kHz.

2.4.6.18 `mac set lbt <scanPer> <trshld> <maxRetry> <numSamples> <enable>`

`<scanPer>`	decimal number representing the scan period, from 0 to 65536; units are ms
`<trshld>`	decimal number representing the treshold, from -150 to 10; units are dBm
`<maxRetry>`	decimal number representing the max Retry, from 0 and 65536
`<numSamples>`	decimal number representing the number of samples, from 0 and 255
`<enable>`	string representing the state of the Listen Before Talk mechanism, either `on` or `off`.
Response:	`ok` if the parameters are valid invalid_param if the parameters are not valid

This command sets the Listen Before Talk mechanism. For the description of LBT, refer to the *LoRaWAN™ Specification V1.1*.

Note:	This command is valid only for some regions; it is available if region has been setted with `mac reset <band>` to `JPN923` or `KR920`. Please refer to 2.4.1

2.4.6.19 `mac set pwridx <pwrIndex>`

`<pwrIndex>`:	decimal number representing the index value for the output power, from 0 to 9.
Response:	`ok` if power index is valid `invalid_param` if power index is not valid

This command sets the Output power to be used on the next transmissions.
The corresponding output power is defined as **BIT48LRMaxOutputPower – 2* `pwrIndex`** where **BIT48LRMaxOutputPower** is 20 dBm.

Note:	It is possible to set the output power above the regulatory limits. This power setting allows some compensation on the cable or transmission line loss. Please refer to the *LoRaWAN™ Specification V1.1* and *LoRaWAN™ Regional Parameters v1.1* for the output power corresponding to the `<pwrIndex>` and also to the *BIT48LR_LoRa®TechnologyTransceiverModule_ds* for the actual radio power capabilities.

Example: **`mac set pwridx 4`** // Sets the TX output power to 12 dBm on the next

// transmission for a 868 MHz EU module.

2.4.6.20 `mac set sync <synchWord>`

`<synchWord>`:	one byte long hexadecimal number representing the synchronization word for the LoRaWAN communication
Response:	`ok` if parameters are valid
	`invalid_param` if parameter is not valid

This command sets the synchronization word for the LoRaWAN communication. The configuration of the synchronization word should be in concordance with the Gateway configuration.

Example: **`mac set sync 34`** // Synchronization word is configured // to use the 0x34 value

2.4.6.21 `mac set upctr <fCntUp>`

`<fCntUp>`:	decimal number representing the value of the uplink frame counter that will be used for the next uplink transmission, from 0 to 4294967295.
Response:	`ok` if parameter is valid
	`invalid_param` if parameter is not valid

This command sets the value of the uplink frame counter that will be used for the next uplink transmission.

Example: **`mac set upctr 10`**

2.4.6.22 `mac set dnctr <fCntDown>`

`<fCntDown>`:	decimal number representing the value of the downlink frame counter that will be used for the next downlink reception, from 0 to 4294967295.
Response:	`ok` if parameter is valid
	`invalid_param` if parameter is not valid

This command sets the value of the downlink frame counter that will be used for the next downlink reception.

Example: **`mac set dnctr 30`**

2.4.6.23 `mac set rxdelay1 <rxDelay>`

`<rxDelay>`:	decimal number representing the delay between the transmission and the first Reception window in milliseconds, from 0 to 65535.
Response:	`ok` if `<rxDelay>` is valid
	`invalid_param` if `<rxDelay>` is not valid

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

Example: **`mac set rxdelay1 1000`** // Set the delay between the transmission // and the first Receive window to 1000 ms.

2.4.6.24 `mac set acktimeout <ackTimeout>`

`<ackTimeout>`	decimal number representing the ACK Timeout, from 0 to 255.
Response:	`ok` if the parameter is valid invalid_param if the parameter is not valid

This command sets the Ack Timeout. Refer to the *LoRaWAN™ Specification V1.1*

Example: **`mac set acktimeout 50`**

2.4.6.25 `mac set joinacceptdelay1 <jDelay1>`

`<jDelay1>`	decimal number representing the delay1, from 0 to 65536.
Response:	`ok` if the parameter is valid invalid_param if the parameter is not valid

This command sets the Delay between the transmission of Join req message and first reception window. Refer LORa Regional parameters spec for more information on default value.

Example: **`mac set joinacceptdelay1 7000`**

2.4.6.26 `mac set joinacceptdelay2 <jDelay2>`

`<jDelay2>`	decimal number representing the delay2, from 0 to 65536.
Response:	`ok` if the parameter is valid invalid_param if the parameter is not valid

This command sets the Delay between the transmission of Join req message and second reception window. Refer LORa Regional parameters spec for more information on default value.

Example: **`mac set joinacceptdelay2  8000`**

2.4.6.27 `mac set rx2 <dataRate> <frequency>`

`<dataRate>`:	decimal number representing the data rate, from 0 to 7.
`<frequency>`:	decimal number representing the frequency, from 863000000 to 870000000 in Hz.
Response:	`ok` if parameters are valid `invalid_param` if parameters are not valid

This command sets the data rate and frequency used for the second Receive window. The configuration of the Receive window parameters should be in concordance with the server configuration.

Example: **`mac set rx2 3 865000000`** // Receive window 2 is configured with
// SF9/125 kHz data rate with a center
// frequency of 865 MHz.

2.4.6.28 `mac set retxcnf <reTxNb>`

`<reTxNb>`:	decimal number representing the number of retransmissions for an uplink confirmed packet, from 0 to 255.
Response:	`ok` if `<reTxNb>` is valid

invalid_param if <reTxNb> is not valid

This command sets the number of retransmissions to be used for an uplink confirmed packet, if no downlink acknowledgment is received from the server.

Example: **`mac set retxcnf 5`** // The number of retransmissions made for // an uplink confirmed packet is set to 5.

2.4.6.29 **`mac set retxuncnf <reTxuncnfNb>`**

`<reTxuncnfNb>`:	decimal number representing the number of retransmissions for an uplink unconfirmed packet, from 0 to 255.
Response:	`ok` if `<reTxuncnfNb>` is valid
	`invalid_param` if `<reTxuncnfNb>` is not valid

This command sets the number of retransmissions to be used for an uplink unconfirmed packet, if no downlink acknowledgment is received from the server.

Example: **`mac set retxuncnf 5`** // The number of retransmissions made for // an uplink unconfirmed packet is set to 5.

2.4.6.30 **`mac set linkchk <linkCheck>`**

`<linkCheck>`:	decimal number that sets the time interval in seconds for the link check process, from 0 to 65535
Response:	`ok` if the time interval is valid
	`invalid_param` if the time interval is not valid

This command sets the time interval for the link check process to be triggered periodically. A `<value>` of '`0`' will disable the link check process. When the time interval expires, the next application packet that will be sent to the server will include also a link check MAC command. For more information on the Link Check MAC command, refer to the *LoRaWAN™ Specification V1.1*.

Example: **`mac set linkchk 600`** // The module will attempt a link check // process at 600-second intervals.

2.4.6.31 **`mac set maxfcntgap <maxFcntGap>`**

<`maxFcntGap`**>**	decimal number representing the Maximum Frame Counter Gap, from 0 to 255.
Response:	`ok` if the parameter is valid
	invalid_param if the parameter is not valid

This command sets the Maximum Frame Counter Gap,. Refer LORa Regional parameters spec for more information on default value.

Example: **`mac set maxfcntgap  100`**

2.4.6.32 **`mac set testmode <testModeEn>`**

`<testModeEn>`	string representing the state of the Test mode, either `on` or `off`.
Response:	`ok` if parameter is valid
	invalid_param if the parameter is not valid

This command sets the Test Mode which Disable Duty Cycle for a Device, Override

Features Supported by Regulatory
Example: **`mac set testmode on`**

2.4.7 MAC Get Commands

TABLE 2-8: MAC GET COMMANDS

Parameter	Description
`deveui`	Gets the globally unique identifier for the BIT48LR module.
`appeui`	Gets the application identifier for the end device.
`devaddr`	Gets the unique network device address for the BIT48LR module.
`appkey`	Gets the application key for the BIT48LR module.
`nwkskey`	Gets the network session key for the BIT48LR module.
`appskey`	Gets the application session key for the BIT48LR module.
`class`	Gets the LoRaWAN operating class.
`mcast`	Gets the state of multicast reception for the end device.
`mcastappskey`	Gets the multicast application session key.
`mcastdevaddr`	Gets the multicast network device address.
`mcastdnctr`	Gets the value of the multicast downlink frame counter that will be used for the next multicast downlink reception.
`mcastnwkskey`	Gets the multicast network session key.
`adr`	Gets the state of adaptive data rate for the device.
`adrackdelay`	Gets the ADR Acknowledgment Delay - Refer LORa Regional parameters spec for more information on default value
`adracklimit`	Gets the ADR Acknowledgment Limit - Refer LORa Regional parameters spec for more information on default value
`ar`	Gets the state of the automatic reply.
`ch`	Gets parameters related information which pertains to channel operation and behaviors.
`dr`	Gets the data rate to be used for the next transmissions.
`lbt`	Gets Lorawan Listen Before Talk parameters
`pwridx`	Gets the output power index value.
`sync`	Gets the synchronization word for the LoRaWAN communication.
`upctr`	Gets the value of the uplink frame counter that will be used for the next uplink transmission.
`dnctr`	Gets the value of the downlink frame counter that will be used for the next downlink reception.

`rxdelay1`	Gets the interval value stored for `rxdelay1`.
`rxdelay2`	Gets the interval value stored for `rxdelay2`
`acktimeout`	Gets Acknowledgment Timeout - Refer LORa Regional parameters spec for more information on default value
`classsupport`	Gets the Device classes supported in current implementation
`ismband`	Gets the ISM band in which End device is operating
`availableband`	Gets the ISM bands enabled at compile time
`joinacceptdelay1`	Gets JoinAcceptWindow1 Delay - Delay between the transmission of Join req message and first reception window. Refer LORa Regional parameters spec for more information on default value
`joinacceptdelay2`	Gets JoinAcceptWindow2 Delay - Delay between the transmission of Join req message and second reception window. Refer LORa Regional parameters spec for more information on default value
`status`	Gets the current status of the BIT48LR module.
`rx2`	Gets the data rate and frequency used for the second Receive window.
`retxcnf`	Gets the number of retransmissions to be used for an uplink confirmed packet.
`retxuncnf`	Gets the number of retransmissions to be used for an uplink unconfirmed packet.
`linkchk`	Gets the Period at which Link req command will be sent to Server by End device
`linkchkgwcnt`	Gets the number of gateways that successfully received the last Link Check Request frame.
`linkchkmargin`	Gets the demodulation margin as received in the last Link Check Answer frame.
`maxfcntgap`	Gets Maximum Frame Counter Gap - Refer LORa Regional parameters spec for more information on default value
`maxaggdcycle`	Gets the Aggregated Dutycycle sent by the Server to the end device by MAC Command

2.4.7.1 `mac get deveui`

Response: 8-byte hexadecimal number representing the device EUI.
This command returns the globally unique end-device identifier, as set in the module.
Default: pre-programmed EUI node address
Example: **`mac get deveui`**

Note: After the `mac reset <band>` command is explicitly called, the device EUI value will be set to all zeros. Make certain that a valid value is given to the device EUI.

2.4.7.2 `mac get appeui`

Response: 8-byte hexadecimal number representing the application EUI.
This command will return the application identifier for the module. The application identifier is a value given to the device by the network.
Default: `0000000000000000`
Example: **`mac get appeui`**

2.4.7.3 `mac get devaddr`

Response: 4-byte hexadecimal number representing the device address, from 00000000 to FFFFFFFF.
This command will return the current end-device address of the module.
Default: `00000000`
Example: **`mac get devaddr`**

2.4.7.4 `mac get appkey`

Response: 16-byte hexadecimal number representing the application key.
This command will return the current application key of the module.
Default: `00000000000000000000000000000000`

Example: **`mac get appkey`**

2.4.7.5 `mac get nwkskey`

Response: 16-byte hexadecimal number representing the network session key.
This command will return the current network session key of the module. This key is 16 bytes in length, and provides security for communication between the module and network server.
Default: `00000000000000000000000000000000`
Example: **`mac get nwkskey`**

2.4.7.6 `mac get appskey`

Response: 16-byte hexadecimal number representing the application session key.
This command will return the current application session key of the module. This key is 16 bytes in length, and provides security for communication between the module and application server.
Default: `00000000000000000000000000000000`
Example: **`mac get  appskey`**

2.4.7.7 `mac get class`

Response: A single letter `A` or `C`
This command will return the LoRaWAN operation class as set in the module.

Default: `A`
Example: **`mac get class`**

2.4.7.8 `mac get mcast`

Response: string representing the Multicast state of the module, either `on` or `off`
This command will return the Multicast state as set in the module.
Default: `off`
Example: **`mac get mcast`**

2.4.7.9 `mac get mcastappskey`

Response: 16-byte hexadecimal number representing the multicast application session key.
This command will return the current multicast application session key used when the network sends a multicast message from an application.
Default: `00000000000000000000000000000000`
Example: **`mac get  mcastappskey`**

2.4.7.10 `mac get mcastdevaddr`

Response: 4-byte hexadecimal number representing the device multicast address, from 00000000 to FFFFFFFF
This command will return the current multicast end-device address of the module.
Default: `FFFFFFFF`
Example: **`mac get mcastdevaddr`**

2.4.7.11 `mac get mcastdnctr`

Response: decimal number representing the value of the downlink frame counter that will be used for the next multilink downlink reception, from 0 to 65535
This command will return the value of the downlink frame counter that will be used for the next downlink reception.
Default: `0`
Example: **`mac get mcastdnctr`**

2.4.7.12 `mac get mcastnwkskey`

Response: 16-byte hexadecimal number representing the multicast network session key.
This command will return the current multicast network session key used when the network sends a multicast message from an application.
Default: `00000000000000000000000000000000`
Example: **`mac get  mcastnwkskey`**

2.4.7.13 `mac get adr`

Response:	string representing the state of the adaptive data rate mechanism, either `on` or `off`.

This command will return the state of the adaptive data rate mechanism. It will reflect if the ADR is `on` or `off` on the requested device.

Default:	`off`
Example:	**`mac get adr`**

2.4.7.14 `mac get adrackdelay`

Response:	decimal number representing the current adaptative ack delay.

This command will return the current current adaptative ack delay.

Default:	`32`
Example:	**`mac get adrackdelay`**

2.4.7.15 `mac get adracklimit`

Response:	decimal number representing the current adaptative ack limit.

This command will return the current current adaptative ack limit.

Default:	`64`
Example:	**`mac get adracklimit`**

2.4.7.16 `mac get ar`

Response:	string representing the state of the automatic reply, either `on` or `off`.

This command will return the current state for the automatic reply (AR) parameter. The response will indicate if the AR is on or off.

Default:	`off`
Example:	**`mac get ar`**

2.4.7.17 MAC GET CHANNEL COMMANDS

TABLE 2-9: MAC GET CHANNEL COMMANDS

Parameter	Description
`freq`	Gets the module operation frequency on a given channel ID.
`dcycle`	Gets the module operation duty cycle on a given channel ID.
`drrange`	Gets the module allowed data rate range (min.- max.) allowed on a given channel ID.
`status`	Gets the status for the specified channel ID to indicate if it is enabled for use.

TABLE 2-10: DEFAULT PARAMETERS FOR CHANNELS

Channel Number	Parameters	Frequency band	
		868	433

Channel 0	Frequency (Hz)	868100000	433175000
	Duty cycle **(1)**	302	302
	Data rate range	0-5	0-5
	Status	On	On
Channel 1	Frequency (Hz)	868300000	433375000
	Duty cycle **(1)**	302	302
	Data rate range	0-5	0-5
	Status	On	On
Channel 2	Frequency (Hz)	868500000	433575000
	Duty cycle **(1)**	302	302
	Data rate range	0-5	0-5
	Status	On	On
Channel 3-15	Frequency (Hz)	0	0
	Duty cycle **(1)**	65535	65535
	Data rate range	15 15	15 15
	Status	Off	Off

Note 1: The default settings consider only the three default channels (0-2), and their default duty cycle is 0.33%. If a new channel is created either by the server or by the user, all the channels (including the default ones) must be updated by the user in terms of duty cycle to comply with the ETSI regulations.

2.4.7.17.1 `mac get ch freq <channelID>`

`<channelID>`: decimal number representing the channel number, from 0 to 15.
Response: decimal number representing the frequency of the channel, from 863000000 to 870000000 or from 433050000 to 434790000, in Hz, depending on the frequency band selected.

This command returns the frequency on the requested `<channelID>`, entered in decimal form.

Default: see TABLE 2-10
Example: **`mac get ch freq 0`**

2.4.7.17.2 `mac get ch dcycle <channelID>`

`<channelID>`: decimal number representing the channel number, from 0 to 15.
Response: decimal number representing the duty cycle of the channel, from 0 to 65535.

This command returns the duty cycle on the requested `<channelID>`. The duty cycle is returned in decimal value. The actual duty cycle (in percentage) can be obtained using the returned value V as: percent = 100/(V + 1).

Default:	see TABLE 2-10	
Example:	**`mac get ch dcycle 0`**	// Reads back duty cycle setting on // Channel ID 0. If the value reported // back is 99, the actual duty cycle on the // channel (in percentage) is 100/(99 + 1) = 1.

2.4.7.17.3 **`mac get ch drrange <channelID>`**

`<channelID>`:	decimal number representing the channel number, from 0 to 15.
Response:	decimal number representing the minimum data rate of the channel, from 0 to 7 and a decimal number representing the maximum data rate of the channel, from 0 to 7

This command returns the allowed data rate index range on the requested `<channelID>`, entered in decimal form. The `<minRate>` and `<maxRate>` index values are returned in decimal form and reflect index values. For the description of data rates and the corresponding spreading factors, refer to the *LoRaWAN™ Specification V1.1*.

Default:	see TABLE 2-10
Example:	**`mac get ch drrange 0`**

2.4.7.17.4 **`mac get ch status <channelID>`**

`<channelID>`:	decimal number representing the channel number, from 0 to 15.
Response:	string representing the state of the channel, either `on` or `off`.

This command returns if `<channelID>` is currently enabled for use. `<channelID>` is entered in decimal form and the response will be `on` or `off` reflecting the channel is enabled or disabled appropriately.

Default:	see TABLE 2-10
Example:	**`mac get ch status 2`**

2.4.7.18 **`mac get dr`**

Response:	decimal number representing the current data rate.

This command will return the current data rate.

Default:	5
Example:	**`mac get dr`**

2.4.7.19 **`mac get lbt`**

Response:

- decimal number representing the scan period, from 0 to 65536; units are ms
- decimal number representing the treshold, from -150 to 10; units are dBm
- decimal number representing the max Retry, from 0 and 65536
- decimal number representing the number of samples, from 0 and 255
- string representing the state of the Listen Before Talk mechanism, either `on` or `off`.

This command gets the Listen Before Talk mechanism parameters. For the description of LBT, refer to the *LoRaWAN™ Specification V1.1*.

Default:	`0 0 0 0 off`

Example: **`mac get lbt`**

2.4.7.20 `mac get pwridx`

Response: decimal number representing the current output power index value, from 0 to 9.

This command returns the current output power index value.

Default: `3` for `EU868`, `2` for `NA915`, `2` for `AU915`, `3` for `AS923`, `3` for `JPN923`, `4` for `KR920`, `0` for `IND865`

Example: **`mac get pwridx`**

2.4.7.21 `mac get sync`

Response: one byte long hexadecimal number representing the synchronization word for the LoRaWAN communication.

This command will return the synchronization word for the LoRaWAN communication.

Default: `34`

Example: **`mac get sync`**

2.4.7.22 `mac get upctr`

Response: decimal number representing the value of the uplink frame counter that will be used for the next uplink transmission, from 0 to 4294967295.

This command will return the value of the uplink frame counter that will be used for the next uplink transmission.

Default: `0`

Example: **`mac get upctr`**

2.4.7.23 `mac get dnctr`

Response: decimal number representing the value of the downlink frame counter that will be used for the next downlink reception, from 0 to 4294967295.

This command will return the value of the downlink frame counter that will be used for the next downlink reception.

Default: `0`

Example: **`mac get dnctr`**

2.4.7.24 `mac get rxdelay1`

Response: decimal number representing the interval, in milliseconds, for `rxdelay1`, from 0 to 65535.

This command will return the interval, in milliseconds, for `rxdelay1`.

Default: `1000`

Example: **`mac get rxdelay1`**

2.4.7.25 `mac get rxdelay2`

Response: decimal number representing the interval, in milliseconds, for `rxdelay2`, from 0 to 65535.
This command will return the interval, in milliseconds, for `rxdelay2`.
Default: `2000`
Example: **`mac get rxdelay2`**

2.4.7.26 `mac get acktimeout`

Response: decimal number representing the acktimeout, in milliseconds from 0 to 255.
This command will return the acktimeout, in milliseconds.
Default: `208`
Example: **`mac get acktimeout`**

2.4.7.27 `mac get classsupport`

Response: A single letter `A` or `C`
This command will return the Device classes supported in current implementation.
Default: `A`
Example: **`mac get classsupport`**

2.4.7.28 `mac get ismband`

Response: string representing the frequency band; Value can be:
`EU868, NA915, AU915, AS923, JPN923, KR920, IND865`
This command will return the ISM band in which End device is operating
Default: `EU868`
Example: **`mac get ismband`**

2.4.7.29 `mac get availableband`

Response: string representing the frequency bands enabled at compile time; Value can be:
`EU868, NA915, AU915, AS923, JPN923, KR920, IND865`
This command will return the ISM bands enabled at compile time.
Default: `EU868, NA915, AU915, AS923, JPN923, KR920, IND865`
Example: **`mac get availableband`**

2.4.7.30 `mac get joinacceptdelay1`

Response: decimal number representing the value of the JoinAcceptWindow1 Delay, from 0 to 65535.
This command will return the value of the JoinAcceptWindow1 Delay - Delay between the transmission of Join req message and first reception window. Refer LORa Regional parameters spec for more information on default value.
Default: `5000`

Example: `mac get joinacceptdelay1`

2.4.7.31 `mac get joinacceptdelay2`

Response: decimal number representing the value of the JoinAcceptWindow2 Delay, from 0 to 65535.

This command will return the value of the JoinAcceptWindow2 Delay - Delay between the transmission of Join req message and first reception window. Refer LORa Regional parameters spec for more information on default value.

Default: `6000`

Example: `mac get joinacceptdelay2`

2.4.7.32 `mac get rx2`

Response: decimal number representing the data rate configured for the second Receive window, from 0 to 7 and a decimal number for the frequency configured for the second Receive window, from 863000000 to 870000000 in Hz.

This command will return the current data rate and frequency configured to be used during the second Receive window.

Default: `0 869525000` // for 868 band

Example: `mac get rx2`

2.4.7.33 `mac get status`

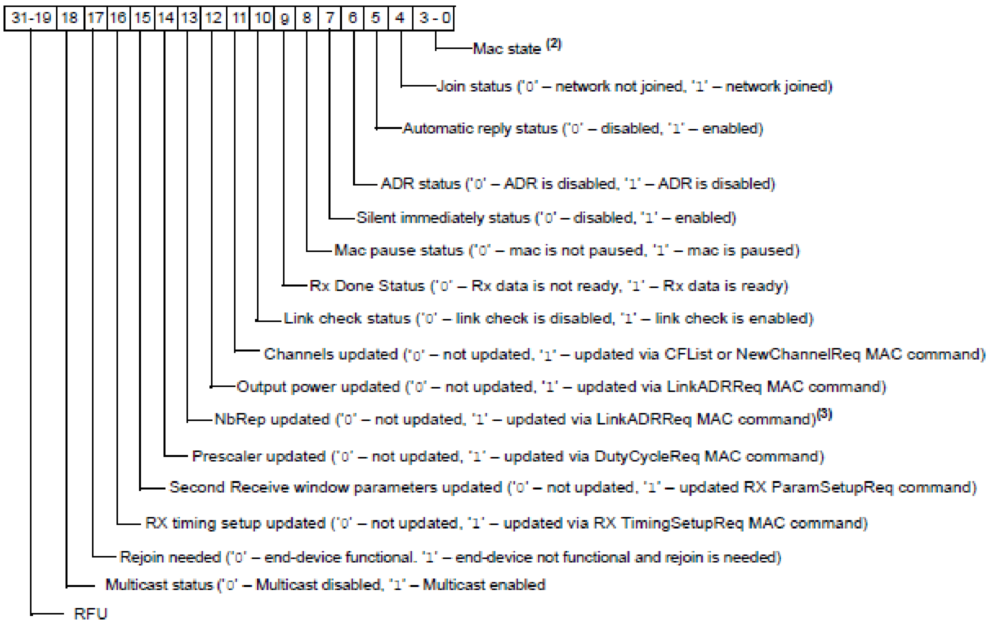
Response: 4-byte hexadecimal number representing the current status of the module.

This command will return the current status of the module. The value returned is a bit mask represented in hexadecimal form. For the significance of the bit mask, refer to FIGURE 2-1: MAC STATUS BIT-MAPPED REGISTER.

Default: `00000000`

Example: `mac get status`

***FIGURE 2-1: MAC STATUS BIT-MAPPED REGISTER** (1)*


31-19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 - 0
Mac state (2)
Join status ('0' – network not joined, '1' – network joined)
Automatic reply status ('0' – disabled, '1' – enabled)
ADR status ('0' – ADR is disabled, '1' – ADR is disabled)
Silent immediately status ('0' – disabled, '1' – enabled)
Mac pause status ('0' – mac is not paused, '1' – mac is paused)
Rx Done Status ('0' – Rx data is not ready, '1' – Rx data is ready)
Link check status ('0' – link check is disabled, '1' – link check is enabled)
Channels updated ('0' – not updated, '1' – updated via CFList or NewChannelReq MAC command)
Output power updated ('0' – not updated, '1' – updated via LinkADRReq MAC command)
NbRep updated ('0' – not updated, '1' – updated via LinkADRReq MAC command)(3)
Prescaler updated ('0' – not updated, '1' – updated via DutyCycleReq MAC command)
Second Receive window parameters updated ('0' – not updated, '1' – updated RX ParamSetupReq command)
RX timing setup updated ('0' – not updated, '1' – updated via RX TimingSetupReq MAC command)
Rejoin needed ('0' – end-device functional, '1' – end-device not functional and rejoin is needed)
Multicast status ('0' – Multicast disabled, '1' – Multicast enabled
RFU

Note 1: Bits 11 (Channels updated), 12 (Output power updated), 13 (NbRep updated), 14 (Prescaler updated), 15 (Second Receive window parameters updated) and 16 (RX timing setup updated) are cleared after issuing a "`mac get status`" command.

2: Mac state:

0 – Idle (transmissions are possible)

1 – Transmission occurring

2 – Before the opening of Receive window 1

3 – Receive window 1 is open

4 – Between Receive window 1 and Receive window 2

5 – Receive window 2 is open

6 – Retransmission delay - used for ADR_ACK delay, FSK can occur

7 – APB_delay

8 – Class C RX2 1 open

9 – Class C RX2 2 open

3: NbRep is the number of repetitions for unconfirmed packets (for more details, refer to the *LoRaWAN™ Specification* V1.0.2).

2.4.7.34 `mac get retxcnf`

Response: decimal number representing the number of retransmissions for an uplink confirmed packet, from 0 to 255.

This command will return the number of retransmissions to be used for an uplink confirmed packet, if no downlink acknowledgment is received from the server.

Default: 7

Example: **`mac get retxcnf`**

2.4.7.35 `mac get retxuncnf`

Response: decimal number representing the number of retransmissions for an uplink unconfirmed packet, from 0 to 255.

This command will return the number of retransmissions to be used for an uplink unconfirmed packet, if no downlink acknowledgment is received from the server.

Default: `0`

Example: **`mac get retxuncnf`**

2.4.7.36 `mac get linkchk`

Response: decimal number representing the number of retransmissions for an uplink unconfirmed packet, from 0 to 65535.

This command will return the Period at which Link req command will be sent to Server by End device.

Default: `0`

Example: **`mac get linkchk`**

2.4.7.37 `mac get linkchkgwcnt`

Response: decimal number representing the number of gateways, from 0 to 255.

This command will return the number of gateways that successfully received the last Link Check Request frame command, as received in the last Link Check Answer.

Default: `0`

Example: **`mac get linkchkgwcnt`**

2.4.7.38 `mac get linkchkmargin`

Response: decimal number representing the demodulation margin, from 0 to 255.

This command will return the demodulation margin as received in the last Link Check Answer frame. For the description of the values, refer to the *LoRaWAN™ Specification V1.1*.

Default: `255`

Example: **`mac get linkchkmargin`**

2.4.7.39 `mac get maxfcntgap`

Response: decimal number representing the Maximum Frame Counter Gap, from 0 to 255.

This command will return the he Maximum Frame Counter Gap - Refer LoRa Regional parameters spec for more information

Default: `0`

Example: **`mac get maxfcntgap`**

2.4.7.40 `mac get maxaggdcycle`

Response: decimal number representing the number of retransmissions for an uplink unconfirmed packet, from 0 to 65535.

This command will return the Aggregated Dutycycle sent by the Server to the end device by MAC Command.

Default: `1`

Example: **`mac get maxaggdcycle`**

2.5 RADIO COMMANDS

TABLE 2-11: RADIO COMMANDS

Parameter	Description
`rx`	This command configures the radio to receive simple radio packets according to prior configuration settings.This command configures a simple radio packet transmission according to prior configuration settings.
`tx`	This command configures a simple radio packet transmission according to prior configuration settings.
`cw`	This command will put the module into a Continuous Wave (cw) Transmission for system tuning or certification use.
`rxstop`	This command causes the radio to exit Continuous Receive mode.
`set`	This command allows modification to the radio setting directly. This command allows for the user to change the method of radio operation within module type band limits.
`get`	This command grants the ability to read out radio settings as they are currently configured.

TABLE 2-12: RADIO PARAMETERS AVAILABILITY FOR DIFFERENT OPERATIONS

Command	`radio get`	`radio set`	Availability for LoRa® Modulation	Availability for FSK Modulation
`freq`	√	√	√	√
`pwr`	√	√	√	√
`sf`	√	√	√	—
`mod`	√	√	√	√
`crc`	√	√	√	√
`bw`	√	√	√	—
`iqi`	√	√	√	—
`prlen`	√	√	—	√
`fdev`	√	√	—	√
`cr`	√	√	√	—
`bt`	√	√	—	√
`afcbw`	√	√	—	√
`rxbw`	√	√	—	√
`wdt`	√	√	√	√
`bitrate`	√	√	—	√

sync	√	√	√	√
reg	√	√	√	√
regdump	√	–	√	√
snr	√	–	√	
lbt	√	√	√	√
pktrssi	√	–	√	–

2.5.1 `radio rx <rxWindowSize>`

`<rxWindowSize>`: decimal number representing the number of symbols (for LoRa modulation) or time-out (in milliseconds, for FSK modulation) that the receiver will be opened, from 0 to 65535. Set `<rxWindowSize>` to '`0`' in order to enable the Continuous Reception mode. Continuous Reception mode will be exited once a valid packet is received.

Response: this command may reply with two responses. The first response will be received immediately after entering the command. If the command is valid (`ok` reply received), a second reply will be received after the reception of a packet or after the time-out occurred.

Response after entering the command:

- `ok` – if parameter is valid and the transceiver is configured in Receive mode
- `invalid_param` – if parameter is not valid
- `busy` – if the transceiver is currently busy

Response after the receive process:

- `radio_rx <data>` – if reception was successful, `<data>`: hexadecimal value that was received;
- `radio_err` – if reception was not successful, reception time-out occurred

Example: **`radio rx 0`** // Puts the radio into continuous Receive mode.

Ensure the radio Watchdog Timer time-out is higher than the Receive window size.

Note: After the `mac reset <band>` command is explicitly called, the device EUI value will be set to all zeros. Make certain that a valid value is given to the device EUI.

Note: When transmitting FSK packets, the payload and the 2-byte CRC is whitened by being XORed with a pseudorandom sequence generated by an LFSR with the polynomial X9 + X5 + 1. This process is automatically reverted on reception so that it is transparent to the user.

2.5.2 `radio tx <data> <count>`

`<data>`: hexadecimal value representing the data to be transmitted, from 0 to

	255 bytes for LoRa modulation and from 0 to 64 bytes for FSK modulation.
`<count>`	decimal value representing the count of the data to transmitted multiple times from 0 to 65535 bytes for LoRa modulation and for FSK modulation.

Response: this command may reply with the following responses.

1. The first response is received immediately after entering the command.
2. If the command is valid (`ok` reply received), a second reply `radio_tx_ok` is received as per the `<count>` value denoting the number of effective transmissions. If the count value is '0', a second reply is received one time after the effective transmission, transmission happens one time.
3. This response gives a summary of the transmission. The responses are:
   - `Total packet` (Total packet transmissions initiated)
   - `Sent` (Total packets transmitted successfully)
   - `Channel busy` (Total packet transmission failures).

Response after entering the command:

- `ok` – if parameter is valid and the transceiver is configured in Transmit mode.
- `invalid_param` – if parameter is not valid.
- `busy` – if the transceiver is currently busy.
- `radio_tx_ok` – if transmission was successful and transmission will be repeated until it reaches the count value.
- `radio_err` – if transmission was unsuccessful (interrupted by radio Watchdog Timer time-out).

This command transmits the <data> passed number of times as per the value given in the `count`.

Example:

```
radio tx 55aa55aa55aa 5 // Transmit a packet 5 times
ok
radio_tx_ok
radio_tx_ok
radio_tx_ok
radio_tx_ok
radio_tx_ok
Total packet: 5,Sent: 5,Channel busy: 0
```

Response after the effective transmission:

- `radio_tx_ok` – if transmission was successful.
- `radio_err` – if transmission was unsuccessful. This command transmits the <data> passed.

Note:	In order to meet ETSI regulations in the given frequency bands, the radio has to use either Listen Before Talk (LBT) + Adaptive Frequency Agility (AFA) or duty cycle limitations. By issuing the `radio tx <data>` command the module does not perform LBT before transmission, thus the user has to make sure that duty cycle limits are not violated. For more information on duty cycle limits please check the EN 300 220-2 v2.4.1 standard.

Note: When transmitting FSK packets, the payload and the 2-byte CRC is whitened by being XORed with a pseudorandom sequence generated by an LFSR with the polynomial X9 + X5 + 1. This process is automatically reverted on reception so that it is transparent to the user.

2.5.3 `radio cw <state>`

`<state>`: string representing the state of the Continuous Wave (CW) mode, either `on` or `off`.

Response: `ok` if state is valid
`invalid_param` if state is not valid

This command will enable or disable the CW mode on the module. CW mode allows the user to put the transceiver into Transmission mode to observe the generated signal.
By altering the settings for the radio the user can observe the changes in transmissions.

Example: **`radio cw on`**

Note: Please note that using `radio cw off` resets the module, this command being semantically identical to `sys reset`.

2.5.4 `radio rxstop`

Response: `ok`

This command causes the radio to exit Continuous Receive mode initiated through the `radio rxstop` command.

Example: **`radio rxstop`**

2.5.5 Radio Set Commands

TABLE 2-13: RADIO SET COMMANDS

Parameter	Description
`freq`	Sets the current operation frequency for the radio.
`pwr`	Sets the output power level used by the radio during transmission.
`sf`	Sets the requested spreading factor (SF) to be used during transmission.
`mod`	Sets the module Modulation mode.
`crc`	Sets if a CRC header is to be used.
`bw`	Sets the value used for the radio bandwidth.
`iqi`	Sets if IQ inversion is used.
`prlen`	Sets the preamble length used during transmissions.
`fdev`	Sets the frequency deviation allowed by the end device.
`cr`	Sets the coding rate used by the radio.

bt	Sets the data shaping for frequency shift keying (FSK) modulation type.
afcbw	Sets the value used by the automatic frequency correction bandwidth.
rxbw	Sets the operational receive bandwidth.
wdt	Sets the time-out limit for the radio Watchdog Timer.
bitrate	Sets the frequency shift keying (FSK) bit rate.
sync	Sets the sync word used.
reg	Sets the given value to a chosen radio register
lbt	Sets the Listen Before Talk mechanism

Note: If any of these parameters was previously saved to user EEPROM by issuing the `sys save` command, after modifying its value, the `sys save` command should be called again.

2.5.5.1 `radio set freq <frequency>`

`<frequency>`: decimal representing the frequency, from 410000000 to 525000000 or from 862000000 to 1020000000,, in Hz.
Response: `ok` if the frequency is valid
`invalid_param` if the frequency is not valid
This command changes the communication frequency of the radio transceiver.
Example: **`radio set freq 868000000`**

2.5.5.2 `radio set pwr <pwrOut>`

`<pwrOut>`: signed decimal number representing the transceiver output power. In 434 MHz band values are from -3 to 15; In 868 MHz band values are from 2 to 17 and 20 (18 and 19 are not implemented). Units are dBm.
Response: `ok` if the output power is valid
`invalid_param` if the output power is not valid
This command changes the transceiver output power. It is possible to set the output power above the regulatory limits. This power setting allows some compensation on the cable or transmission line loss. For more details on output power please check the *BIT48LR_LoRa®TechnologyTransceiverModule_ds*.
Example: **`radio set pwr 14`**

2.5.5.3 `radio set sf <spreadingFactor>`

`<spreadingFactor>`: string representing the spreading factor. Parameter values can be: `sf7`, `sf8`, `sf9`, `sf10`, `sf11` or `sf12`.
Response: `ok` if the spreading factor is valid
`invalid_param` if the spreading factor is not valid
This command sets the spreading factor used during transmission.
Example: **`radio set sf sf7`**

2.5.5.4 `radio set mod <mode>`

`<mode>`: string representing the modulation method, either `lora` or `fsk`.
Response: `ok` if the modulation is valid
`invalid_param` if the modulation is not valid

This command changes the modulation method being used by the module. Altering the mode of operation does not affect previously set parameters, variables or registers. FSK mode also allows GFSK transmissions when data shaping is enabled.

Example: **`radio set mod lora`**

2.5.5.5 `radio set crc < crcHeader >`

`<crcHeader>`: string representing the state of the CRC header, either `on` or `off`.
Response: `ok` if the state is valid
`invalid_param` if the state is not valid

This command enables or disables the CRC header for communications.

Example: **`radio set crc on`** // Enables the CRC header.

2.5.5.6 `radio set bw <bandWidth>`

`<bandWidth>`: decimal representing the operating radio bandwidth, in kHz. Parameter values can be: 125, 250, 500.
Response: `ok` if the bandwidth is valid
`invalid_param` if the bandwidth is not valid

This command sets the operating radio bandwidth for LoRa operation.

Example: **`radio set bw 250`** // The operating bandwidth is 250 kHz.

2.5.5.7 `radio set iqi <iqInvert>`

`<iqInvert>`: string representing the state of the invert IQ, either `on` or `off`.
Response: `ok` if the state is valid
`invalid_param` if the state is not valid

This command enables or disables the Invert IQ for communications.

Example: **`radio set iqi on`** // Invert IQ is enabled.

2.5.5.8 `radio set prlen <preamble>`

`<preamble>`: decimal number representing the preamble length, from 0 to 65535.
Response: `ok` if the preamble length is valid
`invalid_param` if the preamble length is not valid

This command sets the preamble length for transmit/receive.

Example: **`radio set prlen 8`** // Preamble length is 8.

2.5.5.9 `radio set fdev <freqDev>`

`<freqDev>`: decimal number representing the frequency deviation, from 0 to 200000.
Response: `ok` if the frequency deviation is valid

`invalid_param` if frequency deviation is not valid

This command sets the frequency deviation during operation.

Example: **`radio set fdev 5000`** // Frequency deviation is 5 kHz.

2.5.5.10 `radio set cr <codingRate>`

`<codingRate>`: string representing the coding rate. Parameter values can be: `4/5`, `4/6`, `4/7`, `4/8`.

Response: `ok` if the coding rate is valid
`invalid_param` if the coding rate is not valid

This command modifies the coding rate currently being used by the radio.

Example: **`radio set cr 4/7`** // The coding rate is set to 4/7.

2.5.5.11 `radio set bt <gfBT>`

`<gfBT>`: string representing the Gaussian baseband data shaping, enabling GFSK modulation. Parameter values can be: `none`, `1.0`, `0.5`, `0.3`.

Response: `ok` if the data shaping is valid
`invalid_param` if the data shaping is not valid

This command modifies the data shaping applied to FSK transmissions. Entering any `<gfBT>` other than `none` will result in a Gaussian Filter BT being applied to transmissions in FSK mode.

Example: **`radio set bt none`** // Data shaping in FSK mode is disabled or null.

2.5.5.12 `radio set afcbw <autoFreqBand>`

`<autoFreqBand>`: float representing the automatic frequency correction, in kHz. Parameter values can be: 250, 125, 62.5, 31.3, 15.6, 7.8, 3.9, 200, 100, 50, 25, 12.5, 6.3, 3.1, 166.7, 83.3, 41.7, 20.8, 10.4, 5.2, 2.6.

Response: `ok` if the automatic frequency correction is valid
`invalid_param` if the automatic frequency correction is not valid

This command modifies the automatic frequency correction bandwidth for receiving/transmitting.

Example: **`radio set afcbw 125`**

2.5.5.13 `radio set rxbw <rxBandwidth>`

`<rxBandwidth>`: float representing the signal bandwidth, in kHz. Parameter values can be: 250, 125, 62.5, 31.3, 15.6, 7.8, 3.9, 200, 100, 50, 25, 12.5, 6.3, 3.1, 166.7, 83.3, 41.7, 20.8, 10.4, 5.2, 2.6.

Response: `ok` if the signal bandwidth is valid
`invalid_param` if signal bandwidth is not valid

This command sets the signal bandwidth when receiving.

Example: **`radio set rxbw 250`** // Signal bandwidth for receiving is 250 kHz.

2.5.5.14 `radio set wdt <watchDog>`

`<watchDog>`:	decimal number representing the time-out length for the Watchdog Timer, from 0 to 4294967295. Set to '`0`' to disable this functionality.
Response:	`ok` if the watchdog time-out is valid
	`invalid_param` if the watchdog time-out is not valid

This command updates the time-out length, in milliseconds, applied to the radio Watchdog Timer. If this functionality is enabled, then the Watchdog Timer is started for every transceiver reception or transmission. The Watchdog Timer is stopped when the operation in progress in finished.

Example: **`radio set wdt 2000`** // The Watchdog Timer is configured // for 2000 ms.

Note: Ensure the value configured for the Watchdog Timer matches the radio configurations. For example, set the `<watchDog>` value to '`0`' in order to disable this functionality during the radio continuous reception.

2.5.5.15 `radio set bitrate <fskBitRate>`

`<fskBitRate>`:	decimal number representing the FSK bit rate value, from 1 to 300000.
Response:	`ok` if the bit rate value is valid
	`invalid_param` if the bit rate value is not valid

This command sets the FSK bit rate value.

Example: **`radio set bitrate 5000`** // FSK bit rate is set to 5 kb/s.

2.5.5.16 `radio set sync <syncWord>`

`<syncWord>`:	hexadecimal value representing the Sync word used during communication. For LoRa modulation one byte is used, for FSK up to eight bytes can be entered.
Response:	`ok` if the sync word is valid
	`invalid_param` if the sync word is not valid

This command configures the sync word used during communication.

Example: **`radio set sync 12`** // Set the sync word to a single byte // with the value 0x12.

2.5.5.17 `radio set reg <regAddr> <regValue>`

`<regAddr>`:	hexadecimal value representing the address of radio register.
`<regValue>`:	hexadecimal value representing the value to be written to `regAddr`.
Response:	`ok` if the parameters are valid
	`invalid_param` if the `regaddr` & `regvalue` is not valid

This command writes the given value to a chosen radio register.

Example: **`radio set reg 02 05`** //Sets the value `0x05` to a radio register `0x02`.

2.5.5.18 radio set lbt <scanPer> <trshld> <numSamples> <enable>

`<scanPer>`	decimal number representing the scan period, from 0 to 65536
`<trshld>`	decimal number representing the treshold, from -150 to 10
`<numSamples>`	decimal number representing the number of samples, from 0 and 255
`<enable>`	string representing the state of the Listen Before Talk mechanism, either `on` or `off`.
Response:	`ok` if the parameters are valid invalid_param if the parameters are not valid
Example:	**`radio set lbt 100 -95 30 on`**

This command sets the Listen Before Talk mechanism.

2.5.6 Radio Get Commands

TABLE 2-14: RADIO GET COMMANDS

Parameter	Description
`freq`	Gets the current operation frequency for the radio.
`pwr`	Gets the output power level used by the radio during transmission.
`sf`	Gets the requested spreading factor (SF) to be used during transmission.
`mod`	Gets the module Modulation mode.
`crc`	Gets if a CRC header is to be used.
`bw`	Gets the value used for the radio bandwidth.
`iqi`	Gets if IQ inversion is used.
`prlen`	Gets the preamble length used during transmissions.
`fdev`	Gets the frequency deviation allowed by the end device.
`cr`	Gets the coding rate used by the radio.
`bt`	Gets the data shaping for frequency shift keying (FSK) modulation type.
`afcbw`	Gets the value used by the automatic frequency correction bandwidth.
`rxbw`	Gets the operational receive bandwidth.
`wdt`	Gets the time-out limit for the radio Watchdog Timer.
`bitrate`	Gets the frequency shift keying (FSK) bit rate.
`sync`	Gets the sync word used.
`snr`	Gets the signal-to-noise ratio (SNR) of the last received packet.
`reg`	Gets the value of a chosen radio register
`regdump`	Gets the content of radio registers in the given range of address.
`lbt`	Gets the Listen Before Talk mechanism parameters

`pktrssi`	Gets the RSSI value from the last received frame.

2.5.6.1 `radio get freq`

Response: decimal number representing the frequency, from 137000000 to 175000000 or from 410000000 to 525000000 or from 862000000 to 1020000000, in Hz.

This command reads back the current operation frequency of the module.

Default: `868100000`

Example: **`radio get freq`** // Reads back the current frequency the // transceiver communicates on.

2.5.6.2 `radio get pwr`

Response: signed decimal representing the current power level. In 434 MHz band the value is from -3 to 15; In 868 MHz band the value is from 2 to 17 and 20 (18 and 19 are not implemented). Units are dBm.

This command reads back the current power level settings used in operation.

Default: `1`

Example: **`radio get pwr`** // Reads back the current transmit output power.

2.5.6.3 `radio get sf`

Response: string representing the current spreading factor.

This command reads back the current spreading factor being used by the transceiver. Parameter values can be: `sf7`, `sf8`, `sf9`, `sf10`, `sf11`, `sf12`”

Default: `sf7`

Example: **`radio get sf`** // Reads back the current spreading factor settings.

2.5.6.4 `radio get mod`

Response: string representing the current mode of operation of the module, either `lora` or `fsk`.

This command reads back the current mode of operation of the module.

Default: `lora`

Example: **`radio get mod`** // Reads if module is modulating in LoRa or FSK.

2.5.6.5 `radio get crc`

Response: string representing the status of the CRC header, either `on` or `off`

This command reads back the status of the CRC header, to determine if it is to be included during operation.

Default: `on`

Example: **`radio get crc`** // Reads back if the CRC header is enabled for use.

2.5.6.6 `radio get bw`

Response: decimal representing the current operating radio bandwidth, in kHz.

Parameter values can be: 125, 250 or 500.
This command reads back the current operating radio bandwidth used by the transceiver.
Default: `125`
Example: **`radio get bw`** // Reads back the current operational // bandwidth applied to transmissions.

2.5.6.7 `radio get iqi`

Response: string representing the status of the Invert IQ functionality, either `on` or `off`.
This command reads back the status of the Invert IQ functionality.
Default: `off`
Example: **`radio get iqi`** // Reads back the status of the Invert IQ functionality.

2.5.6.8 `radio get prlen`

Response: signed decimal representing the preamble length, from 0 to 65535.
This command reads the current preamble length used for communication.
Default: `8`
Example: **`radio get prlen`** // Reads back the preamble length //used by the transceiver.

2.5.6.9 `radio get fdev`

Response: signed decimal representing the frequency deviation setting, from 0 to 200000.
This command reads frequency deviation setting on the transceiver.
Default: `25000`
Example: **`radio get fdev`** // Reads back current configured frequency // deviation setting.

2.5.6.10 `radio get cr`

Response: string representing the current value settings used for the coding rate. Parameter values can be: `4/5`, `4/6`, `4/7`, `4/8`.
This command reads back the current value settings used for the coding rate during communication.
Default: `4/5`
Example: **`radio get cr`** // Reads back the current coding rate transceiver settings.

2.5.6.11 `radio get bt`

Response: string representing the configuration for data shaping. Parameter values can be: `none`, `1.0`, `0.5`, `0.3`.
This command reads back the current configuration for data shaping applied to FSK transmissions.
Default: `0.5`
Example: **`radio get bt`** // Reads the current data shaping FSK configuration.

2.5.6.12 `radio get afcbw`

Response: float representing the automatic frequency correction band, in kHz. Parameter values can be: 250, 125, 62.5, 31.3, 15.6, 7.8, 3.9, 200, 100, 50, 25, 12.5, 6.3, 3.1, 166.7, 83.3, 41.7, 20.8, 10.4, 5.2, 2.6.

This command reads back the status of the Automatic Frequency Correction Bandwidth.

Default: `41.7`

Example: **`radio get afcbw`** // Reads back the current automatic // frequency correction bandwidth.

2.5.6.13 `radio get rxbw`

Response: float representing the signal bandwidth, in kHz. Parameter values can be: 250, 125, 62.5, 31.3, 15.6, 7.8, 3.9, 200, 100, 50, 25, 12.5, 6.3, 3.1, 166.7, 83.3, 41.7, 20.8, 10.4, 5.2, 2.6.

This command reads back the signal bandwidth used for receiving.

Default: `25`

Example: **`radio get rxbw`** // Reads back the receive signal bandwidth.

2.5.6.14 `radio get wdt`

Response: decimal number representing the length used for the watchdog time-out, from 0 to 4294967295.

This command reads back, in milliseconds, the length used for the watchdog time-out.

Default: `15000`

Example: **`radio get wdt`** // Reads back the current time-out value // applied to the Watchdog Timer

2.5.6.15 `radio get bitrate`

Response: signed decimal representing the configured bit rate, from 1 to 300000.

This command reads back the configured bit rate for FSK communications.

Default: `50000`

Example: **`radio get bitrate`** // Reads back the current FSK bit rate setting.

2.5.6.16 `radio get sync`

Response: hexadecimal number representing the synchronization word used for radio communication.

This command reads back the configured synchronization word used for radio communication. One byte long synchronization word is used for the LoRa modulation while up to eight bytes can be entered for FSK.

Default: `34`

Example: **`radio get sync`**

2.5.6.17 `radio get snr`

Response:	signed decimal number representing the signal-to-noise ratio (SNR), from -128 to 127.

This command reads back the SNR for the last received packet.

Default:	`-128`	
Example:	**`radio get snr`**	// Reads back the measured SNR for the // previously packet reception.

2.5.6.18 `radio get reg <regAddr>`

Response:	hexadecimal value representing the value of radio register at given address .

This command returns the data from the particular radio register

Example:	**`radio get reg 10`**	// Reads back content of the radio register // which address is 10.

2.5.6.19 `radio get regdump <fromAddr> <toAddr>`

Response:	hexadecimal values representing the content of radio registers in the given range of address.

This command returns the content from the particular radio register address range

Example:	**`radio get reg 10 14`**	// Reads back content of the radio registers // from address 10 to 14.

2.5.6.20 `radio get lbt`

Response:	decimal number representing the scan period, from 0 to 65536 decimal number representing the treshold, from -150 to 10 decimal number representing the number of samples, from 0 and 255 string representing the state of the Listen Before Talk mechanism, either `on` or `off`.

This command gets the Listen Before Talk mechanism parameters.

Default:	`0 0 0 off`
Example:	**`radio get lbt`**

2.5.6.21 `radio get pktrssi`

Response:	decimal representing the rssi for the last received frame.

This command reads back the radio Received Signal Strength Indication (rssi) value for the last received frame.

Default:	`-128`	
Example:	**`radio get pktrssi`**	// Reads back the radio rssi.

3 *Revision list*

Revision	Data	Description
0.9.j	2021-03-15	• Changed `sys sleep` command
0.9.d	2020-11-13	• `radio_rx <data>` hex format bug solved
0.9.c	2019-07-22	• `mac set pwridx` and `mac get pwridx` commands changed
0.9.b	2019-07-19	• Dual mode (LoRa mode and FSK radiomodem) Implemented
0.1.c	2019-07-08	• First Stable FW version
0.0.a	2018-12-13	• First Release

Bit is **I**talian **T**echnology S.r.l.
Viale delle Industrie, 20 – 20020 Arese (MI)
Tel.: (+39) 02 30465311, fax: (+39) 0230465396
http://www.bit.it/ **info@bit.it**