API VEGA-LoRa Rev23 (18-January-2019)

Total list of used commands

Possible server responses for invalid input packets

If received packet is not JSON:

```
{
        "err_string":"invalid_json_msg"
}
```

If received packet is JSON, but not contain "cmd" value:

```
{
        "err_string":"cmd_not_found",
        string                                  //received message
}
```

If received packet is JSON, contains "cmd" value, but it's value is unknown:

```
{
        "cmd":string,                                   // "cmd" value from received packet
        "err_string":"unknown_cmd"
}
```

User authorization (auth_req, auth_resp)

Request message:
```
{
        "cmd": "auth_req",
        "login": string,                // case insensitive string
        "password": string              // original password string without any encoding
}
```

Response message:
```
{
        "cmd": "auth_resp",
        "status": boolean,
        "err_string"?: string           //[optional exist if "status" is false] – string code of error
        "token"?: string,               //[optional exist if "status" is true] – session string token (32 HEX symbols)
        "device_access"?: string,       //[optional exist if "status" is true] – access level to devices (see below possible
                                        values)
        "consoleEnable": bool,          //[optional exist if "status" is true] – enable to connect to console subchart with
                                        debug information
        "command_list"?:                //[optional exist if "status" is true] – accessible commands[1] (see below possible
                                        values)
        [
                "command_1",
                …,
                "command_n"
        ],
        "rx_settings"?:                 //[ optional exist if "status" is true] – setting of online receiving messages
        {
                "unsolicited": boolean,//online message is sending [true] or not sending [false - default]
                "direction": string,    //[optional absent if "unsolicited" is false] – direction of possible online messages
                                        (see below)
                "withMacCommands":boolean//[optional] – online message contains MAC commands
        }
}
```

Possible values for "token":
- "FULL" – user have access to all devices;
- "SELECTED" – user have access to device that was selected by administrator.

Possible values for "err_string":
- "invalid_login_or_password" – returns if login isn't found or password isn't validated

Possible string values of "direction":
- "UPLINK" – data from device to server;
- "DOWNLINK" – data from server to device;
- "ALL" – all data from and to device.

[1] This mean that, for example, commands "auth_req" and "auth_resp" are grouped into "auth" command group;

Possible values for "command_list"[2]:

- "get_users"
- "manage_users"
- "delete_users"
- "get_device_appdata"
- "get_ data"
- "send_data"
- "manage_device_appdata"
- "delete_device_appdata"
- "get_gateways"
- "manage_gateways"
- "delete_gateways"
- "get_devices"
- "manage_devices"
- "delete_devices"
- "get_coverage_map"
- "get_device_downlink_queue"
- "manage_device_downlink_queue"
- "server_info"
- "send_email"
- "tx"

Example request message:

```
{
        "cmd": "auth_req",
        "login": "user",
        "password": "123456"
}
```

Example response message:

```
{
        "cmd": "auth_resp",
        "status": true,
        "token": "ABCDEF0012346578ABCDEF0012346578",
        "device_access": "SELECTED",
        "command_list":
        [
                "get_userlist",
                "update_userlist",
                "delete_userlist"
        ]
}
```

[2] Command group "auth", "close_auth" and "token_auth" are accessible for all users.

Close online session (close_auth_req, close_auth_resp)

Request message:

```
{
        "cmd": "close_auth_req",
        "token": string                        // Session string token (32 HEX symbols)
}
```

Response message:

```
{
        "cmd": "close_auth_resp",
        "status": boolean,
        "err_string"?: string              //[optional exist if "status" is false] – string code of error
}
```

Possible values for "err_string":

- "invalid_token" – returns if "token" is not exist or belong to another connection

Example request message:

```
{
        "cmd": "close_auth_req",
        "token": "ABCDEF0012346578ABCDEF0012346578"
}
```

Example response message:

```
{
        "cmd": "close_auth_resp",
        "status": true
}
```

Recover online session via token[3] (token_auth_req, token_auth_resp)

Request message:

```
{
        "cmd": "token_auth_req",
        "token": string                         // Session string token (32 HEX symbols)
}
```

Response message:

```
{
        "cmd": "token_auth_resp",
        "status": boolean,
        "err_string"?: string,                  //[optional exist if "status" is false] – string code of error
        "token": string                         //[optional exist if "status" is true] – new session string token (32 HEX symbols)
}
```

Possible values for "err_string":

• "invalid_token" – returns if token is not exist

Example request message:

```
{
        "cmd": "token_auth_req",
        "token": "ABCDEF0012346578ABCDEF0012346578"
}
```

Example response message:

```
{
        "cmd": "token_auth_resp",
        "status": true,
        "token": "ABCDEF0012346579"
}
```

[3] Token lifetime is equal 1 minute

Get list of registered users (get_users_req, get_users_resp)

Request message:

```
{
        "cmd": "get_users_req",
        "keyword"?:                    //[optional] See below description
        [
                string, …
        ]
}
```

Possible string values of "keyword":
• "no_command_and_devEui" – return list of user without "devEui_list" and "command_list"

Response message:

```
{
        "cmd": "get_users_resp",
        "status": boolean,                    // Main status of execution
        "err_string"?: string,                //[optional exist if "status" is false] – string code of error
        "user_list":
        [
        {
                "login": string,
                "device_access": string,      //Access level to devices (see below possible values). If "FULL",
                                              "devEui_list" would be ignored
                "consoleEnable": bool,        //Enable to connect to console subchart with debug information
                "devEui_list"?:               //[optional absent if "no_command_and_devEui" is exist] List of
                                              DevEUI that is accessible for user
                [
                        "devEui_1",
                        …,
                        "devEui_n"
                ],
                "command_list"?:              //[optional absent if "no_command_and_devEui" is exist] List of
                                              commands that is accessible for user
                [
                        "command_1",
                        …,
                        "command_n"
                ],
                "rx_settings"?:               //[optional absent if "no_command_and_devEui" is exist] – setting of
                                              online receiving messages
                {
                        "unsolicited": boolean,   //online message is sending [true] or not sending [false - default]
                        "direction"?: string,     //[optional absent if "unsolicited" is false] – direction of possible
                                                  online messages (see below)
                        "withMacCommands"?:boolean    //[optional] – online message contains MAC commands
                }
        }, …
        ]
}
```

Possible values for "err_string":
• "inaccessible_command" – returns if current user don't have access for this command;
• "invalid_cmd" – returns if command don't contain "user_list";
• "empty_input_parameter_list" – returns if "user_list" is empty

Possible values for "token":
• "FULL" – user have access to all devices;
• "SELECTED" – user have access to device that was selected by administrator.

Possible string values of "direction":
• "UPLINK" – data from device to server;

- “DOWNLINK” – data from server to device;
- “ALL” – all data from and to device.

Example request message:

```
{
        “cmd”: “get_users_req”
}
```

Example response message:

```
{
        “cmd”: “get_users_resp”,
        “status”: true,
        “user_list”:
        [
        {
                “login”: “user1”,
                “device_access”: “SELECTED”,
                “consoleEnable”: true,
                “devEui_list”:
                [
                        “0000000000000001”
                ],
                “command_list”:
                [
                        “get_userlist”,
                        “update_userlist”,
                        “delete_userlist”
                ]
        }
        ]
}
```

Add new user or modify settings of existed ones (manage_users_req, manage_users_resp)

Request message:

```
{
        "cmd": "manage_users_req",
        "user_list":
        [
        {
                "login": string,                // User login as string
                "password"?: string,            //[optional] – password string. Should be exist when new user is added
                "device_access": string,        //[optional] – access level to devices (see below possible values). If
                                                "FULL", "devEui_list" would be ignored ("SELECTED" - default)
                "consoleEnable": bool,          //[optional] – enable to connect to console subchart with debug
                                                information (false – is default)
                "devEui_list"?:                 //[optional] – list of DevEui that may be accessible by user
                [                               //By default is empty
                        "devEui_1",
                        …,
                        "devEui_n"
                ],
                "command_list"?:                //[optional] – list of commands groups that may be accessible by user
                [                               //By default is empty
                        "command_1",
                        …,
                        "command_n"
                ],
                "rx_settings"?:                 //[optional] – setting of online receiving messages
                {
                        "unsolicited"?: boolean, //[optional] – online message is sending [true] or not sending [false -
                                                default]
                        "direction"?: string,   //[optional] – direction of possible online messages (see below)
                        "withMacCommands"?:boolean    //[optional] – online message contains MAC commands
                }
        }, …
        ]
}
```

Possible values for "token":
- "FULL" – user have access to all devices;
- "SELECTED" – user have access to device that was selected by administrator.

Possible string values of "direction":
- "UPLINK" – data from device to server;
- "DOWNLINK" – data from server to device;
- "ALL" – all data from and to device.

Response message:

```
{
        "cmd": "manage_users_resp",
        "status": boolean,
        "err_string"?: string//[optional exist if "status" is false] – string code of error
        "add_user_list":
        [
        {
                "login":string,
                "status": boolean//User's adding or updating status
        }, …
        ]
}
```

Possible values for "err_string":
- "inaccessible_command" – returns if current user don't have access for this command
- "invalid_cmd" – returns if command don't contain "user_list";

- "empty_input_parameter_list" – returns if "user_list" is empty;
- "login_is_reserved" – was used reserved sequence as user login;

Example request message:

```
{
        "cmd": "manage_users_req",
        "user_list":
        [
        {
                "login": "user1",
                "password": "123456",
                "device_access": "SELECTED",
                "consoleEnable": false,
                "command_list":
                [
                        "get_userlist",
                        "update_userlist",
                        "delete_userlist"
                ],
                "devEui_list":
                [
                        "0000000000000001"
                ],
                "rx_settings":
                {
                        "unsolicited": true,
                        "direction": "ALL",
                        "withMacCommands": true
                }
        }
        ]
}
```

Example response message:

```
{
        "cmd": "manage_users_resp",
        "status": true,
        "add_user_list":
        [
        {
                "login": "user1",
                "status": true
        }
        ]
}
```

Notification that some parameters has been modified (alter_user_resp)

```
{
        “cmd”: “alter_user_resp”,
        “login”: string,                    // User login as string
        “deleted”?: boolean,                //[optional] “True” if user has been deleted
        “device_access”?: string,           //[optional – exist if “deleted” is false] access level to devices. If “FULL”,
                                            “devEui_list” would be ignored (“SELECTED” - default)
        “devEui_list”?:                     //[optional – exist if “device_access” “SELECTED”] – list of DevEui that may be
                                            accessible by user
        [                                   //By default is empty
                “devEui_1”,
                …,
                “devEui_n”
        ],
        “command_list”?:                    //[optional – exist if “deleted” is false] list of commands groups that may be
                                            accessible by user
        [                                   //By default is empty
                “command_1”,
                …,
                “command_n”
        ],
        “consoleEnable”?: boolean,          //[optional] if “true” - console commands would be sent
        “rx_settings”?:                     //[optional – exist if “deleted” is false] setting of online receiving messages
        {
                “unsolicited”: boolean,     //online message is sending [true] or not sending [false - default]
                “direction”: string,        //direction of possible online messages
                “withMacCommands”:boolean   //online message contains MAC commands
        }
}
```

Note: all fields are optional and send (present in packet) in case of modification

<u>Example response message:</u>

```
{
“cmd”: “alter_user_resp”,
“login”: “user1”,
“deleted”: false,
“device_access”: “SELECTED”,
“command_list”:
[
“get_userlist”,
“update_userlist”,
“delete_userlist”
],
“devEui_list”:
[
“0000000000000001”
],
“rx_settings”:
{
“unsolicited”: true,
“direction”: “ALL”,
“withMacCommands”: true
}
}
```

Delete registered users (delete_users_req, delete_users_resp)

Request message:
```
{
        “cmd”: “delete_users_req”,
        “user_list”:
        [
                “login_1”,
                …,
                “login_n”
        ]
}
```

Response message:
```
{
        “cmd”: “delete_users_resp”,
        “status”: boolean,
        “err_string”?: string,          //[optional exist if “status” is false] – string code of error
        “delete_user_list”:
        [
        {
                “login”:string,
                “status”: boolean,
        }, …
        ]
}
```

Possible values for “err_string”:
- “inaccessible_command” – returns if current user don’t have access for this command;
- “invalid_cmd” – returns if command don’t contain “delete_user_list”;
- “empty_input_parameter_list” – returns if “delete_user_list” is empty

---

Example request message:
```
{
        “cmd”: “delete_users_req”,
        “user_list”:
        [
                “user1”,
                “user2”
        ]
}
```

Example response message:
```
{
        “cmd”: “delete_users_resp”,
        “status”: true,
        “delete_user_list”:
        [
        {
                “login”: “user1”,
                “status”: true
        },
        {
                “login”: “user2”,
                “status”: false
        }
        ]
}
```

Verify server connection (ping_req, ping_resp)

Request message:

```
{
        "cmd": "ping_req"
}
```

Response message:

```
{
        "cmd": "ping_resp"
}
```

Get list of devices with attribute set (get_device_appdata_req, get_device_appdata_resp)

Request message:

```
{
        "cmd": "get_device_appdata_req",
        "keyword"?:                      //[optional] See possible values
        [
                string,…
        ]
        "select?:"                       //[optional] Filter object
        {
                "appEui_list"?:          //[optional] List of corresponding AppEUI for request
                [
                        "appEui_1",
                        …,
                        "appEui_n"
                ]
        }
}
```

Possible string values of "keyword":
• "no_attributes" – return list of devEui without attribute sets;
• "add_data_info" – add extra 3 field ("last_data_ts", "fcnt_up" and "fcnt_data") to response.

Response message:

```
{
        "cmd": "get_device_appdata_resp",
        "status": boolean,               //Status of execution of command (global status)
        "err_string"?: string,           //[optional exist if "status" is false] – string code of error
        "devices_list":                  //Sets of attributes
        [
        {
                "devEui": string,        //Device EUI, 16 hex digits (without dashes)
                "appEui": string,        //Application EUI, 16 hex digits (without dashes)
                "devName": string,       //Custom name of device
                "key_name1"?: string, //[optional – is absent if use keyword "no_attributes"] first attribute name
                … ,
                "key_name_n"?: string,           //[optional – is absent if use keyword "no_attributes"] n-st attribute
                                         name
                "last_data_ts"?: integer,        //[optional – is exist if use keyword "add_data_info"] server UTC
                                         timestamp of last received data (ms from Linux epoch)
                "fcnt_up"?: integer,     //[optional – is exist if use keyword "add_data_info"] frame counter upload, a 32-
                                         bit number
                "fcnt_down"?: integer    //[optional – is exist if use keyword "add_data_info"] frame counter download, a
                                         32-bit number
        }, …
        ]
}
```

Possible values for "err_string":
• "inaccessible_command" – returns if current user don't have access for this command;

Example request message:

```
{
        "cmd" : "get_device_appdata_req"
}
```

Example response message:

```
{
        "cmd": "get_device_appdata_resp",
        "status": true,
        "devices_list":
        [
        {
                "devEui": "3933363845366606",
                "appEui": "0000000000000001",
                "devName": "Окно левое в комнате 1",
                "adress1": "Novosibirsk",
                "devType": "SI-11.VA"
                "name": "test"
        },
        {
                "devEui": "3933363845366607",
                "appEui": "0000000000000001",
                "devName": "Окно левое в комнате 2",
                "adress1": "Novosibirsk",
                "devType": "SI-11.VA"
                "name": "test2"
        }
        ]
}
```

Return saved data from device (get_data_req, get_data_resp)

Request message:
```
{
        "cmd": "get_data_req",
        "devEui": string,
        "select"?:                          //[optional] Extra optional for searching
        {
                "date_from"?: integer,      //[optional] server UTC timestamp as number (miliseconds from Linux
                                            epoch)
                "date_to"?: integer,        //[optional] server UTC timestamp as number (miliseconds from Linux
                                            epoch)
                "begin_index"?: integer,    //[optional] begin index of data list [default = 0]
                "limit"?: integer,          //[optional] limit of response data list [default =1000]
                "port"?: integer,           //[optional] select data with noted port
                "direction"?: string,       //[optional] direction of message transition (see below description)
                "withMacCommands"?: boolean//[optional] add MAC commands to response
        }
}
```

Response message:
```
{
        "cmd": "get_data_resp",
        "status": boolean,                  // Status of execution of command (global status)
        "err_string"?: string,              //[optional] If "status" = false, contains error description (see below
                                            description)
        "devEui": string,
        "appEui": string,
        "direction"?: string,               //[optional – exist if "status" = true]
        "totalNum"?: integer,               //[optional – exist if "status" = true] Total existing number of data
                                            corresponding type
        "data_list"?:                       //[optional – exist if "status" = true] Data, transmitted by device
        [
        {
                "ts ": integer,             // Server UTC receiving timestamp (miliseconds from Linux epoch)
                "gatewayId": string,        // Gateway IDs that receive data from device[4]
                "ack": boolean,             // Acknowledgement flag as set by device
                "fcnt": integer,            // Frame counter, a 32-bit number (uplink or downlink based on "direction"
                                            value)
                "port": integer,            // Port (if = 0, use JOIN operations or MAC-commands only)
                "data" : string,            // Decrypted data payload
                "macData"?: string,         //[optional– exist if "withMacCommands" true and MAC command is present]
                                            MAC command data from device
                "freq": integer,            // Radio frequency at which the frame was received/transmitted, in Hz
                "dr": string,               // Spreading factor, bandwidth and coding rate "SF12 BW125 4/5"
                "rssi": integer,            //[optional – exist if packet direction "UPLOAD"] Frame rssi, in dBm, as integer
                                            number
                "snr": float,               //[optional – exist if packet direction "UPLOAD"] Frame snr, in dB
                "type": string,             // Type of packet (see below description). May contains several types joined
                                            via "+"
                "packetStatus"?: string             //[optional – exist if packet direction "UPLOAD"] Status of downlink
                                            message only (see below description)
        }, …
        ]
}
```

[4] Message from one device could be delivered over by several gateways. In this packet, "gatewayId" contains ID of gateways joined via "+". I.e. "0000000000000001+0000000000000002"

Possible string values of "direction":
• "UPLINK" – data from device to server;
• "DOWNLINK" – data from server to device;
• "ALL" – all data from and to device.

Possible string values of "type":
• "UNCONF_UP" – unconfirmed uplink data,
• "REPEATUNCONF_UP" – repeated unconfirmed uplink data,[5]
• "UNCONF_DOWN" – unconfirmed downlink data,
• "CONF_UP" – confirmed uplink data,
• "REPEATCONF_UP" – repeated confirmed uplink data,[5]
• "CONF_DOWN" – confirmed downlink data,
• "JOIN_REQ" – join request message,
• "JOIN_ACC" – join accept message,
• "MAC_LINKCHECK_REQ" – LinkCheckReq MAC command (send by device) [request] [1 Byte length],
• "MAC_LINKCHECK_ANS" – LinkCheckAns MAC command (send by server) [answer] [3 Byte],
• "MAC_ADR_REQ" – LinkADRReq MAC command (send by server) [request] [5 Byte],
• "MAC_ADR_ANS" – LinkADRAns MAC command (send by device) [answer] [2 Byte],
• "MAC_RXPARAM_REQ" – RXParamSetupReq MAC command (send by server) [request] [5 Byte],
• "MAC_RXPARAM_ANS" – RXParamSetupAns MAC command (send by device) [answer] [2 Byte],
• "MAC_STATUS_REQ" – DevStatusReq MAC command (send by server) [request] [1 Byte],
• "MAC_STATUS_ANS" – DevStatusAns MAC command (send by device) [answer] [3 Byte],
• "MAC_NEWCHAN_REQ" – NewChannelReq MAC command (send by server) [request] [6 Byte],
• "MAC_NEWCHAN_ANS" – NewChannelAns MAC command (send by device) [answer] [2 Byte],
• "MAC_RXTIMING_REQ" – RXTimingSetupReq MAC command (send by server) [request] [2 Byte],
• "MAC_RXTIMING_ANS" – RXTimingSetupAns MAC command (send by device) [answer] [1 Byte],
• "MAC_TXPARAM_REQ" – TxParamSetupReq MAC command (send by server) [request] [2 Byte],
• "MAC_TXPARAM_ANS" – TxParamSetupAns MAC command (send by device) [answer] [1 Byte],
• "MAC_DLCHAN_REQ" – DlChannelReq MAC command (send by server) [request] [5 Byte],
• "MAC_DLCHAN_ANS" – DlChannelAns MAC command (send by device) [answer] [2 Byte].

Possible string values of "status" for download message:
• "SUCCESS";
• "TOO_LARGE_GW_PING_ERR" – ping to gateway is too large for transmission;
• "COLLISION_ERR" – packet is collided with another (for device CLASS_A);
• "BEACON_COLLISION_ERR" – current packet is collided with beacon synchro packet;
• "POWER_ERR" – invalid power settings for corresponding base station;
• "FREQ_ERR" – invalid;
• "LATENCY_ERR" – latency for corresponding gateway is too big;
• "NO_VACANT_GW" – no vacant gateway (all gateways is busy);
• "PAYLOAD_SIZE_ERR" – payload is too big for transmission on corresponding SF (may be arise with **send_data_req**).

Possible string values of "err_string":
• "invalidDevEui" – empty or invalid devEui;
• "invalidDirection" – invalid direction value in request;
• "inaccessible_command" – returns if current user don't have access for this command;
• "inaccessible_devEui" – return if current user hasn't access for corresponding device;

[5] "REPEAT" prefix is used in case when device repeat last sending uplink packet. When class C device is used in transparent mode such packets should be ignored.

Example request message:
```
{
        “cmd”: “get_ data_req”,
        “devEui”: “3933363845366606”,
        “select”:
        {
                “date_from”: 354541184,
                “limit”: 100
        }
}
```

Example response message:
```
{
        “cmd”: “get_ data_resp”,
        “status”: true,
        “devEui”: “3933363845366606”,
        “appEui”: “0000000000000001”,
        “data_list”:
        [
        {
                “ts “:6546544313531,
                “gatewayId”: “0000000000000001+0000000000000002”,
                “ack”: false,
                “fcnt”: 10,
                “port”: 40,
                “data” : “3543543545bccb”,
                “macData”: “02”,
                “freq”: 868100000,
                “dr”: “SF12 BW125 4/5”,
                “rssi”: -75,
                “snr”: 2.6,
                “type”: “ UNCONF_UP +MAC_LINKCHECK_ANS”
        }
        ]
}
```

Send data to device (or add data to downlink queue) (send_data_req, send_data_resp)

Request message:

```
{
        “cmd”: “send_data_req”,
        “data_list”:
        [
        {
                “devEui”: string,        // Device EUI, 16 hex digits (without dashes)
                “data”: string,          // Data payload (to be encrypted by server). Should be paired!
                “port”: integer,         // Port to be used (1..223)
                “ack”?: boolean          // [optional] request confirmation (ACK) from end-device
        }, …
        ]
}
```

Response message:

```
{
        “cmd”: “send_data_resp”,
        “status”: boolean,               // Result of execution command [true, false]
        “err_string”?: string,           //[optional] If “status” = false, contains error description (see below
                                         description)
        “append_status”:
        [
        {
                “devEui”: string,
                “status”: boolean        // Result of appending data for corresponding device
        }, …
        ]
}
```

Possible values for “err_string”:

- “inaccessible_command” – returns if current user don’t have access for this command;

Example request message:

```
{
        “cmd”: “send_data_req”,
        “data_list”:
        [
        {
                “devEui”: “3933363845366606”,
                “data”: “25db26a2c8b4”,
                “port”: 40,
                “ack”, true
        },
        {
                “devEui”: “3933363845366658”,
                “data”: “25db26a2c8b5”,
                “port”: 40
        }
        ]
}
```

Example response message:

```
{
        “cmd”: “send_data_resp”,
        “status”: true,
        “append_status”:
        [
        {
                “devEui”: “3933363845366606”,
                “status”: true
        },
        {
                “devEui”: “3933363845366658”,
                “status”: true
        }
        ]
}
```

Manage attributes of corresponding devices (manage_device_appdata_req, manage_device_appdata_resp)

Request message:
```
{
        "cmd": "manage_device_appdata_req",
        "data_list":
        [
        {
                "devEui": string,
                "key_1": string,          // Name of 1-st attribute
                … ,
                "key_n": string           // Name of n-st attribute
        }, …
        ]
}
```

Response message:
```
{
        "cmd": "manage_device_appdata_resp",
        "status": boolean,            // Status of executing command (global status)
        "err_string"?: string,        //[optional] If "status" = false, contains error description (see below
                                      description)
        "update_status"?:             //[optional] Status of updating of corresponding device. Existing only if global
                                      status = "true"
        [
        {
                "devEui": string,
                "status": boolean
        }, …
        ]
}
```

Possible string values of "err_string":
• "inaccessible_command" – returns if current user don't have access for this command;
• "empty_input_parameter_list" – return if "data_list" is empty

---

Example request message:
```
{
        "cmd": "manage_device_appdata_req",
        "data_list":
        [
        {
                "devEui": "3933363845366606",
                "location": "Novosibirsk",
                "name": "test3"
        }
        ]
}
```

Example response message:
```
{
        "cmd": "manage_device_appdata_resp",
        "status": true,
        "update_status":
        [
        {
                "devEui": "3933363845366606",
                    “status”: true
          }
          ]
}
```

Delete attributes of devices (delete_device_appdata_req, delete_device_appdata_resp)

Request message:

```
{
        "cmd": "delete_device_appdata_req",
        "data_list":
        [
        {
                "devEui": string,
                "delete_keys"?:         //[optional] if not exist, would be deleted all attributes for corresponding device
                [
                        "key_1",        // Name of 1-st attribute
                        … ,
                        "key_n"         // Name of n-st attribute
                ]
        }, …
        ]
}
```

Response message:

```
{
        "cmd": "delete_device_appdata_resp",
        "status": boolean,      // Status of executing command (global status)
        "err_string"?: string, //[optional] If "status" = false, contains error description (see below description)
        "delete_status"?:       //[optional] Status of updating of corresponding device. Existing only if global status =
"true"
        [
        {
                "devEui": string,
                "status": boolean
        }, …
        ]
}
```

Possible string values of "err_string":

- "inaccessible_command" – returns if current user don't have access for this command;
- "empty_input_parameter_list" – return if "data_list" is empty

Example request message:
```
{
        “cmd”: “delete_device_appdata_req”,
        “data_list”:
        [
        {
                “devEui”: “3933363845366606”,
                “delete_keys”:
                [
                        “location”,
                        “name”
                ]
        }
        ]
}
```

Example response message:
```
{
        “cmd”: “delete_device_appdata_resp”,
        “status”: true,
        “delete_status”:
        [
        {
                “devEui”: “3933363845366606”,
                “status”: true
        }
        ]
}
```

Get list of registered gateways (get_gateways_req, get_gateways_resp)

Request message:
```
{
        "cmd": "get_gateways_req"
}
```

Response message:
```
{
        "cmd": "get_gateways_resp",
        "status": boolean,
        "err_string"?: string,          //[optional] If "status" = false, contains error description (see below
                                        description)
        "gateway_list":
        [
        {
                "gatewayId": string,    // Gateway ID: 16 hex digits (without dashes)
                "extraInfo": string,    // Any addition information, e.g. location, type and other
                "active": boolean,      // Activity of base station
                "lastOnline"?: integer, //[optional – if gateway is NOT active] UTC time in ms of last keepalive massage
                                        from gateway
                "latency": integer,     // last latency value of gateway [ms]
                "downlinkChannel": integer,     // rfChannel for downlink packets
                "maxPower": integer,    // Maximum power for transitions  [dBm]
                "rxOnly": boolean,      // true if gateway works in receive mode only (two gateway create full-duplex
                                        transceiver )
                "companionGateway"?: string,  //[optional - if rxOnly is true] ID of gateway which can download packets
                "position":
                {
                        "longitude": float,     // geographical longitude of gateway position
                        "latitude": float,      // geographical latitude of gateway position
                        "altitude": integer     // altitude in meters
                }
        }, …
        ]
}
```

Possible string values of "err_string":
"inaccessible_command" – returns if current user don't have access for this command

<u>Example request message:</u>

```
{
        “cmd”: “get_gateways_req”
}
```

<u>Example response message (base stations “024b08050248” and “024b08050249” produce full duplex base station) :</u>

```
{
        “cmd”: “ get_gateways_resp”,
        “gateway_list”:
        [
        {
                “gatewayId”: “024b08050248”,
                “extraInfo”: “Kerlink IoT 868”,
                “active”: true,
                “latency”: 503,
                “downlinkChannel”: 0,
                “maxPower”: 14,
                “rxOnly”: false,
                “position”:
                {
                        “longitude”: 55.08,
                        “latitude”: 82.3,
                        “altitude”: 182
                }
        },
        {
                “gatewayId”: “024b08050249”,
                “extraInfo”: “Kerlink IoT 868”,
                “active”: true,
                “latency”: 25,
                “downlinkChannel”: 0,
                “maxPower”: 14,
                “rxOnly”: true,
                “companionGateway”: “024b08050248”,
                “position”:
                {
                        “longitude”: 55.08,
                        “latitude”: 82.3,
                        “altitude”: 182
                }
        }
        ]
}
```

Add new gateways or modify settings of existed ones (manage_gateways_req, manage_gateways_resp)

Request message:
```
{
        "cmd": "manage_gateways_req",
        "gateway_list":
        [
        {
                "gatewayId": string,            // Gateway ID: 16 hex digits (without dashes)
                "extraInfo"?: string,           //[optional] Any addition custom information, e.g. location, type and other
                "downloadChannel": integer,     // rfChannel for downlink packets (≥0)
                "maxPower"?: integer,           // [optional] Maximum power for transitions (not zero) (default 14 dBm)
                "rxOnly"?: boolean,             //[optional] Receive mode only (two gateway create full-duplex transceiver )
                                                (default false)
                "companionGateway"?: string,    //[optional - if rxOnly is true] ID of gateway which can download packets
                                                16 hex digits
                "position"?:                    //[optional] geographical coordinates
                {
                        "longitude"?: float,    //[optional] geographical longitude of gateway position (default 0)
                        "latitude"?: float,     //[optional] geographical latitude of gateway position (default 0)
                        "altitude"?: integer    //[optional] altitude in meters (default 0)
                }
        }, …
        ]
}
```

Response message:
```
{
        "cmd": "manage_gateways_resp",
        "status": boolean,
        "err_string"?: string,          //[optional] If "status" = false, contains error description (see below
                                        description)
        "gateway_add_status":           // result of command execution
        [
        {
                "gatewayId": string,
                "status": boolean
        }, …
        ]
}
```

Possible string values of "err_string":
- "inaccessible_command" – returns if current user don't have access for this command;
- "empty_input_parameter_list" – return if "gateway_list" is empty

Example request message:
```
{
        “cmd”: “manage_gateways_req”,
        “gateway_list”:
        [
        {
                “gatewayId”: “024b08050248”,
                “extraInfo”: “Kerlink IoT 868”,
                “downloadChannel”: 0,
        },
        {
                “gatewayId”: “024b08050249”,
                “extraInfo”: “Kerlink IoT 868”,
                “downloadChannel”: 0,
                “maxPower”: 27,
                “rxOnly”: true,
                “companionGateway”: “024b08050248”
        }
        ]
}
```

Example response message:
```
{
        “cmd”: “manage_gateways_resp”,
        “status”: true,
        “gateway_add_status”:
        [
        {
                “gatewayId”: “024b08050249”,
                “status”: true
        },
        {
                “gatewayId”: “024b08050249”,
                “status”: true
        },
        ]
}
```

Delete registered gateway (delete_gateways_req, delete_gateway_resp)

Request message:
```
{
        "cmd": "delete_gateways_req",
        "gateway_list":
        [
                "gatewayId _1",         // Gateway ID
                , … ,
                "gatewayId _n"
        ]
}
```

Response message:
```
{
        "cmd": "delete_gateway_resp",
        "status": boolean,              // Result of command execution
        "err_string"?: string,          //[optional] If "status" = false, contains error description (see below
                                        description)
}
```

Possible string values of "err_string":
- "inaccessible_command" – returns if current user don't have access for this command;
- "empty_input_parameter_list" – return if "gateway_list" is empty

---

Example request message:
```
{
        "cmd": "delete_gateways_req",
        "gateway_list":
        [
                "024b08050248"
        ]
}
```

Example response message:
```
{
        "cmd": "delete_gateways_resp",
        "status": true
}
```

Get list of devices with registration info (get_devices_req, get_devices_resp)

Request message:

```
{
        "cmd": "get_devices_req",
        "keyword"?:                    //[optional] keyword: "no_reginfo" – return list of devEui without reginfo sets.
                                       "devEui_list" and "appEui_list" is ignoring.
        [
                string,…
        ]
        "select"?:                     //[optional] filter object
        {
                "devEui_list"?:        //[optional] list of corresponding devEui for request
                [
                        "devEui_1",
                        …,
                        "devEui_n"
                ],
                "appEui_list"?:        //[optional] list of devEui with corresponding appEui
                [
                        "appEui_1",
                        …,
                        "appEui_n"
                ],
        }
}
```

Response message:

```
{
        “cmd”: “get_devices_resp”,
        “status”: boolean,              //status of execution of request
        “err_string”?: string,          //[optional] If “status” = false, contains error description (see below
                                        description)
        “devices_list”:
        [
        {
                “devEui”: string,       // Device EUI, 16 hex digits (without dashes)
                “devName”:string,       // Custom device name
                                        //If device is registered via ABP, response contains “ABP” object, if via OTAA
                                        response contains “OTAA” object. If at registration had been used both sets of
                                        parameters – response would be contain both objects
                “ABP”?:                 //[optional]  Activation By Personalization parameters
                {
                        “devAddress”: integer,  //32-bit device address (should be less then 0x01FFFFFF)
                        “appsKey”: string,      //application session key [contains symbols 0-9a-fA-F]
                        “nwksKey”: string       //network session key [contains symbols 0-9a-fA-F]
                },
                “OTAA”?: //[optional]  Over The Air Activation parameters
                {
                        “appEui”?: string,      //[optional – exist if AppEUI is not empty]  application EUI
                        “appKey”: string,       //application key
                        “last_join_ts”: integer //time of last join procedure [server UTC timestamp (ms from Linux
                                                epoch)]
                },
                “frequencyPlan”:
                {
                        “freq4”: integer,       //frequency for channel 4 in Hz
                        “freq5”: integer,       //frequency for channel 5 in Hz
                        “freq6”: integer,       //frequency for channel 6 in Hz
                        “freq7”: integer,       //frequency for channel 7 in Hz
                        “freq8”: integer        //frequency for channel 8 in Hz
                }
                “channelMask”:          // Masking for frequency of 16 channels
                {
                        “channel1En”: boolean,  //Mask for channel1
                        …,
                        “channel16En”: boolean  //Mask for channel16
                },
                “position”:
                {
                        “longitude”: float,     // geographical longitude of device position
                        “latitude”: float,      // geographical latitude of device position
                        “altitude”: integer     // altitude in meters
                },
                “class”: string,        // device class [“CLASS_A”, ~~“CLASS_B”~~[unsupported], “CLASS_C”]
                “rxWindow”: integer,    // Receive window [1, 2]
                “delayRx1”: integer,    // Delay of start first receive window [1..15], sec
                “delayRx2”: integer,    // Delay of start second receive window [1..15], sec
                “delayJoin1”: integer,  // Delay of start first receive window after joinRequest [1..15], sec
                “delayJoin2”: integer,  // Delay of start second receive window after joinRequest [1..15], sec
                “drRx2”: integer,       // DataRate of second receive window [0..5]
                “freqRx2”: integer,     // Frequency of second receive window, Hz
                “adr”: boolean,         // ADR algorithm is active? [true, false]
                “preferDr”: integer,    // Prefer DR when ADR is enabled [0..5]
                “preferPower”: integer,// Prefer power when ADR is enabled [14,11,8,5,2 dBm]
                “fcnt_up”: integer,     // frame counter upload, a 32-bit number
                “fcnt_down”: integer,   // frame counter download, a 32-bit number
                “last_data_ts”: integer,// time of receive last data [server UTC timestamp (ms from Linux epoch)]
                “lastRssi”: integer,    // RSSI value of last reception
                “lastSnr”: float,       // SNR value of last reception
                “reactionTime”: integer,        // For device CLASS_C only: time between end of RX and begin of possible
                                        TX, msec
                …
```

continued on next page…

```
                    "useDownlinkQueueClassC": boolean,     // For device CLASS_C only: use queue of downlink messages or
                                                  try to transmit online only. If online transmission is failed or device is already busy
                                                  – packet is ignored,
                    "serverAdrEnable": boolean,      // if "adr" (from device) and "serverAdrEnable" (from server for current
                                                  device only) is enabled, server will realize ADR
                    "rawDataStoragePeriod"?: int    //[optional] Period of storage of raw data for device in days [default –
                                                          365 days – 1 year]. The range out data would be deleted to save
                                                          database hard drive space;
          }, …
          ]
}
```

Possible string values of "err_string":

• "inaccessible_command" – returns if current user don't have access for this command

Example request message

```
{
        “cmd”: “get_devices_req”
}
```

Response message:

```
{
        “cmd”: “get_devices_resp”,
        “status”: true,
        “devices_list”:
        [
        {
                “devEui”: “3933363845366606”,
                “devName”: “test device”,
                “ABP”:
                {
                        “devAddress”: 2844301241,                  // соответсвует адресу A98897B9
                        “appsKey”: “C9EDC771CF77B0CAF802FCD867EF46D4”,
                        “nwksKey”: “353E1A29F088F8ACF937D033D5045F0C”
                },
                “OTAA”:
                {
                        “appEui”: “0000000000000015”,
                        “appKey”: “A72D920E7E4A61E967635DEC32E78FBB”,
                        “last_join_ts”: 6541616514
                }
                “frequencyPlan”:
                {
                        “freq4”: 867100000,
                        “freq5”: 867100000,
                        “freq6”: 867100000,
                        “freq7”: 867100000,
                        “freq8”: 867100000
                }
                “channelMask”:
                {
                        “channel1En”: true,
                        …,
                        “channel16En”: true
                }
                “position”:
                {
                        “longitude”: 82.302,
                        “latitude”: 55.03,
                        “altitude”: 182
                }
                “class”: “CLASS_A”,
                “rxWindow”: 1,
                “delayRx1”: 1,
                “delayRx2”: 2,
                “delayJoin1”: 5,
                “delayJoin2”: 6,
                “drRx2”: 0,
                “freqRx2”: 869525000,
                “adr”: false,
                “fcnt_up”: 1005,
                “fcnt_down”: 10,
                “last_data_ts”: 654616464,
                “lastRssi”: -51,
                “lastSnr”: 8.5,
                “useDownlinkQueueClassC”: true,
                “serrverAdrEnable”: true,
                “rawDataStoragePeriod”: 365
        }
        ]
}
```

Add new devices or modify settings of existed ones[6] (manage_devices_req, manage_devices_resp)

Request message:

```
{
        "cmd": "manage_devices_req",
        "devices_list":
        [
        {
                "devEui": string,               //device EUI, 16 HEX digits (without dashes)
                "devName"?:string,              //[optional] Custom device name (by default is empty)
                                                //If need to add device, packet should contain one of "ABP" or "OTAA"
                                                parameters or both (device will work in "ABP" and "OTAA" modes). In
                                                need to update some parameters, packet shouldn't contain any
                                                registration information
                "ABP"?:                         //[optional] Activation By Personalization parameters
                {
                        "devAddress": integer,  //32-bit device address, should be in range [ 0x00000001..0xFFFFFFFF ]
                        "appsKey": string,      //application session key (32 HEX digits)
                        "nwksKey": string       //network session key (32 HEX digits)
                },
                "OTAA"?:                        //[optional] Over The Air Activation parameters
                {
                        "appEui"?: string,      //[optional] application EUI (16 HEX digits)
                        "appKey": string        //application key (32 HEX digits)
                },
                "frequencyPlan"?:               //[optional]
                {
                        "freq4": integer,       //frequency for channel 4 in Hz
                        "freq5": integer,       //frequency for channel 5 in Hz
                        "freq6": integer,       //frequency for channel 6 in Hz
                        "freq7": integer,       //frequency for channel 7 in Hz
                        "freq8": integer        //frequency for channel 8 in Hz
                },
                "channelMask"?:                 //[optional] Masking for frequency of 16 channels (by default all
                                                channels is false)
                {
                        "channel1En": boolean,  //Mask for channel1
                        …,
                        "channel16En": boolean  //Mask for channel16
                },
                "position"?:                    //[optional] geographical coordinates
                {
                        "longitude"?: float,    //[optional] geographical longitude of device position (by default 0)
                        "latitude"?: float,     //[optional] geographical latitude of device position (by default 0)
                        "altitude"?: integer    //[optional] altitude in meters (by default 0)
                },
                "class"?: string,               //[optional] device class ["CLASS_A"(default), ~~"CLASS_B"~~[unsupported],
                                                "CLASS_C"]
                "rxWindow"?: integer,           //[optional] Receive window [1(default), 2]
                "delayRx1"?: integer,           //[optional] Delay of start first receive window [1..15], sec (default 1s)
                "delayJoin1"?: integer,         //[optional] Delay of start first receive window after joinRequest [1..15],
                                                sec (default 1s)
```

[6] Devices of "Vega Absolute" company not decries total counter therefor count of such devices is unlimited. But total counter of devices of other manufacturers is bounded. By default server can register only 1000 devices from other manufacturers. To expand available devices counter contact to "Vega Absolute" company.
To register "Vega Absolute" device enter AppEUI, AppKEY and DevEUI in registration request to server (in this case supported OTAA only)

```
                …

                continued on next page…

                "drRx2"?: integer,              //[optional] DataRate of second receive window [0..5] (by default 0)
                "freqRx2"?: integer,            //[optional] Frequency of second receive window, Hz (by default
                                                869525000 MHz)
                "preferDr"?: integer,           //[optional] Prefer DR when ADR is enabled [0..5] (by default 0)
                "preferPower"?: integer,        //[optional] Prefer power when ADR is enabled [14,10,7,5,2 dBm]
                                                (default 14dBm)
                 "reactionTime"?: integer,      //[optional] Use only for CLASS_C, time between end of receiving
                                                request and begin of possible transition (on device side) [in milliseconds]
                                                (by default 1000msec)
                "useDownlinkQueueClassC"?: boolean,   //[optional] For device CLASS_C only: use queue of downlink
                                                messages or try to transmit online only. If online transmission is failed or
                                                device is already busy – packet is ignored [default – "false"]
                "serverAdrEnable"?: boolean,    // if "adr" (from device) and "serverAdrEnable" (from server for current
                                                device only) is enabled, server will realize ADR [default - true]
                "dataStoragePeriod"?: int       //[optional] Period of storage of raw data for device in days [default –
                                                365 days – 1 year]. The range out data would be deleted to save
                                                database hard drive space;
          }, …
          ]
    }
```

Response message:

```
{
        "cmd": "manage_devices_resp",
        "status": boolean,                  // result of command execution (global)
        "err_string"?: string,              //[optional] If "status" = false, contains error description (see below
                                            description)
        "device_add_status":
        [
        {
                "devEui": string,
                "status": string
        }, …
        ]
}
```

Possible string values of "err_string":
• "inaccessible_command" – returns if current user don't have access for this command

Possible string values of "status" for corresponding device [**before** activation/updating attempt]:
• "invalidDevEui" – invalid size of device EUI;
• "invalidAbpParamList" – invalid mandatory parameters list for "ABP" section;
• "invalidDevAddrValue" – invalid "devAddress" value. It should be less then 0x01FFFFFF and not equal zero;
• "invalidSessionKeyValue" – invaliud size of "appsKey" or/and "nwksKey";
• "invalidOtaaParamList" – invalid mandatory parameters list for "OTAA" section;
• "frequencyPlanIsAbsant" – "frequencyPlan" section is absent with OTAA activation;
• "invalidFrequencyPlan" – invalid mandatory parameters list in section "frequencyPlan";
• "invalidAppKeyValue" – invalid size of "appKey";
• "invalidChannelMaskParamList" – invalid mandatory parameters list in section "channelMask";
• "invalidClass" – invalid value for "class" parameter;
• "unsupportClass" – unsupported class, for example "CLASS_B";
• "invalidRxWindow" – invalid value for "rxWindow" parameter;
• "invalidDataRate" – invalid value for "drRx2" or/and "preferDr" parameters;
• "invalidPower" – invalid value for "preferPower" parameter;
• "invalidDelay" – invalid value for or/and ["delayRx1", "delayRx2", "delayJoin1", "delayJoin2"] parameters.

Possible string values of "status" for corresponding device [**as invalid result** of activation/updating attempt]:
• "noRegisterKeys" – device is not exist yet and packet doesn't contain registration information;
• "repetitionDevAddr" – device with corresponding "devAddress" is already registered on server;
• "abpReginfoAlreadyExist" – ABP activation information for corresponding device is already existed on server;
• "otaaReginfoAlreadyExist" – OTAA activation information for corresponding device is already existed on server;
• "reginfoAlreadyExist" – registration information for corresponding device is already existed on server;
• "maxDevCountReached" – maximum device count limitation has reached.

Possible string values of "status" for corresponding device [**as success result** of activation/updating attempt]:
• "added" – device is not existed on server before and is registered with corresponding registration information;
• "updated" – some parameters are updated for corresponding device;
• "nothingToUpdate" – changes from existing device parameters is not detected;
• "updateViaMacBuffer" – received parameters should be updated via MAC commands and ones couldn't be applied immediately.

Examples request message - response message:

1. Appending of device

```
{
        "cmd": "manage_devices_req",
        "devices_list":
        [
        {
                "devEui": "3933363845366606",
                "ABP":
                {
                        "devAddress": 13548846,
                        "appsKey": "C9EDC771CF77B0CAF802FCD867EF46D4",
                        "nwksKey": "353E1A29F088F8ACF937D033D5045F0C"
                },
                "OTAA":
                {
                        "appKey": "A72D920E7E4A61E967635DEC32E78FBB"
                }
                "frequencyPlan":
                {
                        "freq4": 867100000,
                        "freq5": 867300000,
                        "freq6": 867500000,
                        "freq7": 867700000,
                        "freq8": 867900000
                }

                "class": "CLASS_A",
                "rxWindow": 2
        }
        ]
}

{
        "cmd": "manage_devices_resp",
        "status": true,
        "device_add_status":
        [
        {
                "devEui": "3933363845366606",
                "status": "added"
        }
        ]
}
```

2. Updating parameters (with error)

```
{
        “cmd”: “manage_devices_req”,
        “devices_list”:
        [
        {
                “devEui”: “3933363845366606”,
                “OTAA”:
                {
                        “appKey”: “A72D920E7E4A61E967635DEC32E78FBB”
                }
                “class”: “CLASS_A”,
                “rxWindow”: 2
        }
        ]
}

{
        “cmd”: “manage_devices_resp”,
        “status”: true,
        “device_add_status”:
        [
        {
                “devEui”: “3933363845366606”,
                “status”: “otaaReginfoAlreadyExist”       //packet contain registration information
        }
        ]
}
```

3. Updating parameters (without errors)

```
{
        “cmd”: “manage_devices_req”,
        “devices_list”:
        [
        {
                “devEui”: “3933363845366606”,
                “class”: “CLASS_C”
        }
        ]
}

{
        “cmd”: “manage_devices_resp”,
        “status”: true,
        “device_add_status”:
        [
        {
                “devEui”: “3933363845366606”,
                “status”: “updated”
        }
        ]
}
```

Delete registered device (delete_devices_req, delete_devices_resp)

Request message:
```
{
        "cmd": "delete_devices_req",
        "devices_list":
        [
                "devEui_1",             //device EUI, 16 hex digits (without dashes) for 1-st device
                … ,
                "devEui_n"              // device EUI, 16 hex digits (without dashes) for n-st device
        ]
}
```

Response message:
```
{
        "cmd": "delete_devices_resp",
        "status": boolean,              // result of command execution
        "device_delete_status"?:        // [optional – exist if status "true"] Contains statuses of device removal
        [
        {
                "devEui": string,
                "status": string
        }, …
        ],
        "err_string"?: string,          //[optional] If "status" = false, contains error description (see below
                                        description)
}
```

Possible string values of "status" for corresponding device [**as success result** of removal attempt]:
- "deleted" – device and all data is deleted;
- "notFound" – device is not found;

Possible string values of "err_string":
- "inaccessible_command" – returns if current user don't have access for this command;
- "empty_input_parameter_list" – return if "devices_list" is empty

Example request message:
```
{
        "cmd": "delete_devices_req",
        "devices_list":
        [
                "3933363845366606"
        ]
}
```

Example response message:
```
{
        "cmd": "delete_devices_resp",
        "status": true,
        "device_delete_status":
        [
        {
                "devEui": "3933363845366606",
                "status": "deleted"
        }
        ]
}
```

Information about coverage area (get_coverage_map_req, get_coverage_map_resp)

Request message:

```
{
        "cmd": "get_coverage_map_req",
        "devices_list"?:                //[optional] devEui list for request. If exist, "gateway_list" should not be
                                        existed
        [
                "devEui_1",             //device EUI, 16 hex digits (without dashes) for 1-st device
                … ,
                "devEui_n"              // device EUI, 16 hex digits (without dashes) for n-st device
        ]
        "gateway_list"?:                //[optional] Gateway IDs list for request. If exist, "device_list" should not be
                                        existed
        [
                "gatewayId_1",          // Gateway ID: 16 hex digits (without dashes)
                …,
                "gatewayId_n"           // Gateway ID: 16 hex digits (without dashes)
        ]
}
```

To receive full list grouped by gateways – "devices_list" and "devices_list" should be absent or null or "gateway_list" should be empty.

For example:

```
{
        "cmd": "get_coverage_map_req"
}
or
{
        "cmd": "get_coverage_map_req",
        "gateway_list": []
}
```

To receive full list grouped by devices – "devices_list" should be empty.

For example:

```
{
        "cmd": "get_coverage_map_req",
        "devices_list": []
}
```

Response message:

```
{
        "cmd": "get_coverage_map_resp",
        "status": boolean,                  // result of command execution
        "err_string"?: string,              //[optional] If "status" = false, contains error description (see below
                                            description)
        "gateway_list"?:                    //[optional] default (if "devices_list" and "gateway_list" is absent) or if
                                            "gateway_list" exist
        {
                "gatewayId _1":             //devices for gateway with ID = "gatewayId _1"
                [
                {
                        "devEui": string,   // device identifier
                        "last_rssi": number,     // RSSI of last data packet for this set [device + base station]
                        "last_snr": number,      // SNR of last data packet
                        "last_data_ts": number   // server timestamp as number (milliseconds from Linux epoch)
                }, …
                ],
                …,
                "gatewayId _n":             //devices for gateway with ID = "gatewayId _n"
                [
                {
                        "devEui": string,   // device identifier
                        "last_rssi": number,     // RSSI of last data packet for this set [device + base station]
                        "last_snr": number,      // SNR of last data packet
                        "last_data_ts": number   // server timestamp as number (milliseconds from Linux epoch)
                }, …
                ]
        },
        "devices_list"?:                    //[optional] if "devices_list" exist
        {
                "devEui_1":                 //base stations for device with EUI = "devEui_1"
                [
                {
                        "gatewayId": string,     // Gateway ID
                        "last_rssi": number,     // RSSI of last data packet for this set [device + base station]
                        "last_snr": number,      // SNR of last data packet
                        "last_data_ts": number   // server timestamp as number (milliseconds from Linux epoch)
                },…
                ]
                …,
                "devEui_n":                 // base stations for device with EUI = "devEui_n"
                [
                {
                        "gatewayId": string,     // Gateway ID
                        "last_rssi": number,     // RSSI of last data packet for this set [device + base station]
                        "last_snr": number,      // SNR of last data packet
                        "last_data_ts": number   // server timestamp as number (milliseconds from Linux epoch)
                }, …
                ]
        }
}
```

Possible values for "err_string":

- "inaccessible_command" – returns if current user don't have access for this command;
- "invalid_cmd" – returns if command contains not null "gateway_list" and "devices_list"

Example request message:
```
{
        "cmd": "get_coverage_map_req"
}
```

Example response message:
```
{
        "cmd": "get_coverage_map _resp",
        "status": true,
        "gateway_list":
        {
                "babababababababa":
                [
                {
                        "devEui": "babababababababa00",
                        "last_rssi": -105,
                        "last_snr": -10.2,
                        "last_data_ts": 10546846065
                },
                {
                        "devEui": "babababababababa01",
                        "last_rssi": -56,
                        "last_snr": 8.3,
                        "last_data_ts": 1315438384
                }
                ],
                "024B05FFFF050831":
                [
                {
                        "devEui": "babababababababa01",
                        "last_rssi": -153,
                        "last_snr": -18.6,
                        "last_data_ts": 1315438385
                }
                ]
        }
}
```

Example request message:

```
{
        “cmd”: “get_coverage_map_req”,
        “devices_list”:
        [
                “babababababababa01”
        ]
}
```

Example response message:

```
{
        “cmd”: “get_coverage_map _resp”,
        “status”: true,
        “devices_list”:
        {
                “babababababababa01”:
                [
                {
                        “gatewayId”: “babababababababa”,
                        “last_rssi”: -56,
                        “last_snr”: 8.3,
                        “last_data_ts”: 1315438384
                },
                {
                        “gatewayId”: “024B05FFFF050831”,
                        “last_rssi”: -153,
                        “last_snr”: -18.6,
                        “last_data_ts”: 1315438385
                }
                ]
        }
}
```

Information about queue of downlink packets (get_device_downlink_queue_req, get_device_downlink_queue_resp)

Request message:

```
{
        "cmd": "get_device_downlink_queue_req",
        "select"?:                      //[optional] filter options
        {
                "devices_list"?:        //[optional] list of corresponding devEui for request
                [
                        "devEui_1",
                        …,
                        "devEui_n"
                ]
        }
}
```

Response message:

```
{
        "cmd": "get_device_downlink_queue_resp",
        "status": boolean,
        "err_string"?: string,          //[optional – exist if status is false] contains error string code (see below
                                        description)
        "devices_list"?:                //[optional – exist if status is true] contains list of packets for corresponding
                                        devices
        {
                "devEui_1":
                [
                {
                        "ts": integer,          // Server UTC timestamp (milliseconds from Linux epoch) when packet is
                                                putted to queue
                        "data": string,         // Data payload without encription
                        "port": integer,        // Port (if port = 0 "data" contains MAC payload[7], see LoRaWAN specification)
                        "ack": boolean          // If true, device will send confirmation of receiving this packet
                }, …
                ],
                …,
                "devEui_n":
                [
                {
                        "ts": integer,          // Server UTC timestamp (milliseconds from Linux epoch) when packet is
                                                putted to queue
                        "data": string,         // Data payload without encription
                        "port": integer,        // Port (if port = 0 "data" contains MAC payload, see LoRaWAN specification)
                        "ack": boolean          // If true, device will send confirmation of receiving this packet
                }, …
                ]
        }

}
```

Possible values for "err_string":

- "inaccessible_command" – returns if current user don't have access for this command

[7] Server can itself to create MAC packets and puts it into downlink queue

<u>Example request message:</u>

```
{
        “cmd”: “get_device_downlink_queue_req”
}
```

<u>Example response message:</u>

```
{
        “cmd”: “get_device_downlink_queue_resp”,
        “status”: true,
         “devices_list”:
        {
                “0000000000000001”:
                [
                {
                        “ts”: 1505979552026,
                        “data”: “010203040567”,
                        “port”: 2,
                        “ack”: false
                },
                {
                        “ts”: 1505979553348,
                        “data”: “010203040568”,
                        “port”: 2,
                        “ack”: false
                }
                ],
                 “0000000000000002”:
                [
                {
                        “ts”: 1505979552026,
                        “data”: “0311FF0001”,
                        “port”: 0,                    // MAC data
                        “ack”: false
                }
                ]
        }
}
```

Example request message:

```
{
        “cmd”: “get_device_downlink_queue_req”,
        “select”:
        {
                “devices_list”:
                [
                        “0000000000000002”
                ]
        }
}
```

Example response message:

```
{
        “cmd”: “get_device_downlink_queue_resp”,
        “status”: true,
         “devices_list”:
        {
                “0000000000000002”:
                [
                {
                        “ts”: 1505979552026,
                        “data”: “0311FF0001”,
                        “port”: 0,              // MAC data
                        “ack”: false
                }
                ]
        }
}
```

Modification queue of downlink packets (manage_device_downlink_queue_req, manage_device_downlink_queue_resp)

At now this command lets only to remove some or all packets for corresponding devices.

Request message:

```
{
        "cmd": "manage_device_downlink_queue_req",
         "devices_list":                    // List of corresponding devEui packet parameters for request
        [
        {
                "devEui": string,           // Device identifier
                "action": string,           // Code string for action (see below description)
                "ts_list"?:                 // [optional] – set of timestamp of corresponding packets (see
                                            get_device_downlink_queue_resp command description)
                [
                        ts_1,
                        …,
                        ts_n
                ]
        },
        …
        ]
}
```

Response message:

```
{
        "cmd": "manage_device_downlink_queue_resp",
        "status": boolean,
        "err_string"?: string,              //[optional – exist if status is false] contains error string code (see below
                                            description)
        "queue_manage_status"?:             //[optional – exist if status is true] contains list of devices with status of
                                            "action" executing
        [
        {
                "devEui": string,
                "action": string,
                "status": boolean           // Status of execution of corresponding "action"
        },
        …,
        ]
}
```

Possible values for "actions":

- "delete" – command to delete packets

Possible values for "err_string":

- "inaccessible_command" – returns if current user don't have access for this command;
- "invalid_cmd" – returns if command don't contain "devices_list";
- "empty_input_parameter_list" – returns if "devices_list" is empty

Example request message:

```
{
        “cmd”: “manage_device_downlink_queue_req”,
         “devices_list”:
        [ ]
}
```

Example response message:

```
{
        “cmd”: “get_device_downlink_queue_resp”,
        “status”: false,
        “err_string”: “empty_input_parameter_list”
}
```

Example request message:

```
{
        “cmd”: “manage_device_downlink_queue_req”,
         “devices_list”:
        [
        {
                “devEui”: “0000000000000001”
                “action”: “delete”,
                “ts_list”:
                [
                        1505979552026
                ]
        },
        {
                “devEui”: “0000000000000001”:                // two blocks for one device with similar “action” would
                                                             be joined in one
                “action”: “delete”,
                “ts_list”:
                [
                        1505979553348
                ]
        },
        {
                “devEui”: “0000000000000001”:                // this block would be skipped, “action” is absent
                 “ts_list”:
                [
                        1505979553348
                ]
        },

        {
                “devEui”: “0000000000000002”:                // all packets would be deleted (“ts_list” is absent)
                “action”: “delete”
        }
        ]
}
```

Example response message:

```
{
        “cmd”: “get_device_downlink_queue_resp”,
        “status”: true,
        “queue_manage_status”:
        [
        {
                “devEui”: “0000000000000001”,
                “action”: “delete”,
                “status”: true
        },
        {
                “devEui”: “0000000000000002”,
                “action”: “delete”,
                “status”: true
        }
        ]
}
```

Server information (server_info_req, server_info_resp)

Request message:
```
{
        "cmd": "server_info_req"
}
```

Response message:
```
{
        "cmd": "server_info_resp",
        "status": boolean,                      // status of command execution
        "err_string"?: string,                  // [optional – exist if "status" is false] error string code (see below
                                                   description)
        "version"?: string,                     // [optional – exist if "status" is true] Version of server (e.g. "1.1.4")
        "time"?:                                // [optional – exist if "status" is true] Time info
        {
                "utc": number,                  // Server UTC timestamp (milliseconds from Linux epoch)
                "time_zone": string,            // Time zone (e.g. "UTC+01:00")
                "time_zone_utc_offset": number  // Time offset from UTC in seconds (e.g. 3600)
        },
        "device_count"?:[8]                     // [optional – exist if "status" is true] Device count info
        {
                "device_count_vega": number,    // Count of registered devices of "Vega Absolute" company
                "device_count_other": number,   // Count of registered devices of others manufacturers
                "available_device_count": number // Count of free remain devices
        }
}
```

Example response message:
```
{
        "cmd": "server_info_resp",
        "status": true,
        "version": "1.2.0",
        "time":
        {
                "utc": 1513584871084,
                "time_zone": "UTC+07:00",
                "time_zone_utc_offset": 25200
        },
        "device_count":
        {
                "device_count_vega": 24,
                "device_count_other": 75,
                "available_device_count": 925
        }
}
```

Possible values for "err_string":
- "inaccessible_command" – returns if current user don't have access for this command;
- "server_is_not_ready" – returns if server is not ready for answer. Repeat request later.

[8] Devices of "Vega Absolute" company not decries total counter therefor count of such devices is unlimited. However, total counter of devices of other manufacturers is bounded. By default server can register only 1000 devices from other manufacturers.
To expand available devices count contact to "Vega Absolute" company.
To register "Vega Absolute" device enter AppEUI, AppKEY and DevEUI in registration request to server (in this case supported OTAA only)

Send e-mail message (send_email_req, send_email_resp, send_email_result_resp)

Request message:
```
{
        "cmd": "send_email_req",
        "smtp_server":                      // Parameters of SMTP server
        {
                "address": string,          // Address of server, i.e. "smtp.mail.ru"
                "port": number,             // Port of server, i.e. 465
                "connection_type": string   // String code of connection type (see below description)
        },
        "sender":                           // Sender's parameters
        {
                "name"?: string,            // [optional] sender's name
                "email": string,            // sender's email address
                "password": string          // sender's password
        }
        "recipients":
        [
                {
                        "name"?: string,    // [optional] recipient's name
                        "email": string     // recipient's email address
                }, …
        ]
        "message":
        {
                "subject": string,
                "text": string
        }
}
```

Response message – immediate answer:
```
{
        "cmd": "send_email_resp",
        "status": boolean,                  // Status of processing command
        "err_string"?: string,              // [optional – exist if "status" = false] error string (see below
                                            description)
        "email_id"?: string                 // [optional – exist if "status" = true] 16 symbols email identificator
                                            for waiting sending result
}
```

Response message – answer with sending result:
```
{
        "cmd": "send_email_result_resp",
        "status": string,                   // Status of sending (see below description)
        "email_id": string                  // ID of sending email
}
```

Possible values for "connection_type":
- "TCP" – don't use encryption;
- "SSL" – use SSL encryption;
- "TLS" – use TLS encryption.

Possible values for "err_string":
- "inaccessible_command" – returns if current user don't have access for this command;
- "invalid_input<field_name>" – show fieldname with error;
- "invalid_email_sending_timeout" – returns if time between two serial email sending is less than 1 minute (this timeout may be switch off in server configuration).

Possible values for "status" in "send_email_result_resp":
- "OK" – sending is successful;
- "INVALID_HOST_CONNECTION" – connection to email host is unsuccessful;

- "INVALID_LOGIN" – login on email host is invalid;
- "INVALID_MAIL_SENDING" – some errors happened with email sending, verify input fields.

Example request message:

```
{
        “cmd”: “send_email_req”,
        “smtp_server”:
        {
                “address”: “smtp.mail.ru”,
                “port”: 465,
                “connection_type”: “SSL”
        },
        “sender”:
        {
                “name”?: “user”,
                “email”: “user@mail.ru”,
                “password”: “examplepassword”
        }
        “recipients”:
        [
                {
                        “name”: “user1”,
                        “email”: “user1@mail.ru”
                },
                {
                        “email”: “user2@mail.ru”
                }
        ]
        “message”:
        {
                “subject”: “example”,
                “text”: “example message”
        }
```

Example response message – immediate answer:

```
{
        “cmd”: “send_email_resp”,
        “status”: true,
        “email_id”: “1234567890ABCDEF”
}
```

Example response message – answer with sending result:

```
{
        “cmd”: “send_email_result_resp”,
        “status”: “OK”,
        “email_id”: “1234567890ABCDEF”
}
```

Online response with packet info (uplink / downlink) for corresponding device (rx)

Response message:

```
{
        "cmd": "rx",
        "devEui": string,          // device EUI, 16 hex digits (without dashes)
        "appEui": string,          // application EUI, 16 hex digits (without dashes)
        "ts ": integer,            // Server UTC receiving timestamp (miliseconds from Linux epoch)
        "gatewayId": string,       // Gateways that receive data from device[9]
        "ack": boolean,            // Acknowledgement flag as set by device
        "fcnt": integer,           // Frame counter, a 32-bit number (uplink or downlink based on "direction" value)
        "port": integer,            // Port (if = 0, use JOIN operations or MAC-commands only)
        "data" : string,           // Decrypted data payload
        "macData"?: string,        //[optional – exist if MAC command is present] MAC command data from device
        "freq": integer,           // Radio frequency at which the frame was received/transmitted, in Hz
        "dr": string,              // Spreading factor, bandwidth and coding rate "SF12 BW125 4/5"
        "rssi": integer,           //[optional – exist if packet direction "UPLOAD"] Frame rssi, in dBm, as integer number
        "snr": float,              //[optional – exist if packet direction "UPLOAD"] Frame snr, in dB
        "type": string,            // Type of packet (see description for get_data). May contains several types joined
                                   via "+"
        "packetStatus"?: string    //[optional – exist if packet direction "UPLOAD"] Status of downlink message only (see
                                   description for get _data)
}
```

Example:

```
{
        "cmd": "rx",
        "devEui": "3933363845366606",
        "appEui": "0000000000000001",
        "ts ":6546544313531,
        "gatewayId": "0000000000000001+0000000000000002",
        "ack": false;
        "fcnt": 10;
        "port": 40;
        "data" : "3543543545bccb";
        "macData": "02",
        "freq": 868100000,
        "dr": "SF12 BW125 4/5",
        "rssi": -75,
        "snr": 2.6,
        "type": " UNCONF_UP +MAC_LINKCHECK_ANS"
}
```

[9] Message from one device could be delivered over by several gateways. In this packet, "gatewayId" contains ID of gateways joined via "+". I.e. "0000000000000001+0000000000000002"

Send single-frame data to device (tx)

Request message:
```
{
        "cmd": "tx",
        "status": bool
        "err_string"?: string,      //[optional] If "status" = false, contains error description (see below
                                    description)
        "devEui": string,           //device EUI, 16 hex digits (without dashes)
        "data": string,             //data payload (to be encrypted by server)
        "port": number,             //port to be used (1..223)
        "ack"?: boolean             //[optional] request confirmation (ACK) from end-device
}
```

Example:
```
{
        "cmd": "tx",
        "status": true,
        "devEui": "3933363845366606",
        "data": "25db26a2c8b4",
        "port": 40
}
```

Online response with server debug information (console)

Response message:
```
{
        "cmd": "console",
        "message": string,          //string console message
        "color": string             //color of message (see below description)
}
```

Possible values for "color":
- "common" – color for common messages. It may be white on black or black on white;
- "red" – color for message with warnings or errors;
- "yellow" – color for "JOIN_REQ" messages;
- "green" – color for valid uplink messages;
- "cyan" – color for "JOIN_ACCEPT" messages;
- "purple" – color for downlink messages;
- "blue" – color for repeated uplink messages.

Example:
```
{
        "cmd": "console",
        "message": "Some debug information",
        "color": "common"
}
```

Revision history

Rev20 => Rev21	1. Rename commands "bs" to "gateway", "macBs" to "gatewayId" and "base station" to "gateway"; 2. Add section "settingsApplyStatus" to "get_device_resp" commands; 3. Add new results strings into "manage_device_resp"; 4. Append new error types into "get_data_resp" and same command; 5. Append "appEui_list" into "select" field of "get_device_appdata_req" and "get_devices_req" for filtering devices with inherent AppEUI; 6. Add new parameter to device parameters ("get_device_resp" and "manage_device_req"): "useDownlinkQueueClassC" and "serverAdrEnable"
Rev21 => Rev22	1. Modification of "alter_user_resp" packet; 2. Add "server_info_req" and "server_info_resp"; 3. Add "appEui" field into "get_device_appdata_resp", "get_data_resp" and "rx" packets; 4. Add new group of command "send_email"; 5. Add "port" field into "get_data_req" "select" field; 6. Add "ping_req" and "ping_resp" packets; 7. Add "dataStoragePeriod" field into "get_devices" and "manage_devices" groups;
Rev22 => Rev23	1. Add description for "REPEAT" prefix for type field in "get_data_resp" 2. Fix incorrect example for registering ABP device (see "manage_devices_req")