**Application API Data Format**

**General**

The data passed over to the various APIs / transport layers are always JSON formatted objects. Every object starts with the '{' character and ends with '}' character.  
No delimiters are used between the objects.

The cmd field is present in all messages and indicates the message type

**Uplink data message (extended verbosity)**

Every uplink JSON message corresponds to one frame produced by your device. It contains not only the raw application payload, but also additional meta-data.

If APPSKEY is assigned to a device, the data field will be populated with decrypted data, and encdata will be ommited.  
If APPSKEY is missing, the encdata field will be populated with encypted payload, and data will be ommited.

The extended radio information is reported as seen by the first gateway that received the frame.

**Example message**

{

"cmd" : "rx",

"EUI" : "0102030405060708",

"ts" : 1470850675433,

"ack" : false,

"fcnt" : 1,

"port" : 1,

"data" : "0102AABB",

"freq" : 868500000,

"dr" : "SF12 BW125 4/5",

"rssi" : -130,

"snr" : 1.2

}

**Message format** [**Typescript interface notation**](https://www.typescriptlang.org/docs/handbook/interfaces.html)

{

cmd : 'rx'; // identifies type of message, rx = uplink message

EUI : string; // device EUI, 16 hex digits (without dashes)

ts : number; // server timestamp as number (miliseconds from Linux epoch)

ack : boolean; // **NEW!** acknowledgement flag as set by device

fcnt : number; // frame counter, a 32-bit number

port : number; // port as sent by the end device

encdata? : string; // data payload (APPSKEY encrypted hex string)

// only present if APPSKEY is assigned to device

data? : string; // data payload (decrypted ,plaintext hex string)

// only present if APPSKEY is not assigned to device

// extended radio information

freq : number; // radio frequency at which the frame was received, in Hz

dr : string; // radio data rate - spreading factor, bandwidth and coding rate

// e.g. SF12 BW125 4/5

// **UPDATED: previously the field was called 'sf'**

rssi : number; // frame rssi, in dBm, as integer number

snr : number; // frame snr, in dB, one decimal place

}

**Parameter table**

| **Parameter** | **Type** | **Description** |
| --- | --- | --- |
| cmd | string | identifies type of message, always 'rx' for uplink data messages |
| EUI | string | device EUI, 16 hex digits (without dashes) |
| ts | number | server timestamp as a number of seconds from Linux epoch |
| ack | boolean | **NEW!** acknowledgement flag as set by device |
| fcnt | number | frame counter, a 32-bit integer number |
| port | number | port number as sent by the end device |
| encdata | string (optional) | encrypted data payload as hexadecimal string, only present if APPSKEY is not assigned to device |
| data | string (optional) | decrypted data payload as hexadecimal string, only present if APPSKEY is assigned to device |
| freq | number | radio frequency at which the frame was received, in Hz |
| dr | string | radio data rate - spreading factor, bandwidth and coding rate, e.g. SF12 BW125 4/5 |
| rssi | number | frame rssi, in dBm, as integer number |
| snr | number | frame snr, in dB, one decimal place |

**Gateway information**

Keep in mind this messages has higher latency than the standard data 'rx' message, as it has to wait for all the gateways to report the status.

**Example message**

{

"cmd" : "gw",

"EUI" : "0102030405060708",

"ts" : 1470850675433,

"ack" : false,

"fcnt" : 1,

"port" : 1,

"data" : "0102AABB",

"freq" : 868500000,

"dr" : "SF12 BW125 4/5",

"gws" : [

{

"rssi" : -130,

"snr" : 1.2,

"ts" : 43424140,

"gweui" : "1122334455667788.0",

"lat" : 47.284687,

"lon" : 8.565746

}

]

}

**Message format** [**Typescript interface notation**](https://www.typescriptlang.org/docs/handbook/interfaces.html)

{

cmd : 'gw'; // identifies type of message, gw = gateway information

EUI : string; // device EUI, 16 hex digits (without dashes)

ts : number; // server timestamp as number (miliseconds from Linux epoch)

ack : boolean; // **NEW!** acknowledgement flag as set by device

fcnt : number; // frame counter, a 32-bit number

port : number; // port as sent by the end device

encdata? : string; // data payload (APPSKEY encrypted)

// only present if APPSKEY is assigned to device

// string format based on application setup (default is hex string)

data? : string; // data payload

// only present if APPSKEY is not assigned to device

// string format based on application setup (default is hex string)

// extended radio information

freq : number; // radio frequency

dr : string; // radio spreading factor, bandwidth and coding rate

// e.g. SF12 BW125 4/5

gws : array; // array of gateway information objects

// see the verbosity levels below

}

**Parameter table**

| **Parameter** | **Type** | **Description** |
| --- | --- | --- |
| cmd | string | identifies type of message, always 'gw' for gateway information messages |
| EUI | string | device EUI, 16 hex digits (without dashes) |
| ts | number | server timestamp as a number of seconds from Linux epoch |
| ack | boolean | **NEW!** acknowledgement flag as set by device |
| fcnt | number | frame counter, a 32-bit integer number |
| port | number | port number as sent by the end device |
| encdata | string (optional) | encrypted data payload as hexadecimal string, only present if APPSKEY is not assigned to device |
| data | string (optional) | decrypted data payload as hexadecimal string, only present if APPSKEY is assigned to device |
| freq | number | radio frequency at which the frame was received, in Hz |
| dr | string | radio data rate - spreading factor, bandwidth and coding rate, e.g. SF12 BW125 4/5 |
| gws | array | array of gateway information objects, see the verbosity levels below |

**gws: Verbosity level 'Location'**

**Message format** [**Typescript interface notation**](https://www.typescriptlang.org/docs/handbook/interfaces.html)

{

rssi : number; // radio rssi

snr : number; // radio snr

ts : number; // timestamp (gateway internal counter)

gweui : string; // gateway extended EUI as EUI.RADIO

// e.g. "1122334455667788.0"

lat : number; // gateway latitude

lon : number; // gateway longitude

}

**Parameter table**

| **Parameter** | **Type** | **Description** |
| --- | --- | --- |
| rssi | number | frame rssi, in dBm, as integer number |
| snr | number | frame snr, in dB, one decimal place |
| ts | number | gateway internal counter |
| gweui | string | gateway extended EUI as EUI.RADIO, e.g. 1122334455667788.0 |
| lat | number | gateway latitude, fractional degrees |
| lon | number | gateway longitude, fractional degrees |

**Downlink messaging**

* ***please ask CRA for corresponding Application and security token***
* ***the specified URL for sending downlink commands is*** [*https://parrot.iot.cra.cz/1/rest*](https://parrot.iot.cra.cz/1/rest)

You can send data down to the end devices and their applications can respond to over-the-air join requests.

You can send downlink messages using either plaintext (we encrypt the data in the server) or ciphertext (you encrypt the data).

Messages will be enqueued and delivered based on target device class.

Please ask CRA (your LoRaWAN network operator) for corespond Application

**Sending plaintext**

**Example message**

{

"cmd" : "tx",

"EUI" : "0102030405060708",

"port" : 1,

"confirmed" : true,

"data" : "0102AABB"

}

**Message format** [**Typescript interface notation**](https://www.typescriptlang.org/docs/handbook/interfaces.html)

{

cmd : 'tx'; // must always have the value 'tx'

EUI : string; // device EUI, 16 hex digits (without dashes)

port : number; // port to be used (1..223)

confirmed? : boolean; // **NEW!** (optional) request confirmation (ACK) from end-device

data : string; // data payload (to be encrypted by our server)

// if no APPSKEY is assigned to device, this will return an error

}

**Parameter table**

| **Parameter** | **Type** | **Description** |
| --- | --- | --- |
| cmd | string | identifies type of message, always 'tx' for downlink messages |
| EUI | string | device EUI, 16 hex digits (without dashes) |
| port | number | port number (1 to 223) |
| confirmed | boolean (optional) | **NEW!** request confirmation (ACK) from end-device |
| data | string | data payload (to be encrypted by our server) as a hexadecimal string |

***Please use* curl *commands to send downlink messages***

**Example message**

curl -k <https://parrot.iot.cra.cz/1/rest> -H "Authorization: Bearer XXXXXXXXXXXXXXXXXXXXXXXXXXXX" -H "Content-Type: application/json" --data "{\"appid\":\"BE010002\",\"cmd\":\"tx\",\"EUI\":\"393933387232620F\",\"port\":2,\"data\" : \"00\"}"

**Sending ciphertext**

The payload has to be encrypted using the latest downlink sequence number and the appropriate APPSKEY

The downlink sequence number (seqdn) must correspond to the latest downlink sequence number reported by the 'txd' message. If sequence number doesn't match, an error will be returned with the current seqdn

**Example message**

{

"cmd" : "tx",

"EUI" : "0102030405060708",

"port" : 1,

"confirmed" : true,

"encdata" : "0102AABB"

"seqno" : 1,

}

**Message format** [**Typescript interface notation**](https://www.typescriptlang.org/docs/handbook/interfaces.html)

{

cmd : 'tx'; // must always have the value 'tx'

EUI : string; // device EUI, 16 hex digits (without dashes)

port : number; // port to be used (1..223)

confirmed? : boolean; // **NEW!** (optional) request confirmation (ACK) from end-device

encdata : string; // data payload (already APPSKEY encrypted)

seqno : number; // must correspond to the latest seqdn reported by the 'txd' message

}

**Parameter table**

| **Parameter** | **Type** | **Description** |
| --- | --- | --- |
| cmd | string | identifies type of message, always 'tx' for downlink messages |
| EUI | string | device EUI, 16 hex digits (without dashes) |
| port | number | port number (1 to 223) |
| confirmed | boolean (optional) | **NEW!** request confirmation (ACK) from end-device |
| encdata | string | data payload (APPSKEY encrypted) as a hexadecimal string |
| seqno | number | must correspond to the latest seqdn reported by the txd message |

**Acknowledgment of send request**

Acknowledgement is returned immediately after issuing the 'tx' command.

**Example message**

{

"cmd" : "tx",

"EUI" : "0102030405060708",

"success" : "Downlink message enqueued.",

"data" : "0102AABB"

}

**Message format** [**Typescript interface notation**](https://www.typescriptlang.org/docs/handbook/interfaces.html)

{

cmd : 'tx'; // always has the value 'tx'

EUI : string; // device EUI, 16 hex digits (without dashes)

success? : string; // on success, will contain a confirmation message

// only present if the command succeeded

error? : string; // string describing the encountered error

// only present if the command failed

data? : string; // data that was enqueued (either plaintext or ciphertext)

// only present if the command succeeded

}

**Parameter table**

| **Parameter** | **Type** | **Description** |
| --- | --- | --- |
| cmd | string | identifies type of message, always 'tx' for this type of message |
| EUI | string | device EUI, 16 hex digits (without dashes) |
| success | string (optional) | If command succeeded, it will contain a confirmation message. |
| error | string (optional) | If command failed, it will report the error description. |
| data | string (optional) | data that was enqueued (either plaintext or ciphertext), only present on success. |

**Downlink confirmation event**

The confirmation event is sent once the packet has been sent to gateway for delivery

**Example message**

{

"cmd" : "txd",

"EUI" : "0102030405060708",

"seqdn" : 1,

"ts" : 1470850675434

}

**Message format** [**Typescript interface notation**](https://www.typescriptlang.org/docs/handbook/interfaces.html)

{

cmd : 'txd'; // always has the value 'txd'

EUI : string; // device EUI, 16 hex digits (without dashes)

seqdn : number; // sequence number used for the downlink

ts : number; // unix timestamp, moment of the transfer to gateway

}

**Parameter table**

| **Parameter** | **Type** | **Description** |
| --- | --- | --- |
| cmd | string | identifies type of message, always 'txd' for this type of message |
| EUI | string | device EUI, 16 hex digits (without dashes) |
| seqdn | number | Sequence number used for the downlink |
| ts | number | Unix timestamp, at the moment of the transfer to gateway |