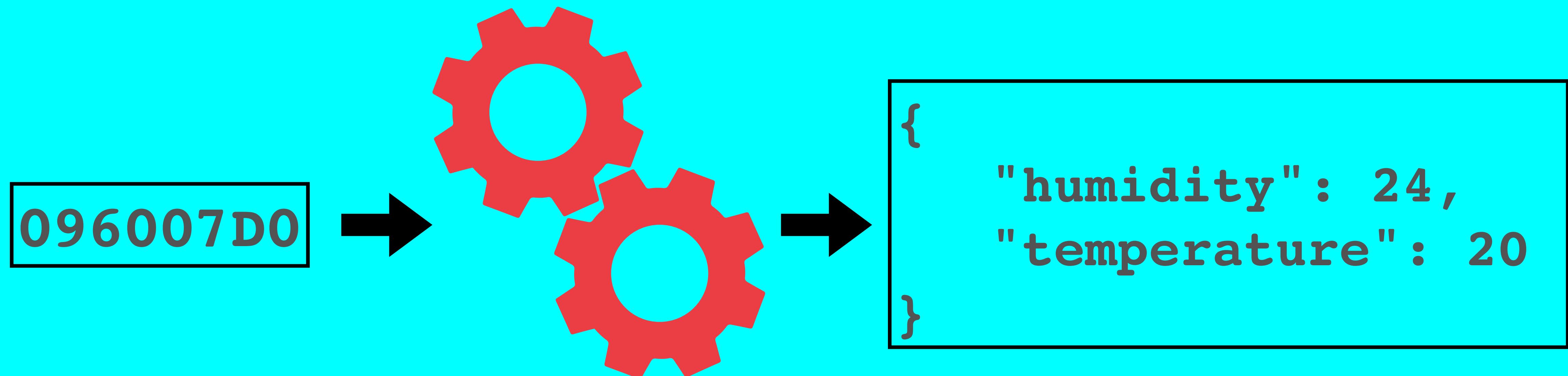


LORA / LORAWAN TUTORIAL 53

Payload Formatters (U3), fport & json



INTRO

- In this tutorial I will explain what fport is and a short refresher what a json object is.
- I will also explain what payload formatters are and show you several Javascript payload formatters examples.

PRESENTATION

- This presentation can be found at:

https://www.mobilefish.com/download/lora/lora_part53.pdf

- All my LoRa/LoRaWAN tutorials and presentations can be found at:

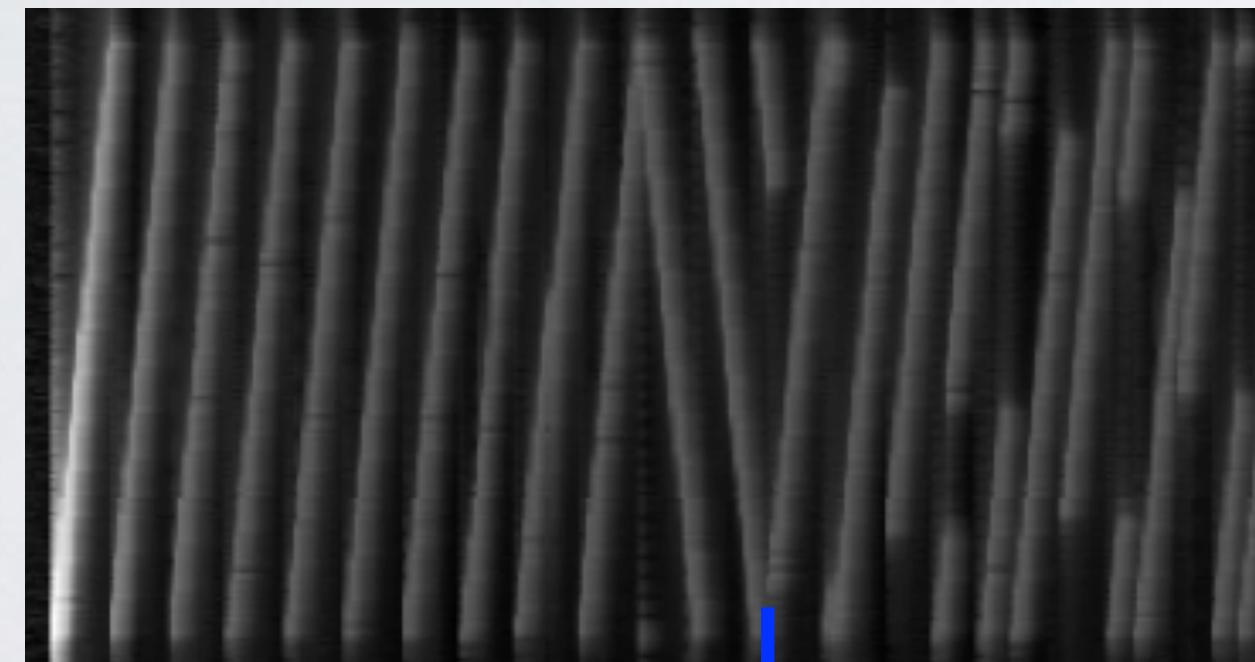
https://www.mobilefish.com/developer lorawan lorawan_quickguide tutorial.html

- In this video when V2 is mentioned, V2 refers to The Things Network and when V3 is mentioned, V3 refers to The Things Stack Community Edition. This is to keep this video as short as possible.

LORA PACKET FORMAT

- In tutorial 17, I have discussed the LoRa packet format. The LoRa packet comprises of three elements: Preamble, header (optional) and payload.

CRC = Cyclic Redundancy Check



Spectrogram of a LoRa packet

preamble | header + payload

RTL-SDR DONGLE

- To create the Radio Frequency (RF) waterfall, I used a RTL-SDR (Software Define Radio) dongle and a sleeve dipole antenna (antenna C in tutorial 33).

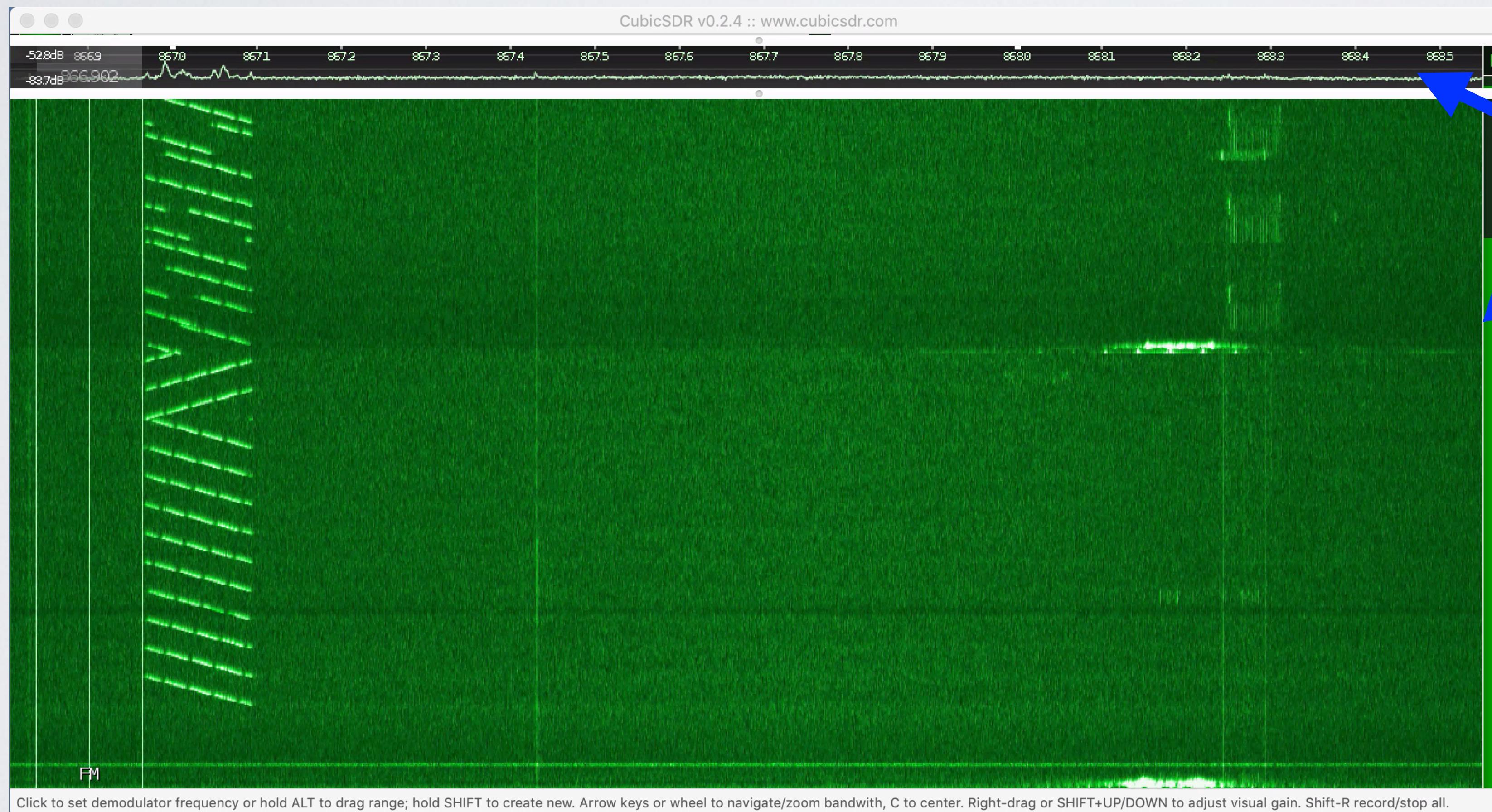
Information about RTL-SDR dongle:
<https://youtu.be/bSAa2aOXpCc>

- The RTL-SDR dongle is nameless but has the following chipsets:
The radio tuner: Rafael Micro R820T2
The demodulator: RealTek RTL2832U



CUBICSDR SOFTWARE

- To create the Radio Frequency (RF) waterfall the CubicSDR software v0.2.4 is used on a macbook. More information: <https://cubicsdr.com/>



**Make sure
the freq. range
lies between
867.1 - 868.5 MHz
(Europe).**

**Waterfall max speed
changed to 400 lines
per second.**

LORA PACKET FORMAT

Radio PHY layer:



Figure 5: Radio PHY structure (CRC* is only available on uplink messages)

PHYPayload:



or



or



Figure 6: PHY payload structure

MACPayload:



Figure 7: MAC payload structure

FHDR:



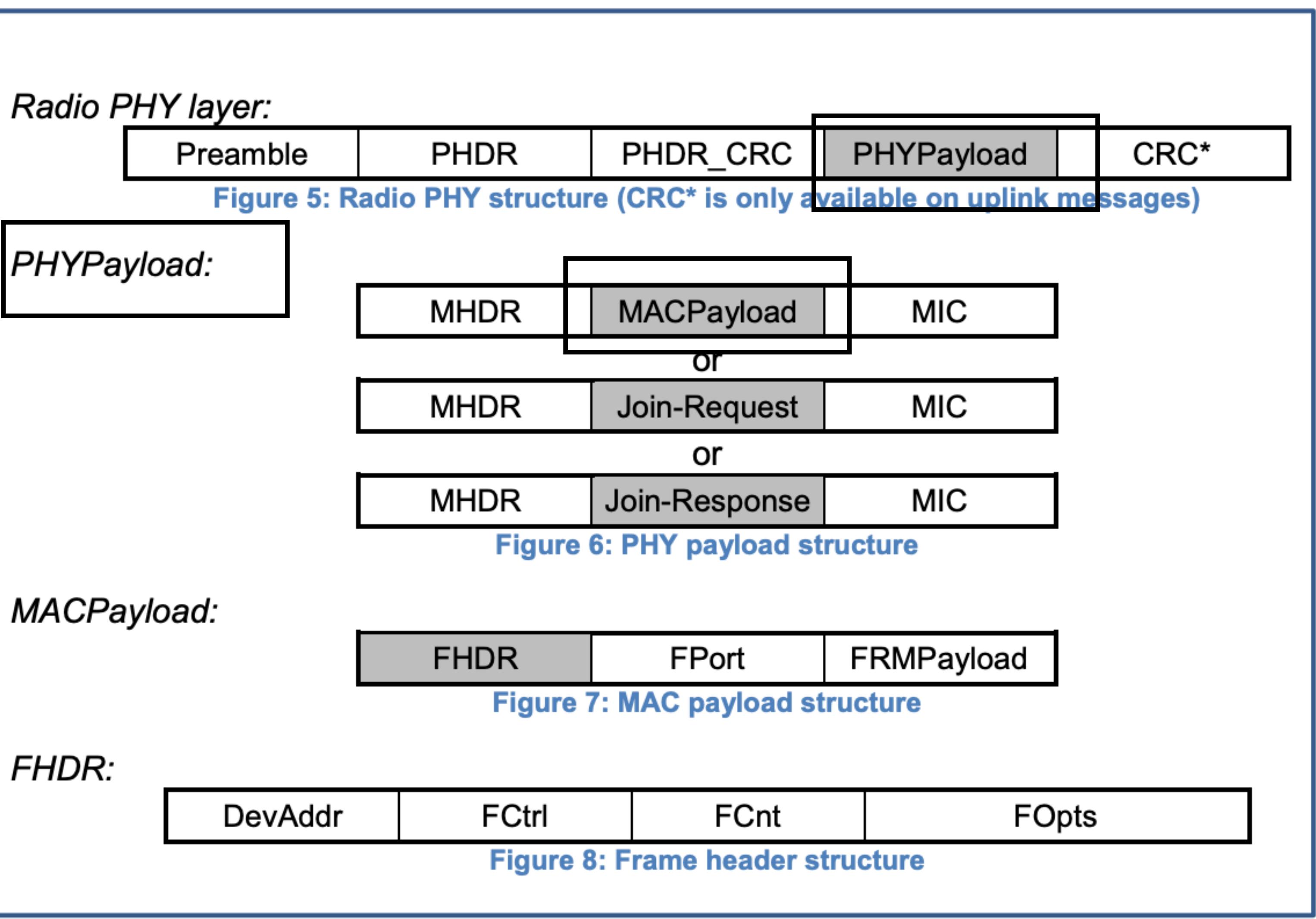
Figure 8: Frame header structure

Source:

LoRaWAN Specification
v1.0.2

[https://lora-alliance.org/
wp-content/uploads/
2020/11/
lorawan1_0_2-20161012
_1398_1.pdf](https://lora-alliance.org/wp-content/uploads/2020/11/lorawan1_0_2-20161012_1398_1.pdf)

LORA PACKET FORMAT



All LoRa uplink and downlink messages carry a PHYPayload starting with a MAC header (MHDR), followed by a MACPayload and ending with a message integrity code (MIC).

LORA PACKET FORMAT

Radio PHY layer:



Figure 5: Radio PHY structure (CRC* is only available on uplink messages)

PHYPayload:

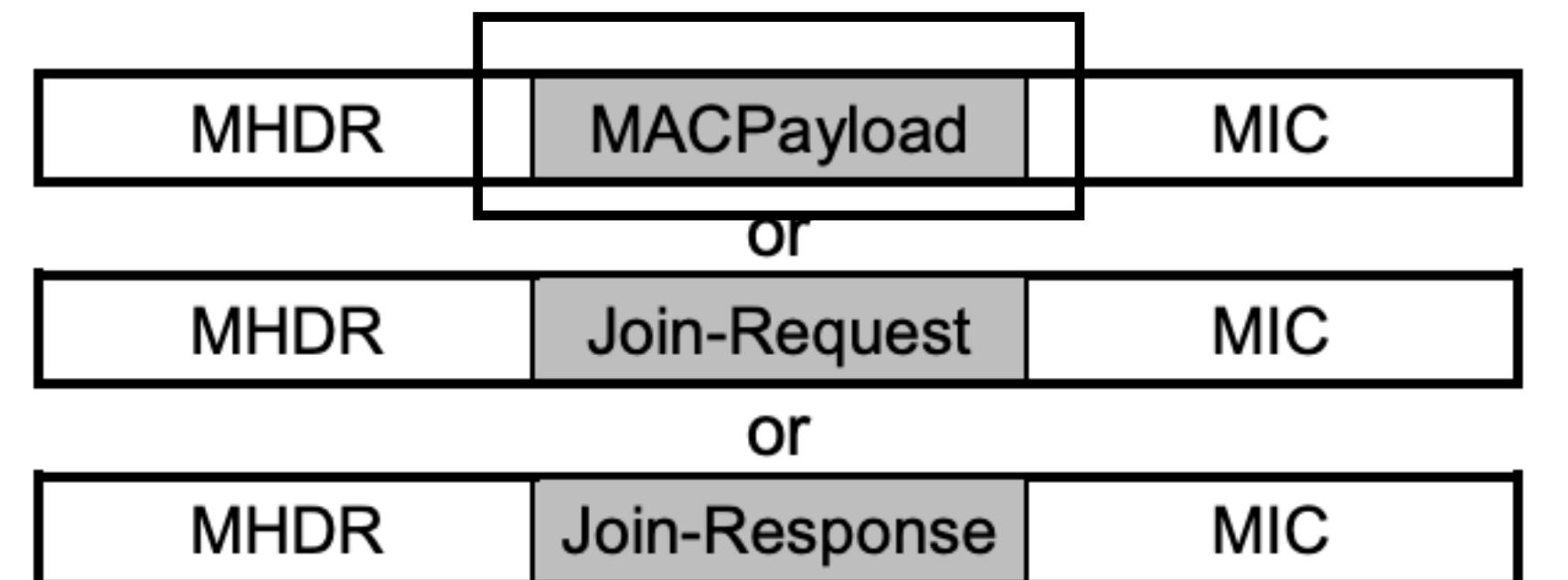


Figure 6: PHY payload structure

MACPayload:



Figure 7: MAC payload structure

FHDR:



Figure 8: Frame header structure

The MACPayload contains a frame header (FHDR) followed by an optional port field (FPort) and an optional frame payload field (FRMPayload).

FPORT

- **FPort** means **Frame Port**. If the frame payload field (FRMPayload) is not empty, the port field must be present. If present, an FPort value of 0 indicates that the FRMPayload contains MAC (Media Access Control) commands only.



- According to the LoRaWAN specification:
 - FPort values 1 - 223 (dec) are application specific.
 - FPort value 224 is dedicated to LoRaWAN Mac layer test protocol.
 - FPort values 225 - 255 (dec) are reserved for future standardized application extensions.

FPORT

- ...but in the technical recommendation "FUOTA Process Summary Technical Recommendation TR002 v1.0.0" regarding **Firmware Update Over-the-Air** on top of the LoRaWAN protocol, fports 225 and 200-203 are listed for specific usage.

https://lora-alliance.org/wp-content/uploads/2020/11/tr002-fuota_process_summary-v1.0.0.pdf

Package ID	Allocated FPort	Proposed FPort	Package version	Name	Version	Document link	Status	Release Date
0	225		1DRAFT	Multi Package Access Protocol over LoRaWAN	RC1	Multi_Package_Access_rc1.docx	Release candidate	
1		202	1	LoRaWAN Application Layer Clock Synchronization	v1.0.0	LoRaWAN Application Layer Clock Synchronization Specification v1.0.0	Published	September 10, 2018
2		200	1	LoRaWAN Remote Multicast Setup Specification	v1.0.0	LoRaWAN Remote Multicast Setup Specification v1.0.0	Published	September 10, 2018
3		201	1	LoRaWAN Fragmented Data Block Transport	v1.0.0	LoRaWAN Fragmented Data Block Transport Specification v1.0.0	Published	September 10, 2018
4		203	1DRAFT	LoRaWAN Firmware Management Protocol	RC1	Firmware_Management_Protocol_rc1.docx	Release candidate	

Table 1: Package Identifier List

FPORT

- Combining the LoRaWAN specification and the FUOTA Process Summary Technical Recommendation, users should only use the following ports:

fport 1-223, except 200, 201, 202 and 203

FPORT FOR DUMMIES EXPLANATION

Radio PHY layer:



Figure 5: Radio PHY structure (CRC* is only available on uplink messages)

PHYPayload:

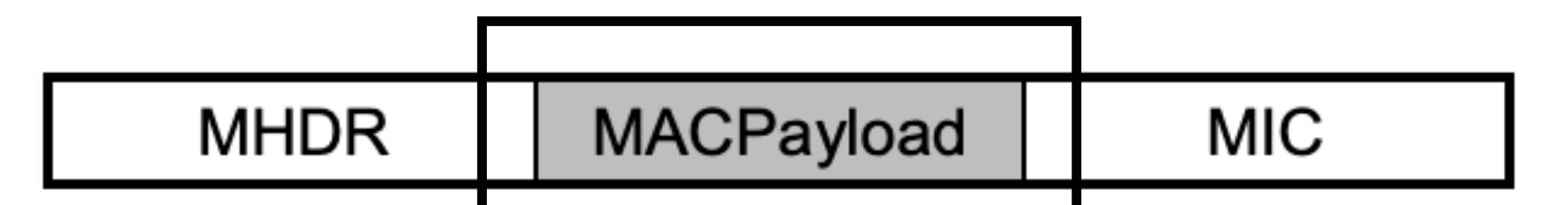


Figure 6: PHY payload structure

MACPayload:

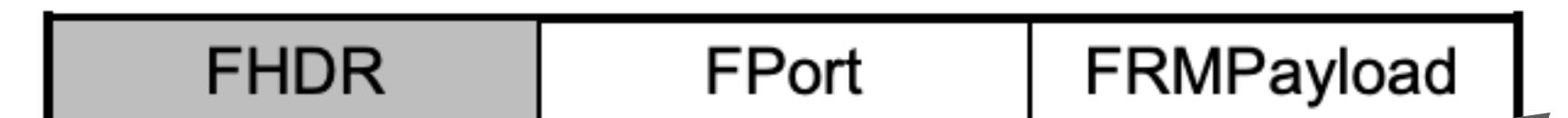


Figure 7: MAC payload structure

FHDR:



Figure 8: Frame header structure

FPort is 1 byte in size.

Allowed values:

1 - 223, except fports 200-203.

This byte is always sent, even if you use it or not.

Hackers can read this value, but if they change this value, the package is dropped by the gateway because of the Message Integrity Code (MIC).

Encrypted sensor data is stored here.
(Frame Payload)

FPORT HAS ANOTHER PURPOSE

- If a data frame carries a payload, FRMPayload must be encrypted before the message integrity code (MIC) is calculated.



- The FRMPayload is encrypted with the NwkSkey if FPort is 0 otherwise it is encrypted with the AppSkey.
- *But you can ignore the above mentioned information because it is not relevant for this tutorial.*

WHERE DO I SET THE FPORT?

- My end device uses this Arduino sketch:
<https://www.mobilefish.com/download/lora/ttn-otaa-pro-mini-sensors.ino.txt>
- This sketch uses the MCCI LoRaWAN LMIC library in my Arduino IDE (v1.8.10)
<https://github.com/mcci-catena/arduino-lmic>

MCCI LoRaWAN LMIC library by **IBM, Matthijs Kooijman, Terry Moore, ChaeHee Won, Frank Rose** Version **3.3.0 INSTALLED**
Arduino port of the LMIC (LoRaWAN-MAC-in-C) framework provided by IBM. Supports LoRaWAN 1.0.2/1.0.3 Class A devices implemented using the Semtech SX1272/SX1276 (including HopeRF RFM92/RFM95 and Murata modules). Support for EU868, US, AU, AS923, KR and IN regional plans. Untested support for Class B and FSK operation. Various enhancements and bug fixes from MCCI and The Things Network New York. Original IBM URL <http://www.research.ibm.com/labs/zurich/ics/lrsc/lmic.html>.
[More info](#)

- In the sketch, change function LMIC_setTxData2 to change the fport value.
LMIC_setTxData2(fport, data, data_length, confirmed)
LMIC_setTxData2(1, payload, sizeof(payload), 0);

WHERE DO I SET THE FPORT?

- If you use the MCCI LoRaWAN LMIC library, more sketch examples:
<https://github.com/mcci-catena/arduino-lmic/tree/master/examples>

FPORT VALUE CHANGED TO DIFFERENT VALUES

The screenshot shows a log of sensor data from an application named "youtube-demo-app2". The data is categorized by type: "Forward uplink data message". The log includes the following entries:

Time	Entity ID	Type	Data preview	Verbose stream	Pause	Clear
↑ 14:38:42	youtube-demo-de...	Forward uplink data message	Payload: { humidity: 23, temperature: 21 } 08 FC 08 34	<input type="checkbox"/>	FFPort: 11 SNR: 5.2	
↪ 14:38:36	youtube-demo-de...	Accept join-request				
↪ 14:37:25	youtube-demo-de...	Accept join-request				
↪ 14:36:17	youtube-demo-de...	Accept join-request				
↪ 14:31:43	youtube-demo-de...	Accept join-request				
↑ 14:30:39	youtube-demo-de...	Forward uplink data message	Payload: { humidity: 23, temperature: 21 } 08 FC 08 34	<input type="checkbox"/>	FFPort: 2 SNR: 4.5	
↪ 14:30:34	youtube-demo-de...	Accept join-request				
↪ 14:29:27	youtube-demo-de...	Accept join-request				
↑ 14:28:28	youtube-demo-de...	Forward uplink data message	Payload: { humidity: 23, temperature: 21 } 08 FC 08 34	<input type="checkbox"/>	FFPort: 1 SNR: -3.2	
↑ 14:27:51	youtube-demo-de...	Forward uplink data message	Payload: { humidity: 23, temperature: 21 } 08 FC 08 34	<input type="checkbox"/>	FFPort: 1 SNR: -7.2	

A large text overlay "No sensor data received" is centered over the log entries.

On the right side of the interface, there are four bold text labels corresponding to the FPort values:

- FFPort = 11
- FFPort = 0
- FFPort = 2
- FFPort = 1

FPORT VALUE CHANGED TO DIFFERENT VALUES

The screenshot shows the 'Live data' section of the The Things Network web interface. At the top, there are logos for 'THE THINGS NETWORK' and 'THE THINGS STACK Community Edition'. Below this, a sidebar on the left includes icons for Applications, Devices, Sensors, and LoRaWAN. The main area displays a table of data with columns: Time, Entity ID, Type, Data preview, Verbose stream (checkbox), Pause (button), and Clear (button). The table lists five entries:

Time	Entity ID	Type	Data preview	Verbose stream	Pause	Clear
↑ 14:41:56	youtube-demo-de...	Forward uplink data message	Payload: { humidity: 23, temperature: 21 } 08 FC 08 34	<input type="checkbox"/>	FPort: 223	SNR: -2
↑ 14:41:19	youtube-demo-de...	Forward uplink data message	Payload: { humidity: 23, temperature: 21 } 08 FC 08 34	<input type="checkbox"/>	FPort: 223	SNR: -7
✉ 14:41:14	youtube-demo-de...	Accept join-request				
↑ 14:40:31	youtube-demo-de...	Forward uplink data message	Payload: { humidity: 23, temperature: 21 } 08 FC 08 34	<input type="checkbox"/>	FPort: 11	SNR: -4.

FPort = 223

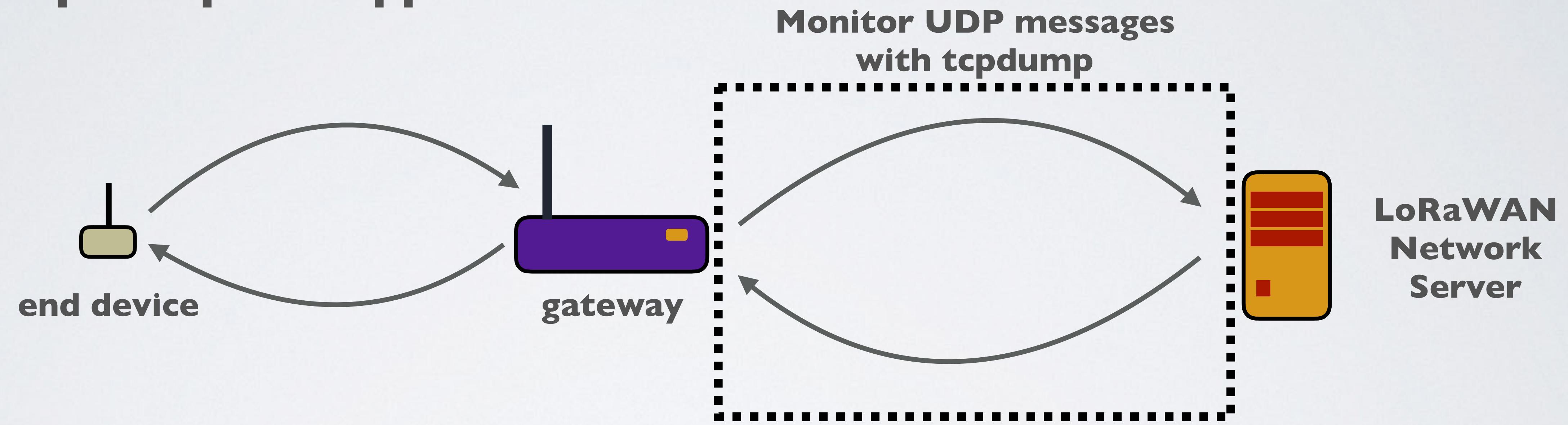
DEMO TO INTERCEPT THE FPORT VALUE

- I am using the RAK7244C (WisGate Developer D4+) LoRaWAN gateway.
This gateway uses the Raspberry Pi 4.
More information this gateway, see tutorial 51.
- To monitor UDP messages send between the RAK7244C and the V3 network install the tcpdump application:
 - Access the RAK7244C, for example: **ssh pi@192.168.2.167**
 - Upgrade the Raspberry Pi packages:
sudo apt-get update && sudo apt-get upgrade -y
 - Install tcpdump:
sudo apt-get install tcpdump -y

DEMO TO INTERCEPT THE FPORT VALUE

- Start tcpdump:

sudo tcpdump -AUq port 1700



```
● ● ● robertlie — pi@rak-gateway: ~ — ssh pi@192.168.2.167 — 77x22
E..._I@.0.....4.....2..6[G>{"rxpk": [{"tmst":922138443, "chan":0, "rfch":1, "freq":868.100000, "stat":1, "modu":"LORA", "datr":"SF7BW125", "codr":"4/5", "lsnr":9.0, "rss":-40, "size":17, "data":"QIbHCyaAAQAKGjp4Iigma3E="}]}]
```

DEMO TO INTERCEPT THE FPORT VALUE

- More information about the UDP messages, see tutorial 29.
- There is an online tool available which can decode the LoRaWAN 1.0.x packet.
<https://lorawan-packet-decoder-0ta6puiniaut.runkit.sh/>
This tool uses the LoRa radio packet decoder created by Anthony Kirby:
<https://github.com/anthonykirby/lora-packet>

**USE THE ONLINE TOOL FOR EDUCATIONAL,
DEMONSTRATION OR TEST PURPOSES.**

**NEVER ENTER YOUR PRODUCTION SECRET NWKSKEY OR
APPSKEY.**

DEMO TO INTERCEPT THE FPORT VALUE

- From tcpdump get the base64 encoded data:

QIbHCyaAAQAKGjp4Iigma3E=

- Enter the base64 encoded data in the LoRaWAN 1.0.x packet decoder.

Base64 or hex-encoded packet
QIbHCyaAAQAKGjp4Iigma3E=

Secret NwkSKey (hex-encoded; optional)
Network Session Key

Secret AppSKey (hex-encoded; optional)
Application (Session) Key

Decode

Assuming base64-encoded packet
QIbHCyaAAQAKGjp4Iigma3E=

Message Type = Data
PHYPayload = 4086C70B268001000A1A3A782228266B71

(PHYPayload = MHDR[1] | MACPayload[...] | MIC[4])
MHDR = 40
MACPayload = 86C70B268001000A1A3A7822
MIC = 28266B71

(MACPayload = FHDR | FPort | FRMPayload)
FHDR = 86C70B26800100
FPort = 0A
FRMPayload = 1A3A7822

← **FPort = 10**

(FHDR = DevAddr[4] | FCtrl[1] | FCnt[2] | FOpts[0..15])
DevAddr = 260BC786 (Big Endian)
FCtrl = 80
FCnt = 0001 (Big Endian)
FOpts =

Message Type = Unconfirmed Data Up
Direction = up
FCnt = 1
FCtrl.ACK = false
FCtrl.ADR = true

CHANGE FPORT IN LORAWAN END DEVICES

- There are LoRaWAN end devices where fports can not be changed. These devices have fixed fport values.
- There are LoRaWAN end devices where fports can be changed programmatically (for example in an Arduino sketch) or with at-commands.
- There are LoRaWAN end devices where fports can only be changed by modifying, compiling and uploading the new firmware to the end device.
- If you want to know if you can change your LoRaWAN end device fport value, check your end device manual.

FPORT USAGE

- It is recommended that you use the fport because this byte is always sent, you might as well use it.
- Every byte not transmitted improves the end device battery life.
- But be careful of the kind of information the fport value represents.
Hackers can use this information for criminal activities.

EXAMPLE NOT USING FPORT

- Do not use fport if it contains information which can be useful to criminals.



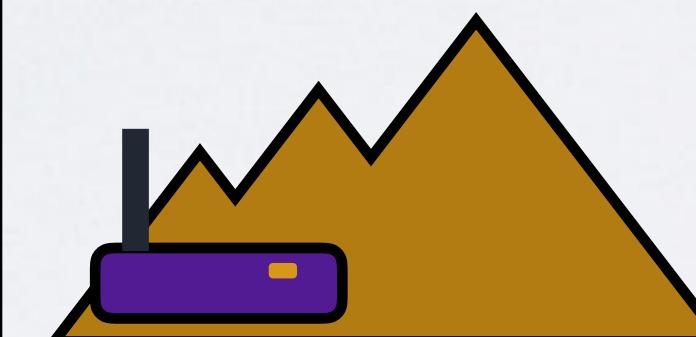
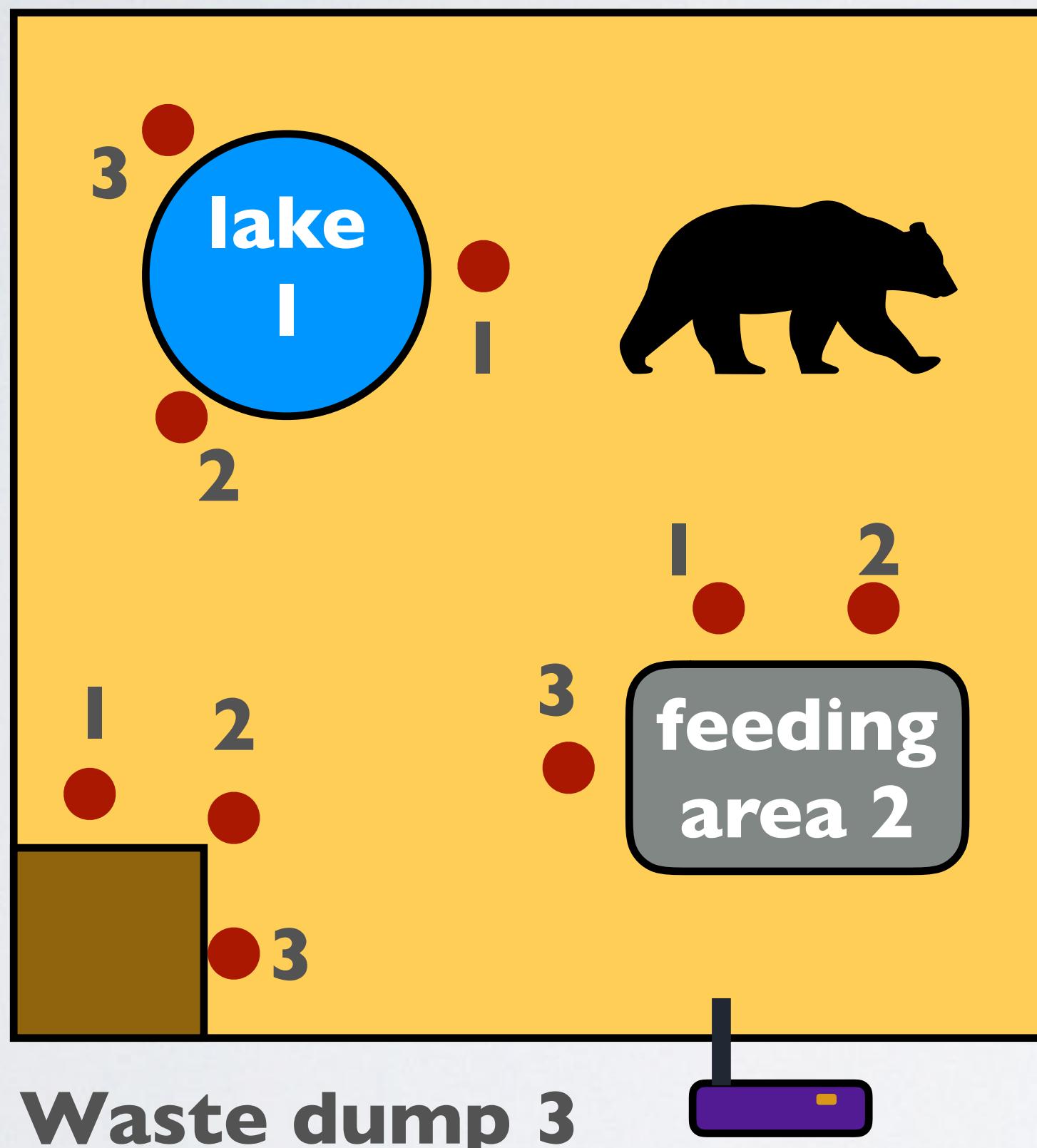
Park rangers are researching the bear's behaviour. This bear is unique, it is 3x its normal size (very large). Rangers are monitoring how often does the bear visits the lake, feeding area and waste dump.

Motion sensors are placed near the entrances.

Lake	- fport = 1
Feeding area	- fport = 2
Waste dump	- fport = 3

EXAMPLE NOT USING FPORT

- A criminal can sell the bear's location to illegal hunters.



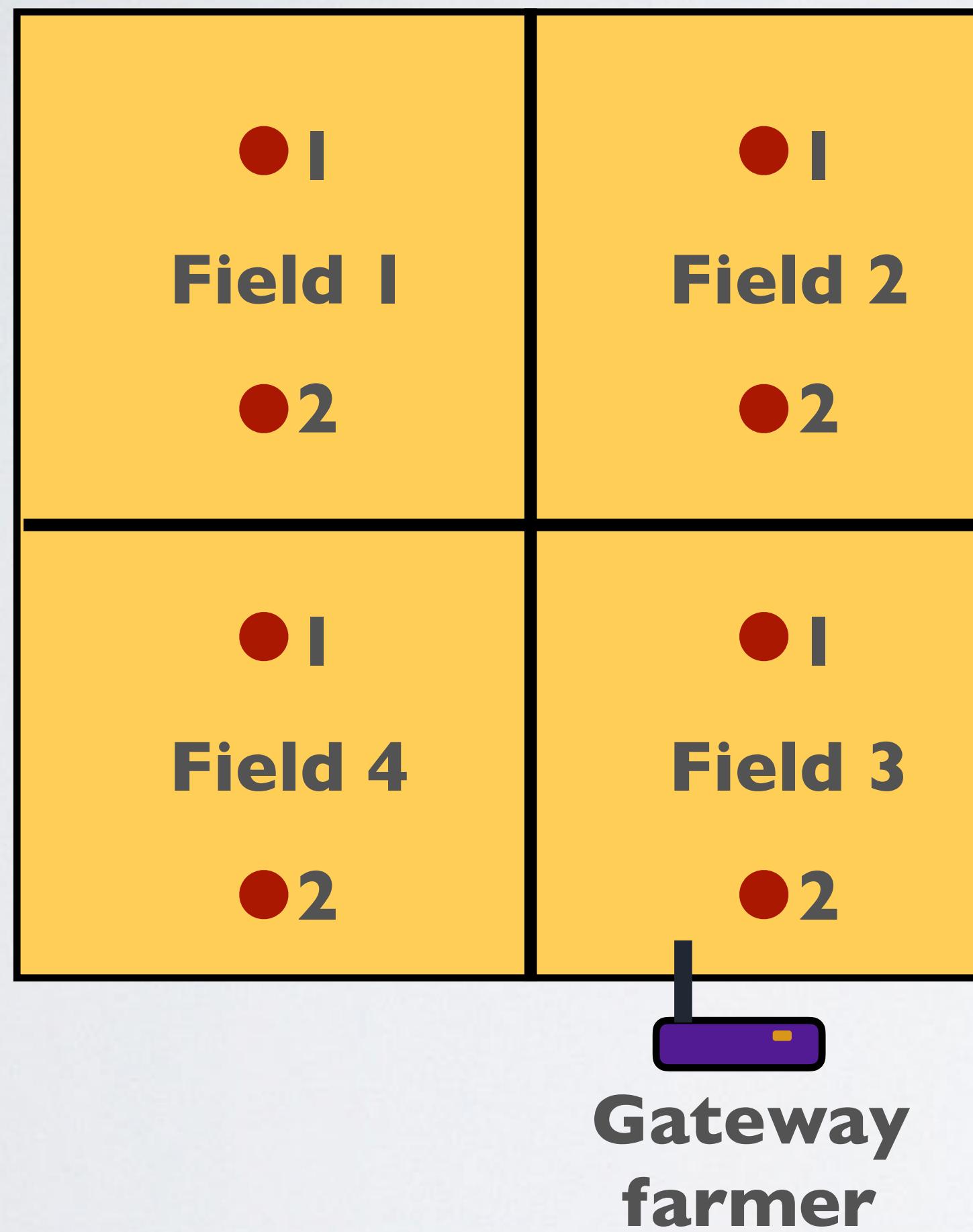
**Gateway
criminal**

Lets assume the criminal knows which area each fport value represents.

By monitoring the fport value the criminal will know in which area the bear is located.

EXAMPLE USING FPORT

- Potato farm with sensors measuring temperature and soil moisture.



The fport value is not interesting to hackers.
There is no monetary gain, if someone knows the fport values.

Field 1	- fport = 1
Field 2	- fport = 2
Field 3	- fport = 3
Field 4	- fport = 4

JSON

- JSON (JavaScript Object Notation) is a syntax for storing and exchanging data.
- This is a json object:

```
{  
  "device": 2,  
  "id": "ab123",  
  "temperature": [12, 56],  
  "location": {  
    "lat": 12.44,  
    "lon": 45.88  
  },  
  "info": null,  
  "enable": true  
}
```
- The json object starts and ends with curly brackets.
- Data consists of key/value pairs separated by commas.
- **The key must be wrapped in double quotes.**
- The key and its value are separated by a colon.
- The value can be a number (12 or 23.56) , string ("Hello"), boolean (true/false), array ([“aa”, “bb”]) another json object ({...}), or the value null).

JAVASCRIPT OBJECT

- This is a Javascript object.

```
{  
  device: 2,  
  id: "ab123",  
  temperature: [12, 56],  
  location: {  
    lat: 12.44,  
    lon: 45.88  
  },  
  info: null,  
  enable: true  
}
```

- In Javascript objects the keys are not required to be wrapped in double quotes.
- There are other differences but I will not discuss these in this tutorial.

JAVASCRIPT OBJECT

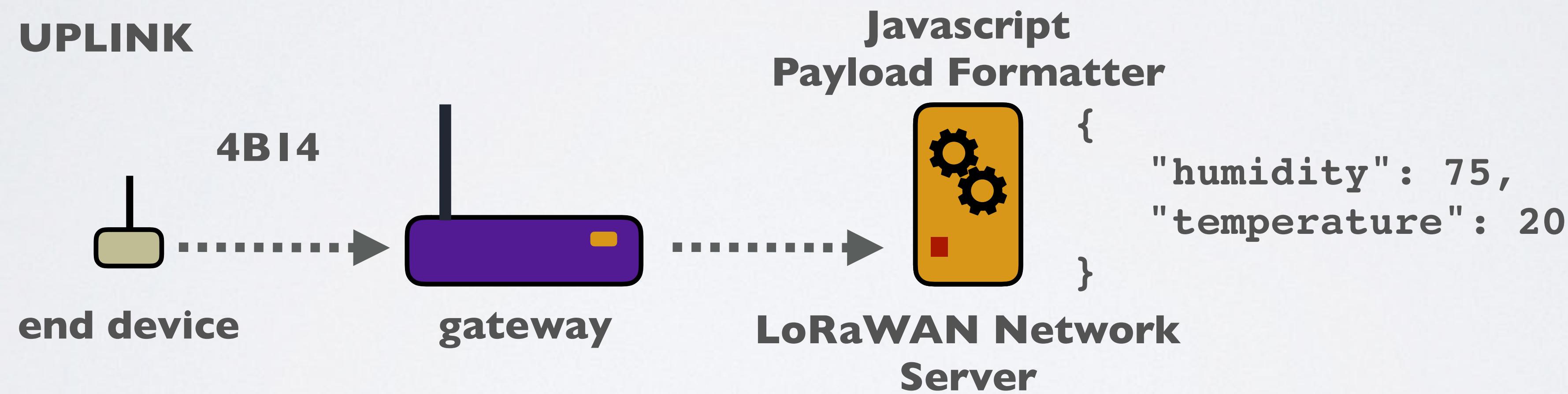
```
var sensor =  
{  
    device: 2,  
    id: "ab123",  
    temperature: [12, 56],  
    location: {  
        lat: 12.44,  
        lon: 45.88  
    },  
    info: null,  
    enable: true  
}
```

To create the sensor Javascript object:

```
var sensor = {};  
sensor.device = 2;  
sensor.id = "ab123";  
var temp_arr = [];  
temp_arr[0] = 12;  
temp_arr[1] = 56;  
sensor.temperature = temp_arr;  
var coord = {};  
coord.lat = 12.44;  
coord.lon = 15.88;  
sensor.location = coord;  
sensor.info = null;  
sensor.enable = true;
```

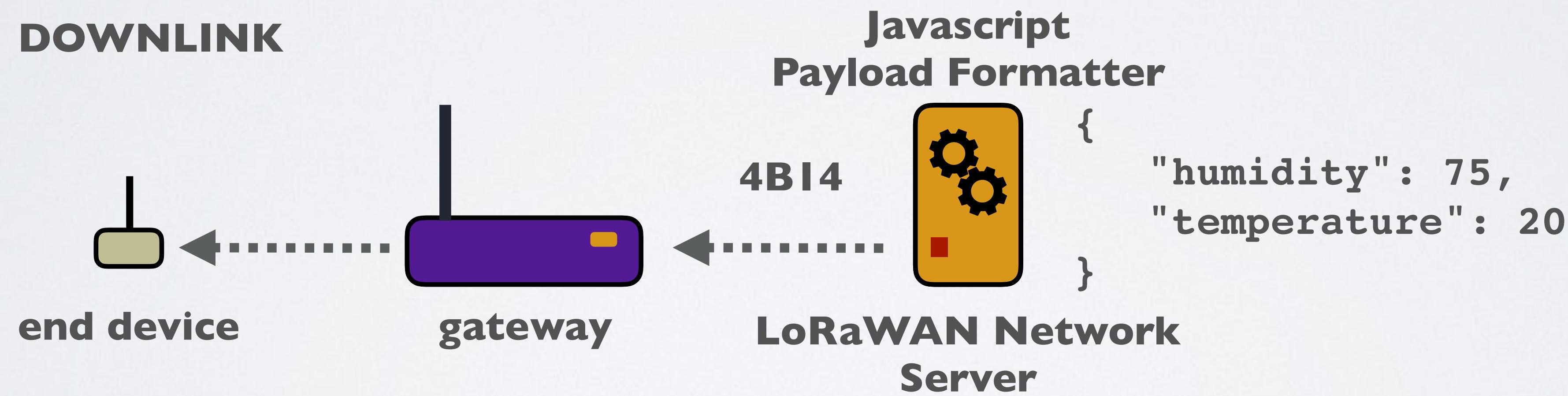
WHAT ARE PAYLOAD FORMATTERS

- Payload formatters allow you to process data going to and from end devices.
- When a binary payload is sent from end device to LoRaWAN network (uplink), the data can be converted using, for example, a Javascript payload formatter. The data can be converted to human readable data.



WHAT ARE PAYLOAD FORMATTERS

- When a message is sent from the LoRaWAN network to the end device (downlink), the message can be converted using, for example, a Javascript payload formatter. The message can be converted from human readable data to binary data.



THE V2 AND V3 PAYLOAD FORMATTERS ARE DIFFERENT

- The payload formatters used in the V2 console can not be copy-and-pasted directly in the V3 console.
- The V2 and V3 payload formatters are different.
- For example, the function names and the function parameters are different.
- In this tutorial I will only focus on the payload formatters used in the V3 console.

JAVASCRIPT PAYLOAD FORMATTER (UPLINK)

```
function Decoder(bytes, port) {
  if(bytes.length == 1) {
    if(bytes[0] == 1) {
      return {
        'button': 'activated'
      }
    } else {
      return {
        'error': 'button action unknown'
      }
    }
  } else if(bytes.length == 4) {
    var humidity = (bytes[0]<<8) | bytes[1];
    var temperature = (bytes[2]<<8) | bytes[3];
    return {
      'humidity': humidity/100,
      'temperature': temperature/100
    }
  } else {
    return {
      'error': 'payload unknown'
    }
  }
}
```

V2

```
function decodeUplink(input) {
  var data = {};
  var warnings = [];
  var errors = [];

  if(input.bytes.length == 1) {
    if(input.bytes[0] == 1) {
      data.button = "activated";
    } else {
      errors.push("button action unknown");
    }
  } else if(input.bytes.length == 4) {
    var humidity = (input.bytes[0]<<8) | input.bytes[1];
    data.humidity = humidity/100;
    var temperature = (input.bytes[2]<<8) | input.bytes[3];
    data.temperature = temperature/100;
  } else {
    errors.push("payload unknown");
  }

  return {
    data: data,
    warnings: warnings,
    errors: errors
  };
}
```

V3

JAVASCRIPT PAYLOAD FORMATTER (UPLINK)

- Decoder example (V2)

https://www.mobilefish.com/download/lora/tutorial_26_decodertxt

- decodeUplink example (V3)

https://www.mobilefish.com/download/lora/tutorial_53_decodeuplink.txt

PAYLOAD FORMATTERS

- In the V3 console several payload formatters can be used:
 - Javascript payload formatters
<https://www.thethingsindustries.com/docs/integrations/payload-formatters/javascript>
 - Cayenne Low Power Payload formatters
<https://www.thethingsindustries.com/docs/integrations/payload-formatters/cayenne/>
 - Device specific payload formatters
Device manufacturers may have payload formatters designed to work with their devices.
<https://github.com/TheThingsNetwork/lorawan-devices>
- I will only focus on the Javascript payload formatters.

PAYOUT FORMATTERS

- The V3 payload formatters can be applied to an entire application, or to a specific end device.

The screenshot shows the The Things Stack interface with the following navigation path: Applications > youtube-demo-app2 > End devices > youtube-demo-device. The interface includes sections for Overview, End devices, Live data, Payload formatters, Integrations, and Applications. A red box highlights the 'youtube-demo-app2' application name. Another red box highlights the 'youtube-demo-device' end device name. A large red arrow points from the 'Payload formatters' section under the end device to the 'Payload formatters' section in the sidebar under the application. A third red arrow points from the 'Payload formatters' section under the application to the 'Payload formatters' section under the end device. The text 'end device specific' is overlaid near the end device's payload formatters, and the text 'entire application' is overlaid near the application's payload formatters.

THE THINGS NETWORK THE THINGS STACK Community Edition Overview Applications Gateways Organizations EU1 Community No SLA applicable

Applications > youtube-demo-app2 > End devices > youtube-demo-device

youtube-demo-app2

youtube-demo-device ID: youtube-demo-device

Last seen 10 seconds ago ↑ 16 ↓ 1 Created 2 hours ago

Overview Live data Messaging Location Payload formatters Claiming General settings

General information End device ID youtube-demo-device

Live data See all activity →

↑ 12:16:45 Forward uplink data message Payload: { hu }

↑ 12:15:40 Forward uplink data message Payload: { hu }

Payload formatters

Integrations

entire application

end device specific

PAYLOAD FORMATTERS

- Depending on the selected payload formatter (entire application or end device specific) you will see 5 or 6 formatter type options.

entire application

Formatter type *

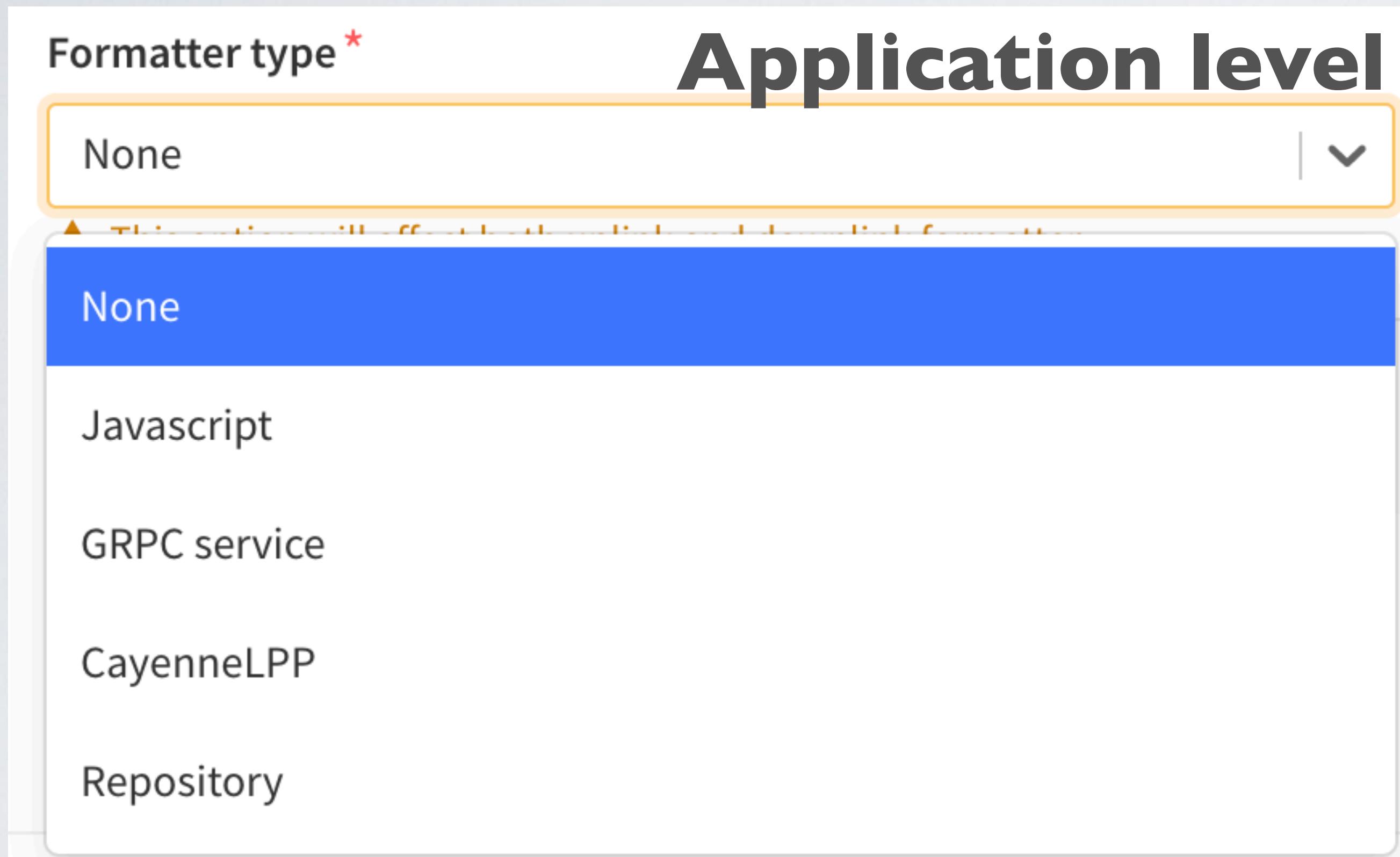
- None
- None
- Javascript
- GRPC service
- CayenneLPP
- Repository

end device specific

Formatter type *

- Use application payload formatter
- Use application payload formatter
- None
- Javascript
- GRPC service
- CayenneLPP
- Repository

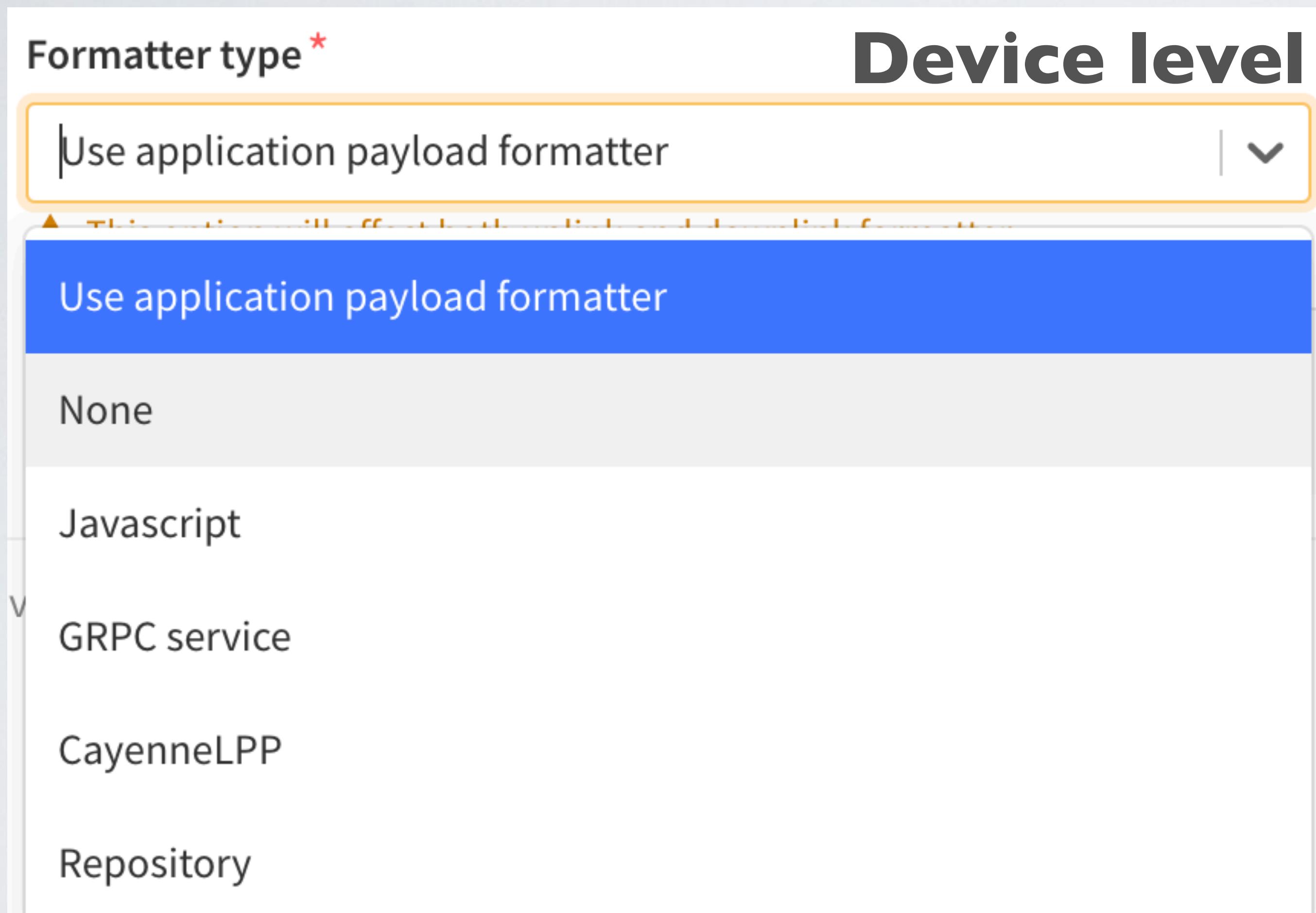
PAYLOAD FORMATTERS



On the **application** level these are the formatter type options.

When selecting None any entered payload formatter is removed and replaced with the default payload formatter.

PAYLOAD FORMATTERS



On the **device** level these are the formatter type options.

When both device and application payload formatter are specified the device payload formatter is used. But when you select the option “Use application payload formatter”, the application payload formatter is used.

PAYLOAD FORMATTERS

- It is recommended to use application payload formatters and only use device payload formatters when there is a need.
- If your application has two or more LoRaWAN end devices and they all transmit the same kind of sensor data, of course, you should specify the payload formatter on the application level.

PAYLOAD FORMATTER TESTER

The screenshot shows the 'Payload formatters' tab selected in the navigation bar. A modal window titled 'Test' is open, showing a successful payload decoding. The byte payload '09 60 07 D0' is mapped to FPort 1. The decoded JSON object is: { "humidity": 24, "temperature": 20 }.

youtube-demo-device
ID: youtube-demo-device

Last seen 55 seconds ago ↑ 68 ↓ 3
Created 3 hours ago

Overview Live data Messaging Location Payload formatters Claiming General settings

Uplink Downlink

Setup

Formatter type *
Javascript

Formatter parameter *

```
function decodeUplink(input) {
  var data = {};
  var warnings = [];
  var errors = [];

  if(input.bytes.length == 1) {
    if(input.bytes[0] == 1) {
      data.button = "activated";
    } else {
      errors.push("button action unknown");
    }
  } else if(input.bytes.length == 4) {
    var humidity = (input.bytes[0]<<8) | input.bytes[1];
    data.humidity = humidity/100;
    var temperature = (input.bytes[2]<<8) | input.bytes[3];
    data.temperature = temperature/100;
  }
}
```

Save changes Test decoder

The payload formatter tester is only available on the device level and not on the application level.

NO PAYLOAD FORMATTER USED

- When no payload formatter is used:

The screenshot shows the The Things Stack Community Edition interface. The top navigation bar includes links for Overview, Applications (which is selected), Gateways, Organizations, and EU1 Community (with a note about fair use policy). On the left, there's a sidebar with icons for Overview, End devices, and a specific application named youtube-demo-app2. The main content area displays a table of live data messages. The columns are Time, Entity ID, Type, Data preview, Verbose stream (checkbox), and Pause (button). Two messages are listed: one from 17:33:08 and another from 17:32:01. Both messages are of type "Forward uplink data message". In the "Data preview" column, the first message shows a MAC payload of 09 60 07 D0, and the second message shows a MAC payload of 0C 1C 07 D0. A red arrow points to the MAC payload bytes in the second row.

Time	Entity ID	Type	Data preview	Verbose stream	Pause
↑ 17:33:08	youtube-demo-...	Forward uplink data message	MAC payload: 09 60 07 D0		
↑ 17:32:01	youtube-demo-...	Forward uplink data message	MAC payload: 0C 1C 07 D0		

**only received payload
(bytes)**

NO PAYLOAD FORMATTER USED

Applications > youtube-demo-app2 > Live data

Time	Entity ID	Type	Event details
↑ 17:35:21	youtube-demo-...	Forward u	39 40 41 42 43 44 45 46 47 48 49 50 51 52
↑ 17:34:15	youtube-demo-...	Forward u	"gs:up:host:01F6M1PHMP1CSAN0M14SXV3MJB", "gs:uplink:01F6SSQ4S5RSVEJP7BKNYYDXXP", "ns:uplink:01F6SSQ4S6A30FDHSK1ZYMKKV1", "rpc:/ttn.lorawan.v3.GsNs/HandleUplink:01F6SSQ4 "rpc:/ttn.lorawan.v3.NsAs/HandleUplink:01F6SSQ4 ,
↑ 17:33:08	youtube-demo-...	Forward u	"received_at": "2021-05-28T15:33:08.215928905Z", "uplink_message": {
↑ 17:32:01	youtube-demo-...	Forward u	"session_key_id": "AXmzmno40bAjcg5V4Gr+LA==", "f_port": 1, "f_cnt": 1, "frm_payload": "CWAH0A==", "rx_metadata": [
↪ 17:31:56	youtube-demo-...	Accept jc	{
↪ 17:30:48	youtube-demo-...	Accept jc	}

No formatted payload

PAYLOAD FORMATTER IS USED

- When a payload formatter is used:

The screenshot shows the The Things Stack Community Edition interface. The top navigation bar includes links for THE THINGS NETWORK, THE THINGS STACK Community Edition, Overview, Applications (which is selected), Gateways, Organizations, EU1 Community (No SLA applicable), and a user profile. The main content area displays the 'Applications > youtube-demo-app2 > Live data' view. On the left, there's a sidebar with 'youtube-demo-app2' selected, and options for 'Overview' and 'End devices'. The main table lists two data messages:

Time	Entity ID	Type	Data preview	Verbose stream	Pause	Clear
↑ 12:48:09	youtube-demo-de...	Forward uplink data message	Payload: { humidity: 24, temperature: 20 }	09 60 07 D0	FPort: 1 SNR: 8	
↑ 12:47:02	youtube-demo-de...	Forward uplink data message	Payload: { humidity: 24, temperature: 20 }	09 60 07 D0	FPort: 1 SNR: 8	

**payload
formatted**

**received
payload
(bytes)**

PAYLOAD FORMATTER IS USED

Applications > youtube-demo-app2 > Live data

Time	Entity ID	Type
↑ 12:58:09	youtube-demo-de...	Forward uplink data
↑ 12:57:02	youtube-demo-de...	Forward uplink data
↑ 12:55:56	youtube-demo-de...	Forward uplink data
↑ 12:54:49	youtube-demo-de...	Forward uplink data
↑ 12:53:44	youtube-demo-de...	Forward uplink data

Event details

```
44 ] ,  
45 "received_at": "2021-05-25T10:47:02.611022456Z",  
46 "uplink_message": {  
47     "session_key_id": "AXmi9no9i1M57A7Bu7c5Ug==",  
48     "f_port": 1,  
49     "f_cnt": 46,  
50     "frm_payload": "CWAH0A==",  
51     "decoded_payload": {  
52         "humidity": 24,  
53         "temperature": 20  
54     },  
55     "rx_metadata": [  
56         {
```

Formatted payload

UPLINK JAVASCRIPT PAYLOAD FORMATTER

UPLINK JAVASCRIPT PAYLOAD FORMATTER (V3)

```
fPort = 8  
sensor data = 010A
```

```
input = {  
  "fPort": 8,  
  "bytes": [1,10]  
}
```

```
function decodeUplink(input) {  
  data = {};  
  data.led = input.bytes[0];  
  data.color = input.bytes[1];  
  data.port = input.fPort;  
  return {  
    data: data  
  };  
}
```

```
{  
  led: 1,  
  color: 10,  
  port: 8  
}
```

Formatted data displayed in console.

UPLINK JAVASCRIPT PAYLOAD FORMATTER (V3)

```
TX sensor  
data = 0A0B  
fPort = 5
```

```
input = {  
  "fPort": 4,  
  "bytes": [11,12]  
}
```

```
function decodeUplink(input) {  
  :  
  return {  
    data: {...},  
    warnings: ["warning1", "warning2"],  
    errors: ["error1", "error2"]  
  }  
}
```

UPLINK JAVASCRIPT PAYLOAD FORMATTER (V3)

```
input = {  
  "fPort": 4,  
  "bytes": [9,96,7]  
}
```

```
Tx sensor  
data:  
096007  
fPort:4
```

```
function decodeUplink(input) {  
  var dt = {};  
  dt.port = input.fPort;  
  dt.val1 = input.bytes[0];  
  dt.val2 = input.bytes[1];  
  dt.val3 = input.bytes[2];  
  return {  
    data: dt  
  };  
}
```

```
{  
  port: 4,  
  val1: 9,  
  val2: 96,  
  val3: 7,  
}
```

returned object

The decodeUplink function has only one function parameter (input) which is a json object.

The json object is automatically created and always contains two key/value pairs.
fPort contains the fport value.
When binary sensor data is transmitted (eg: 096007) this data is automatically converted to an array of separate bytes each in its decimal representation (eg: [9,96,7]) and is assigned to the bytes key in the json object.

UPLINK JAVASCRIPT PAYLOAD FORMATTER (V3)

```
input = {  
  "fPort": 4,  
  "bytes": [9,96,7]  
}
```

```
Tx sensor  
data:  
096007  
fPort:4
```

```
function decodeUplink(input) {  
  var dt = {};  
  dt.port = input.fPort;  
  dt.val1 = input.bytes[0];  
  dt.val2 = input.bytes[1];  
  dt.val3 = input.bytes[2];  
  return {  
    data: dt  
  };  
}
```

```
{  
  port: 4,  
  val1: 9,  
  val2: 96,  
  val3: 7,  
}
```

returned object

The payload formatter outputs a javascript (JS) object. The JS object can be empty {} or contains:

- a data key with a javascript object as its value,
- an errors key with an array of error messages as its value,
- a warnings key with an array of warning messages as its value.

```
{  
  data: {...},  
  errors: [...],  
  warnings: [...]  
}
```

BYTE PAYLOAD IS CONVERTED TO DECIMAL VALUES

Setup

Formatter type *

Javascript

Formatter parameter *

```
1 function decodeUplink(input) {  
2     var dt = {};  
3     dt.port = input.fPort;  
4     dt.val1 = input.bytes[0];  
5     dt.val2 = input.bytes[1];  
6     dt.val3 = input.bytes[2];  
7     return {  
8         data: dt  
9     };  
10}
```

Test

Byte payload

09 60 07

hex values

FPort

4

 Payload is valid

```
{  
    "port": 4,  
    "val1": 9,  
    "val2": 96,  
    "val3": 7  
}
```

```
input = {  
    "fPort": 4,  
    "data": [9,96,7]  
}
```

The hex values are automatically converted into an array of decimal values:

JAVASCRIPT PAYLOAD FORMATTER (UPLINK) EXAMPLE I

```
function decodeUplink(input) {
  var dt = {};
  var warn = [];
  var err = [];
  if (input.fPort == 2) {
    warn.push("Please use port 1");
  } else if(input.fPort == 3) {
    err.push("Do not use port 3");
  } else {
    dt.val = input.bytes[0] + input.bytes[1];
    dt.port = input.fPort;
  }

  return {
    warnings: warn,
    errors: err,
    data: dt
  };
}
```

```
input = {
  "fPort": 1,
  "bytes": [01, 02]
}
```

```
input = {
  "fPort": 2,
  "bytes": [01, 02]
}
```

```
input = {
  "fPort": 3,
  "bytes": [01, 02]
}
```

Test

Byte payload FPort

01 02 1

✓ Payload is valid

```
{
  "port": 1,
  "val": 3
}
```

Test

Byte payload FPort

01 02 2

⚠ Please use port 1

```
{}
```

Test

Byte payload FPort

01 02 3

❗ Do not use port 3

```
{}
```

If a warning is present in `warnings`, the warning is displayed. The payload is valid and the message will not be dropped.
 If an error is present in `errors`, the error is displayed. The payload is invalid and the message will be dropped.

JAVASCRIPT PAYLOAD FORMATTER (UPLINK) EXAMPLE 2

```
var directions = ["N", "E", "S", "W"];
function decodeUplink(input) {
  switch (input.fPort) {
    case 1:
      return {
        // Decoded data
        data: {
          direction: directions[input.bytes[0]],
          speed: input.bytes[1]
        }
      }
    default:
      return {
        errors: ["unknown FPort"]
      }
  }
}
```

```
input = {
  "fPort": 1,
  "bytes": [1, 98]
}
```

```
input = {
  "fPort": 2,
  "bytes": [1, 98]
}
```

```
input = {
  "fPort": 1,
  "bytes": [3, 10]
}
```

The figure consists of three vertically stacked screenshots of the mobilefish.com payload formatter tool. Each screenshot shows a 'Test' section with a 'Byte payload' input field containing two hex digits, an 'FPort' selection dropdown, and a 'Payload is valid' status indicator.

- Screenshot 1 (FPort 1):** The 'Byte payload' is '01 62'. The 'Payload is valid' status is green with a checkmark. The decoded payload is shown as a JSON object: { "direction": "E", "speed": 98 }.
- Screenshot 2 (FPort 2):** The 'Byte payload' is '01 62'. The 'Payload is valid' status is red with a warning exclamation mark. The message 'Unknown FPort' is displayed below the input fields.
- Screenshot 3 (FPort 1):** The 'Byte payload' is '03 0A'. The 'Payload is valid' status is green with a checkmark. The decoded payload is shown as a JSON object: { "direction": "W", "speed": 10 }.

If a warning is present in warnings, the warning is displayed. The payload is valid and the message will not be dropped.
 If an error is present in errors, the error is displayed. The payload is invalid and the message will be dropped.

DLINK JAVASCRIPT PAYLOAD FORMATTER

DLINK JAVASCRIPT PAYLOAD FORMATTER (V3)

```
fPort = 8
```

```
{
  "button": 2
}
```

```
input = {
  "data": {
    "button": 2
  },
  "fPort": 8
}
```

```
{
  "bytes": [2],
  "fPort": 8
}
```

Send to end device

```
function encodeDownlink(input) {
  :
  return {
    bytes: [input.data.button],
    fPort: input.fPort
  };
}
```

```
function decodeDownlink(input) {
  var data = {};
  :
  data.button = input.bytes[0];
  return {
    data: data
  };
}
```

The encodeDownlink and decodeDownlink functions are inverses of each other.

**encodeDownlink is required
decodeDownlink is optional**

```
{
  button: 2
}
```

Formatted data displayed in console.

DLINK JAVASCRIPT PAYLOAD FORMATTER (V3)

```
fPort = 4  
payload = {  
  "led": 1,  
  "color": 3  
}
```

```
input = {  
  "fPort": 4,  
  "data": {  
    "led": 1,  
    "color": 3  
  }  
}
```

```
function encodeDownlink(input) {  
  :  
  return {  
    bytes: [input.data.led, input.data.color],  
    fPort: input.fPort,  
    warnings: ["warning1", "warning2"],  
    errors: ["error1", "error2"]  
  }  
}
```

```
{  
  bytes: [1,3],  
  fPort: 4  
}
```

DLINK JAVASCRIPT PAYLOAD FORMATTER (V3)

```
encodeDownlink  
output converted  
to json input  
object  
  
input = {  
  "bytes": [1,3],  
  "fPort": 4  
}
```

```
function decodeDownlink(input) {  
  :  
  return {  
    data: {...},  
    warnings: ["warning1", "warning2"],  
    errors: ["error1", "error2"]  
  }  
}
```

If you specify a downlink formatter the encodeDownlink function is required but not the decodeDownlink function. The decodeDownlink formats the binary data “bytes” into human readable data, just like the decodeUplink function.

DLINK JAVASCRIPT PAYLOAD FORMATTER (V3)

```
fPort = 5
```

```
{  
  "color": "green"  
}
```

1

The encodeDownlink function has only one function parameter (input) which is a json object.

```
input = {  
  "data": {  
    "color": "green"  
  }  
  "fPort": 5  
}
```

2

The user creates a json object (1). This object is assigned to the data key of the input json object (2). This is done automatically

```
function encodeDownlink(input) {  
  var colors = ["red", "green",  
    "blue"];  
  
  return {  
    bytes: [colors.indexOf(  
      input.data.color)],  
    fPort: input.fPort  
  };  
}
```

The encodeDownlink has access to the input object.

```
{  
  bytes: [1],  
  fPort: 5  
}
```

DLINK JAVASCRIPT PAYLOAD FORMATTER (V3)

Setup

Formatter type *

Javascript

Formatter parameter *

```
1 function encodeDownlink(input) {  
2     var colors = ["red", "green", "blue"];  
3     return {  
4         bytes: [colors.indexOf(input.data.color)],  
5         fPort: input.fPort  
6     };  
7 }
```

```
input = {  
    "data": {  
        "color": "green"  
    },  
    "fPort": 5,  
}
```

Test

JSON payload

```
1 {  
2     "color": "green"  
3 }
```

FPort

5

Payload is valid

01

DLINK JAVASCRIPT PAYLOAD FORMATTER (V3)

Setup

Formatter type *

Javascript

Formatter parameter *

```
1 function encodeDownlink(input) {  
2     var colors = ["red", "green", "blue"];  
3     return {  
4         bytes: [colors.indexOf(input.data.color)],  
5         fPort: input.fPort,  
6         errors: ['aaaa']  
7     };  
8 }
```

Test

JSON payload

```
1 {  
2     "color": "blue"  
3 }
```

FPort

10

If a warning is present in warnings, the warning is displayed.

The payload is valid and the message will not be dropped.

If an error is present in errors, the error is displayed.
The payload is invalid and the message will be dropped.

! Aaaa

DLINK JAVASCRIPT PAYLOAD FORMATTER (V3)

Setup

Formatter type *

Javascript

Formatter parameter *

```
1 function encodeDownlink(input) {  
2     var colors = ["red", "green", "blue"];  
3     return {  
4         bytes:[colors.indexOf(input.data.color)],  
5         fPort: input.fPort,  
6         warnings: ['bbbb']  
7     };  
8 }
```

Test

JSON payload

```
1 {  
2     "color": "blue"  
3 }
```

FPort

10

If a warning is present in
warnings, the warning is
displayed.

The payload is valid and the
message will not be dropped.

If an error is present in
errors, the error is displayed.
The payload is invalid and
the message will be dropped.

⚠ bbbb

02

JS PAYLOAD FORMATTER (DOWNLINK) EXAMPLE I

Setup

Formatter type *

Javascript

Formatter parameter *

```
1 function encodeDownlink(input) {  
2     return {  
3         bytes: [1, 2, 3],  
4         fPort: 1  
5     }  
6 }
```

```
input = {  
    "data": {},  
    "fPort": 4,  
}
```

Test

JSON payload

```
1 { } 4
```

 Payload is valid

01 02 03

JS PAYLOAD FORMATTER (DOWNLINK) EXAMPLE 2

```
fPort = 8
```

```
{  
  "fan": "low"  
}
```

```
input = {  
  "data": {  
    "fan": "low"  
  },  
  "fPort": 8  
}
```

```
function encodeDownlink(input) {  
  var out = [];  
  var fan_position = ["low", "medium", "high"];  
  if (input.data.fan == fan_position[1]) {  
    out.push(11); //medium 0B  
  } else if (input.data.fan == fan_position[2]) {  
    out.push(12); //high 0C  
  } else {  
    out.push(10); //low 0A  
  }  
  return {  
    bytes: out,  
    fPort: input.fPort  
  };  
}
```

```
{  
  bytes: [10],  
  fPort: 8  
}
```

JS PAYLOAD FORMATTER (DOWNLINK) EXAMPLE 2

Setup

Formatter type *

Javascript

Formatter parameter *

```
1 function encodeDownlink(input) {
2     var out = [];
3     var fan_position = ["low", "medium", "high"];
4     if (input.data.fan == fan_position[1]) {
5         out.push(11); //medium 0B
6     } else if (input.data.fan == fan_position[2]) {
7         out.push(12); //high 0C
8     } else {
9         out.push(10); // low 0A
10    }
11    return {
12        bytes: out,
13        fPort: input.fPort
14    };
15 }
```

Test

JSON payload

```
1 {
2     "fan": "low"
3 }
```

FPort

8

 Payload is valid

0A