Steady ground station USB interface protocol overview + Fluctus 1.7b firmware data parsing architecture

May 23rd, 2025

1 Overview of the radio ⇄ USB data path

- The flight computer (Fluctus) fills a **binary telemetry buffer** and calls the sending.
- Fluctus appends a 16-bit additive checksum and transmits the radio frame.
- The **ground station** verifies that checksum, strips it, converts the remaining bytes to a printable representation and pushes a **single ASCII line** to its USB-CDC port:

```
F B <hex ...>|Grssi-65/Gsnr6\n
^ ^    ^          ^
│ │    │          └─ diagnostics added by the ground station
│ │    └─ all remaining telemetry bytes rendered as two-digit hex
│ └─ packet-type byte  ('B' = binary, 'C' = ASCII string)
└─ radio callsign of the packet origin
```

The checksum is never exposed on USB—the PC application can trust the payload.

---

2 Starting and stopping the ground station

2.1 *start* command

Send **one line** to the ground-station serial port:

start <band><chan><chan> Fluctus \n

Field	Size	Allowed values	Purpose
band	1	0 = 902–928 MHz (US) 1 = 863–870 MHz (EU)	Regional ISM band
chan	2	00…25 or A…Z → (902.5 MHz + chan × 1 MHz)	Centre frequency
device	7	Fluctus	must match the flight computer
\n	1	line feed	terminator

Example (US, channel 03):

start003Fluctus\n

Ground-station reply

Gstartok123 <- “123” is the station firmware ID

After that handshake it forwards every LoRa packet it receives.

2.2 *ping* and *startf* commands

Command line (USB)	In-air effect	Down-link indication
ping\n	queued and wrapped into a G packet on the next RX window	flight computer returns FCpong (type C)
startf\n	arms the flight sequencer	status field in the next telemetry frame changes from **0 (IDLE)** to **1 (ARMED)** – there is **no explicit string reply**

All commands are ASCII, case-sensitive and must end with \n.

---

3 Binary telemetry frame (“B”)

Little-endian unless noted.

3.1 Field list & scaling

The firmware’s encoded types.
The table cross-references that mnemonic, the internal variable, and the factor required to obtain engineering units.

#	Name (indexConfig)	Enc. type	Size (B)	Firmware variable	Unit / range	Decode formula
0	**uid**	i16	2	fluctusUID	–	16-bit signed
1	**fw**	i16	2	fwver	–	16-bit signed
2	**rx**	i8	1	rxPacketCount	–	8-bit signed
3	**timeMPU**	i32	4	_time	ms	int32
4	**status**	i8	1	status	see §3.3	enum
5	**altitude**	i24	3	sf_altitude	m	24-bit signed
6	**speedVert**	i16	2	sf_speedvert	$m \cdot s^{-1}$	int16
7	**accel**	f16	2	accelGlob	$m \cdot s^{-2}$	int16 / 10
8	**angle**	ui8	1	angle	deg	0–255
9	**battVoltage**	i16	2	realBattVoltage	mV	int16
10	**time**	f16	2	flightTime	s	int16 / 10
11	**pyroStates**	i8	1	pyrostatus	see §3.4	bitfield
12	**logStatus**	i8	1	logstatus	% free (0–100) or 101 = OFF	int8
13	**gpsLat**	gps	4	gpsLat	° WGS84	int32 / 1 000 000
14	**gpsLng**	gps	4	gpsLng	° WGS84	int32 / 1 000 000
15	**gpsState**	i8	1	gpsState	0–5	int8
16	**warnCode**	i8	1	warncode	bit-mask	int8
17	**message**	msg	4	rolling statistics	see §3.2	custom
18	**userIn1***	i16	2	userInputVoltage_in1	mV	int16
19	**userIn2***	i16	2	userInputVoltage_in2	mV	int16

* The last two fields are **optional**; they are present only when configured by the user in FCC. Parse them **only if the USB payload is at least 42 bytes** (after stripping F B).

3.2 Rolling message (field 17 – 4 B)

Every time a packet it sent, the message rotates through three statistics:

Telemetry-counter (mod 3)	ID byte (message[0])	Encoded value
0	'A' (0x41)	sf_maxAltitude
1	'S' (0x53)	sf_maxSpeedvert
2	'G' (0x47)	maxAccelGlob

Encoding

Byte 0 : ID ('A','S','G')

Byte 1 : LSB of value×10

Byte 2 : Mid

Byte 3 : MSB

Signed 24-bit integer, little-endian.
To decode:

```
raw  = b1 | (b2<<8) | (b3<<16)
if raw & 0x800000:      # sign-extend
   raw |= 0xFF000000
value = raw
```

Example – bytes 41 04 1E 00
→ ID 'A', raw = 0x001E04 = 7684 → **7684 m** maximum altitude.

3.3 Flight-sequencer status codes (field 4)

Code	Meaning	Typical phase
0	**IDLE**	powered, disarmed
1	**ARMED**	safety pin removed, still idle
2	**COUNTDOWN ENGAGED**	countdown running
3	**WAITING FOR LAUNCH**	countdown done, hold-off until launch-detect
4	**ASCENT**	motor burning / coasting up
5	**DESCENT**	after apogee
6	**TOUCHDOWN**	motionless, flight complete

3.4 Pyrotechnic output bitmap (field 11)

Bit 1-0 : Pyro A

Bit 3-2 : Pyro B

Bit 5-4 : Pyro C

Per-output value:

Enc.	Meaning (original sequencer value)
0	**DISABLED** (no continuity)
1	**CONTINUITY** (disabled but wire intact)
3	**ENABLED / FIRED** (original value 10 maps to 3)

Bits 6-7 are unused and always 0.

---

4 Worked example

USB line captured:

FB3E00070100BEDD01000000000000006C00AA89109CFF00650000000000000000000E53000000|Grssi-65/Gsnr6

1. Split at the pipe (|).
2. Discard the first two chars (F, B). The remaining hex string is 80 characters → 40 bytes.
3. Convert to a byte array and feed the table above.

Partial outcome

Name	Raw hex	Decoded
uid	3E00	62
fw	0701	0x0107 → 263
timeMPU	BEDD0100	122 046 ms
status	00	**IDLE**
altitude	000000	0 m
…	…	…

Because the frame length is 40 B the *optional* user inputs are absent (they would extend to 44 B).

---

5 Implementing a parser (C#/Python/…)

1. After USB read, **trim CR/LF**, locate the '|', and split.
2. Verify that the first character is 'F'; second char is the *type*.
3. For type 'B' remove the two header letters and call HexToBytes().
4. Walk the byte array with the field list in §3.1.
    - Sign-extend **i24** (see code below).
    - Divide **f16** by 10, **gps** by 1 000 000.
    - Decode **msg**, **status**, **pyroStates** using §3.2 – §3.4.
5. If bytes remain after field 17, read **userIn1/2** (two i16 each).

```
static int ReadInt24(byte[] buf, int ofs)
{
  int val = buf[ofs] | (buf[ofs+1]<<8) | (buf[ofs+2]<<16);
  if ((val & 0x800000) != 0) val |= unchecked((int)0xFF000000);
  return val;
}
```

---

6 Ground-station command summary

Command (USB)	Purpose	Flight-computer reaction
start<band><chan><chan>Fluctus\n	Initialise RF link	Returns Gstartok<fw>
ping\n	Connectivity test	Sends back FCpong (type C)
startf\n	Arm sequencer	status field changes 0 → 1

Appendix – Type mnemonic cheat-sheet

Mnemonic	Meaning	Scaling in this firmware
i8 / ui8	1-byte signed / unsigned	none
i16	2-byte signed	none
i24	3-byte signed	none
i32	4-byte signed	none
f16	2-byte signed fixed-point	divide by 10
gps	4-byte signed fixed-point	divide by 1 000 000
msg	1 B ID + 24-bit value	divide by 10
b	Boolean	0 / 1
c	ASCII char	–