GROUND Communications Protocol Specification

Version 1.2.2

B.B.F.M. Verspaandonk

December 16, 2024

Contents

1 Overview

The GROUND (GAIA Radio OUtput Network Delivery) protocol is designed for transmitting data from the Cansat to the ground station. All data is serialized in little-endian format, meaning the least significant byte is sent first. For example, the number `0x1234` would be transmitted as `0x34 0x12`.

2 Packet Structure

A packet comprises the following fields:

#	Field Name	Size
1	`magic_number`	4 bytes
2	`content_type`	2 bytes
3	`content_size`	2 bytes
4	`content`	`content_size` bytes

Field Descriptions

`magic_number`: A constant value `0x47414941` (ASCII for `GAIA`) that marks the start of a packet.

`content_type`: Specifies the type and structure of the data. Its two bytes are interpreted as follows:

- **First byte**:
  - High nibble (`0xF0`): Indicates which CRC checksum is included (`0x0` = none, `0x1` = CRC-8, `0x2` = CRC-16-IBM, `0x3` = CRC-32-ISO-HDLC. See Checksums).
  - Low nibble (`0x0F`): Specifies the data category (see Data Types). May never be `0x0`.
- **Second byte**:
  - High nibble (`0xF0`): Indicates whether the data is a single value (`0x0`) or an array (`0x1`).
  - Low nibble (`0x0F`): Specifies the data's primitive type (see Data Types).

`content_size`: The number of bytes in the `content` field.

`content`: The actual data payload. Its interpretation depends on `content_type`.

3 Handling `magic_number` in Content

If the `magic_number` sequence `0x47414941` appears in the `content`, it must be escaped by appending a `0x00` byte immediately after. For example:

`47 41 49 41` → `47 41 49 41 00`

The escape byte contributes to `content_size` but is excluded from array length calculations. It should be removed during packet parsing.

4 Data Types

Raw Data Types

Value	Type	Description
`0x00`	`u8`	Unsigned 8-bit integer
`0x01`	`u16`	Unsigned 16-bit integer
`0x02`	`u32`	Unsigned 32-bit integer
`0x03`	`u64`	Unsigned 64-bit integer
`0x04`	`s8`	Signed 8-bit integer
`0x05`	`s16`	Signed 16-bit integer
`0x06`	`s32`	Signed 32-bit integer
`0x07`	`s64`	Signed 64-bit integer
`0x08`	`float`	32-bit floating point number
`0x09`	`double`	64-bit floating point number
`0x0A`	`bool`	Boolean
`0x0B`	`char`	ASCII character

Categorical Data Types

Value	Type	Description
`0x01`	`GPS`	GPS coordinates
`0x02`	`G-force`	G-force measurement
`0x03`	`Angle`	Angle measurement
`0x04`	`Time`	GPS Time
`0x05`	`Age`	Time in ms since last gps fix
`0x06`	`HDOP`	Horizontal Dilution of Precision
`0x07`	`Satellites`	Number of satellites in view
`0x08`	`GPS Fail %`	Percentage of GPS cheksums failed
`0x09`	`CO2`*	CO_2 concentration in ppb

`0x0A`	`Temperature`	Temperature in °C
`0x0B`	`Pressure`	Pressure in Pa
`0x0C`	`Dust`*	Dust concentration in idk what
`0x0D`	`UV`*	UV radiation in idk what
`0x0E`	`Packet`	Packet number

* These sensors may not be present on the final Cansat.

5 Checksums

The checksum nibble can take on the following values:

Value	Checksum Type
`0x0`	None
`0x1`	CRC-8
`0x2`	CRC-16-IBM
`0x3`	CRC-32-ISO-HDLC

CRC-8

The following fields are used for calculating the CRC-8 checksum:

- Polynomial: `0x07`
- Initial value: `0x00`
- Final XOR value: `0x00`
- Reverse input: `false`
- Reverse output: `false`

CRC-16-IBM

The following fields are used for calculating the CRC-16-IBM checksum:

- Polynomial: `0x8005`
- Initial value: `0x0000`
- Final XOR value: `0x0000`
- Reverse input: `true`
- Reverse output: `true`

CRC-32-ISO-HDLC

The following fields are used for calculating the CRC-32-ISO-HDLC checksum:

- Polynomial: `0x04C11DB7`
- Initial value: `0xFFFFFFFF`
- Final XOR value: `0xFFFFFFFF`
- Reverse input: `true`
- Reverse output: `true`

6 Encoding order

The fields in a packet are encoded in the following order:

1. Check if the `magic_number` sequence appears in the `content` field. If so, escape it.
2. Calculate the content size and add 1, 2 or 4 bytes for the checksum.
3. Add the `magic_number` sequence, `content_type`, `content_size` and `content` fields.
4. Calculate the checksum and append it to the packet.
5. Transmit the packet.

7 Examples

Single Value

Packet encoding a single unsigned 16-bit integer with value `4660`:

```
47 41 49 41 01 01 02 00 34 12
```

Breakdown:

```
47 41 49 41 // Magic number
21          // Content type: Unsigned 16-bit integer
02 00       // Content size: 2 bytes
34 12       // Content: 4660
```

Array

Packet encoding GPS coordinates (latitude, longitude) as two 64-bit doubles:

```
47 41 49 41 11 19 11 00 8C 01 E2 36 9D B4 49 40 1B F1 90 7B 96 F4 15 40 6D
```

Breakdown:

```
47 41 49 41             // Magic number
11                      // Content category: GPS coordinates with CRC-8
19                      // Content type: Array of 64-bit doubles
10 00                   // Content size: 17 bytes
1F 8B 8C 0C F4 B6 49 40 // Latitude: 51.411047802309525
77 05 98 4A 64 D4 15 40 // Longitude: 5.488855295869395
6D                      // CRC-8 checksum
```

Escaped Magic Number

GPS coordinates with a `magic_number` sequence in the content:

```
47 41 49 41 21 19 13 00 47 41 49 41 00 F8 B6 49 40 10 61 4A 8F 35 D4 15 40 82 E8
```

Breakdown:

```
47 41 49 41                // Magic number
21                         // Content category: GPS coordinates with CRC-16
19                         // Content type: Array of 64-bit doubles
13 00                      // Content size: 17 bytes
47 41 49 41 00 F8 B6 49 40 // Latitude: 51.429451142090834 (with escaped magic number)
10 61 4A 8F 35 D4 15 40    // Longitude: 5.457235564150565
82 E8                      // CRC-16-IBM checksum
```