RadioControl-Protocol

Paul Nykiel

October 24, 2017

Contents

Chapter 1

Introduction

Fast and extensible communication over various serial-interfaces. Primarily designed for radio-controlled models, mesh capable.

1.1 Example Application

1.1.1 Trainer

Transmitter → Transmitter → Plane

1.1.2 Long Range

Long Range: Multiple Transmitters, Drone as Repeater. TCP for Transmitters.

Chapter 2

Protocol

The protocol is designed for serial interfaces, the data is binary encoded to optimize for slow connections. The protocol consists of packages, one package contains all channel data.

2.1 Timing

A new package should be transmitted by the sender at a rate of 50Hz (every 20ms) which is the rate a normal Servo or ESC gets signals at. The minimum time between each package should be 10ms to allow every device to parse the data.

2.2 Mesh behavior

The sender of a message is the one deciding wheter a package should be mesh package or a normal package. A normal package gets tranmitted by the sender and every device in reach receives the message and can use the data.

In a mesh application the sender needs to set the mesh flag. Furthermore the sender needs to specify a routing length which sets the maximum amount of nodes the package passes by. Each recipient resends the message and decreases the routing length value by one until the routing length value reaches the value zero. To avoid that the same packages passes multiple nodes the same time every node needs to keep track of the packages it forwarded, this is done by having a sufficient sized ring buffer which saves the last UIDs of the packages forwarded, the size is determined by the amount of packages per time, in a normal application the size should be between 64 and 128.

In complex applications where two nodes can be connected via multiple paths it is not guaranteed for the communication to work, in general this should be avoided.

2.3 Package structure

Every package consists of four major parts:

1. Header
2. Configuration
3. Data
4. Footer

2.3.1 Header

The header consists of three bytes:

1. Start byte
2. Unique-ID
3. Transmitter-ID

Start byte

The start byte is one byte which is always $C9_{\mathrm{HEX}}$.

Unique-ID

The unique-ID is generated by the initial sender. It is used to identify a message, especially in a mesh-application.

Algorithm for generation of the UID The easiest way of generation is having a counter shared by all devices, every time a device receives or transmits a package the counter gets incremented. If the upper limit of a byte (255) is reached the counter overflows. This implementation guarantees maximum time between reoccurring UIDs.

Transmitter-ID

A persistent ID which should be unique to the transmitter. Optimally it should be easily configurable by the user, for example via a configuration file, a settings menu or DIP-Switches on the module.

2.3.2 Configuration

The configuration consists of at least one byte.

Value	Resolution (Steps)	Bit per Channel
000_2	32	5
001_2	64	6
010_2	128	7
011_2	256	8
100_2	512	9
101_2	1024	10
110_2	2048	11
111_2	4096	12

Figure 2.1: Possible resolution values

First configuration byte (general)

The first byte consists of the following bits:

- Bit 0-2: Resolution
- Bit 3-5: Channel count
- Bit 6: Error
- Bit 7: Following

Resolution

Three bytes containing the resolution of each channel transmitted, possible values are listened in table 2.1.

Channel count

Three bytes containing the amount of channels used, possible values are listed in table 2.2.

Value	Channels
000_2	1
001_2	2
010_2	4
011_2	8
100_2	16
101_2	32
110_2	64
111_2	256

Figure 2.2: Possible channel count values

Error

If a device registers an error (checksum, timeout during package transmission) this flag is set for the next 4 packages send by this device.

Following

This byte is a flag whether the following byte is still a configuration byte and not a data byte.

Second configuration byte (mesh)

The second byte consists of the following bits:

- Bit 0: Mesh-Message
- Bit 1-4: Routing Length
- Bit 5-6: unused
- Bit 7: Following

Mesh-Message

A flag whether the message is intended as a Mesh-Message, this flag defaults to false if the following flag in the first configuration byte is not set.

Routing Length

A number counting the amounts of nodes this message has been passed by. The initial sender sets the value and every node decrements the value. If the value reaches zero, the package doesn't get forwarded.

Following

This byte is a flag whether the following byte is still a configuration byte and not a data byte. At the moment this byte needs to be false.

2.3.3 Data

The size (in bytes) of the data section is determined by the resolution and the channel count:

$$\text{size} = \left\lceil \frac{\text{resolution} \cdot \text{Channel count}}{8} \right\rceil \tag{2.1}$$

The data is just queued together so the first r (with r being the channel resolution) bits are the first channel, bits r+1 to r*2 are the second until the last channel. If there are unused bits left at the end they are ignored.

2.3.4 Footer

The footer consists of two bytes Checksum End byte

Checksum

For the checksum think of the transmitted data as list starting with the unique-ID and ending with the last data byte. The checksum is once calculated by the sender and once by the recipient to verify the data is still correct.

Algorithm

Now take every but the last byte and do a XOR Operation with the next byte. Every operation returns a new byte, so your list which was initially n bytes long now is only $n-1$ bytes long. Repeat this Process until the list is only one byte long, this byte is your checksum.

End byte

The end byte is one byte which is always 93_{HEX}.