

Bluetooth Remote Control
Protocol Description
Version 0.2, 11/8/2007

1. Overview

The model car can be controlled via a bluetooth link supporting the Serial Port Profile (SPP). The car can be sent commands which will trigger some behavior (e. g. turn left, stop, ...) Additionally, the car will send data in the other direction (e. g. current speed, revolutions per minute) – see figure 1.

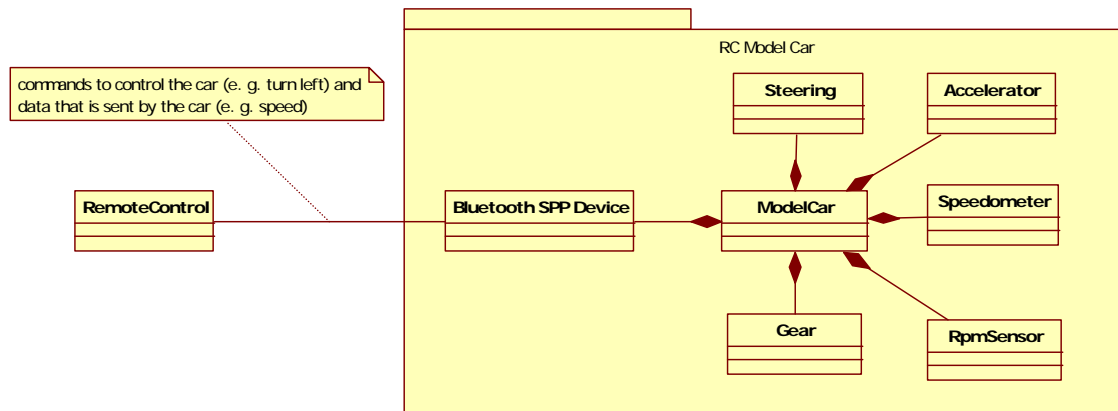


Figure 1: Domain Model

2. Frame Structure

This section describes the format of the messages that are sent to and from the car via the bluetooth SPP link. All messages are encoded binary, one byte consists of eight bits.

Each frame starts with a preamble (0xFF) which is used to identify the start of a frame, followed by a device identification byte. The device identification allows to address the device a message will be sent to or gives information about the origin of a message (right now, Steering, Accelerator, Speedometer, and rpmSensor are defined). Depending on the device, several messages can be sent which can be selected by using a message id (figure 2).

Preamble	DeviceID	MessageID	DataLength	Data	Checksum
----------	----------	-----------	------------	------	----------

Figure 2: Frame Structure

- Preamble: 1 Byte, always 0xFF, used to identify the start of a frame
- DeviceID: 1 Byte, identifies the device, see table 1 below.
- MessageID: 1 Byte, identifies the message type a device sends or receives, see table 2 below.
- DataLength: 1 Byte, indicates how many data bytes will follow (0 – 255)
- Data: 0 – 255 Bytes
- Checksum: 1 Byte, calculated by performing a modulo 256 sum over the entire packet (i. e. preamble, device id, message id, data length, and data)

2.1 Supported Devices

DeviceID	Description
0	RPMSensor
1	Gear
2	Speedometer
3	Steering
4	Accelerator
5	Horn

Table 1: Supported Devices

2.2 Supported Messages

DeviceID	MessageID	Description	DataLength	Data
0	0	Engine revolution per minutes, sent by the car whenever the values changes	2	16-bit unsigned int, little endian 0 – 65535 rpm
1	1	Set the gear servo	1	8-bit unsigned char,

				0, 255: servo end positions
2	0	current speed, sent by the car whenever the values changes	1	8-Bit unsigned char, 0 – 255 km/h
3	1	set direction	1	8-bit signed char 2's complement 0: straight ahead +1 .. + 127: left -1 .. - 128: right
4	1	set acceleration	1	8-Bit signed char 2's complement 0: car is stopped + 1 .. + 127: forward - 1 .. - 128: backward
5	0	turn on/off horn	1	0: horn off 1: horn on

Table 2: Supported Messages

3. Timing and Safety Considerations

To provide a safe state for the model car in case the wireless connection breaks, the car will stop automatically if it does not receive acceleration messages for approx. 500 ms: One has to make sure that acceleration messages are constantly sent to the car, otherwise the car will stop (see figure 3).

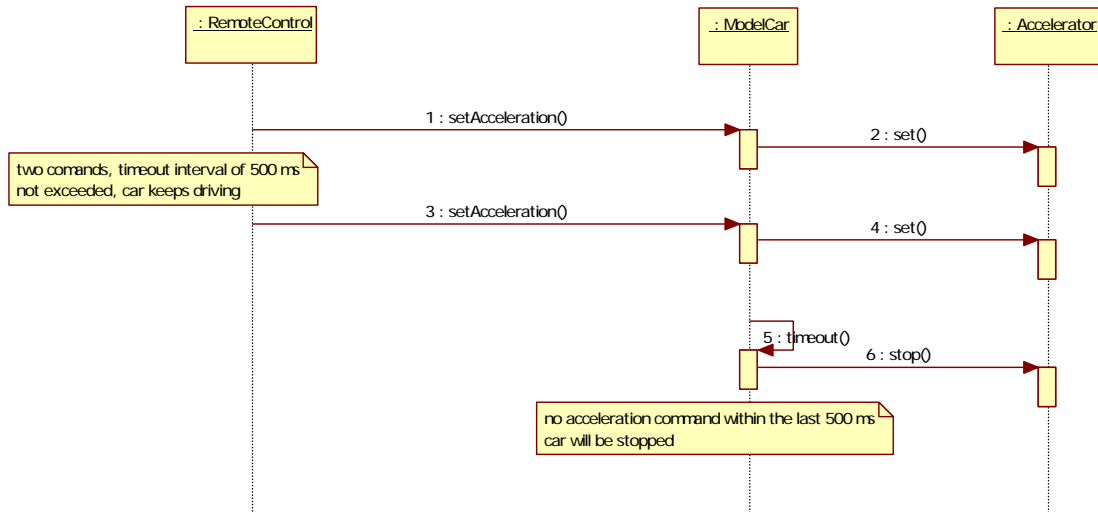


Figure 3: Sequence Diagram – car is stopped

Once a command is received again, the safe state will be exited and the car continues (figure 4).

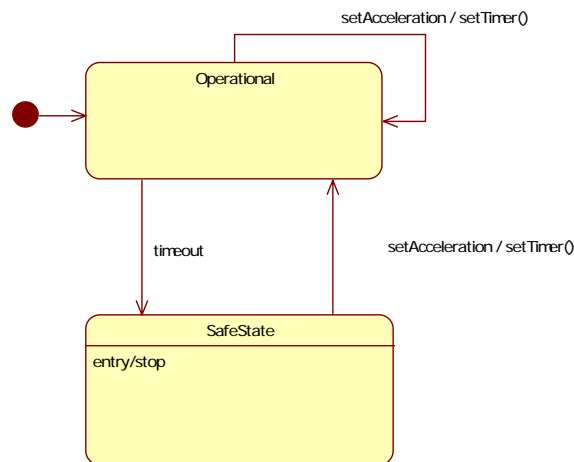


Figure 4: State Machine Diagram

4. Examples

4.1 Revolutions Per Minute

The remote control receives the following data from the model car:

0xFF 0x00 0x00 0x02 0x03 0x02 0x07

Preamble: 0xFF

DeviceID, MessageID: 0x00 0x00 ... revolutions per minute

DataLength: 0x02 ... 2 bytes data will follow

Data: 0x03 0x02 (little endian) => 0x0203 (big endian) ... 515 revolutions per minute

Checksum: 0x07: $0xFF + 0x00 + 0x00 + 0x02 + 0x03 + 0x03 = 0x0107 \Rightarrow 0x07$

The engine operates at 515 rpm.

4.2 Direction

The car receives the following data:

0xFF 0x03 0x01 0x01 0x00 0x05

Preamble: 0xFF

DeviceID, MessageID: 0x03 0x01 ... steering, set direction

DataLength: 0x01 ... 1 byte data will follow

Data: 0x00 ... keep straight ahead

Checksum: 0x05: $0xFF + 0x03 + 0x01 + 0x01 + 0x00 = 0x0105 \Rightarrow 0x05$

The car should drive straight ahead.