PiRail

A raspberry pi controlled model railway

Contents

[General Overview 2](#_Toc516844527)

[Requirements 2](#_Toc516844528)

[Websocket 2](#_Toc516844529)

[Trains 2](#_Toc516844530)

[Website 3](#_Toc516844531)

[Track 3](#_Toc516844532)

[Algorithm 5](#_Toc516844533)

[UDP Z21 5](#_Toc516844534)

[WebSocket Protocol 5](#_Toc516844535)

[Admin Controls 5](#_Toc516844536)

[Train (0x40 Flag) 7](#_Toc516844537)

[Track (0x20 Flag) 9](#_Toc516844538)

[General (0x10 Flag) 11](#_Toc516844539)

[Circuit of RailNet 12](#_Toc516844540)

[Protocol of RailNet 13](#_Toc516844541)

[General 13](#_Toc516844542)

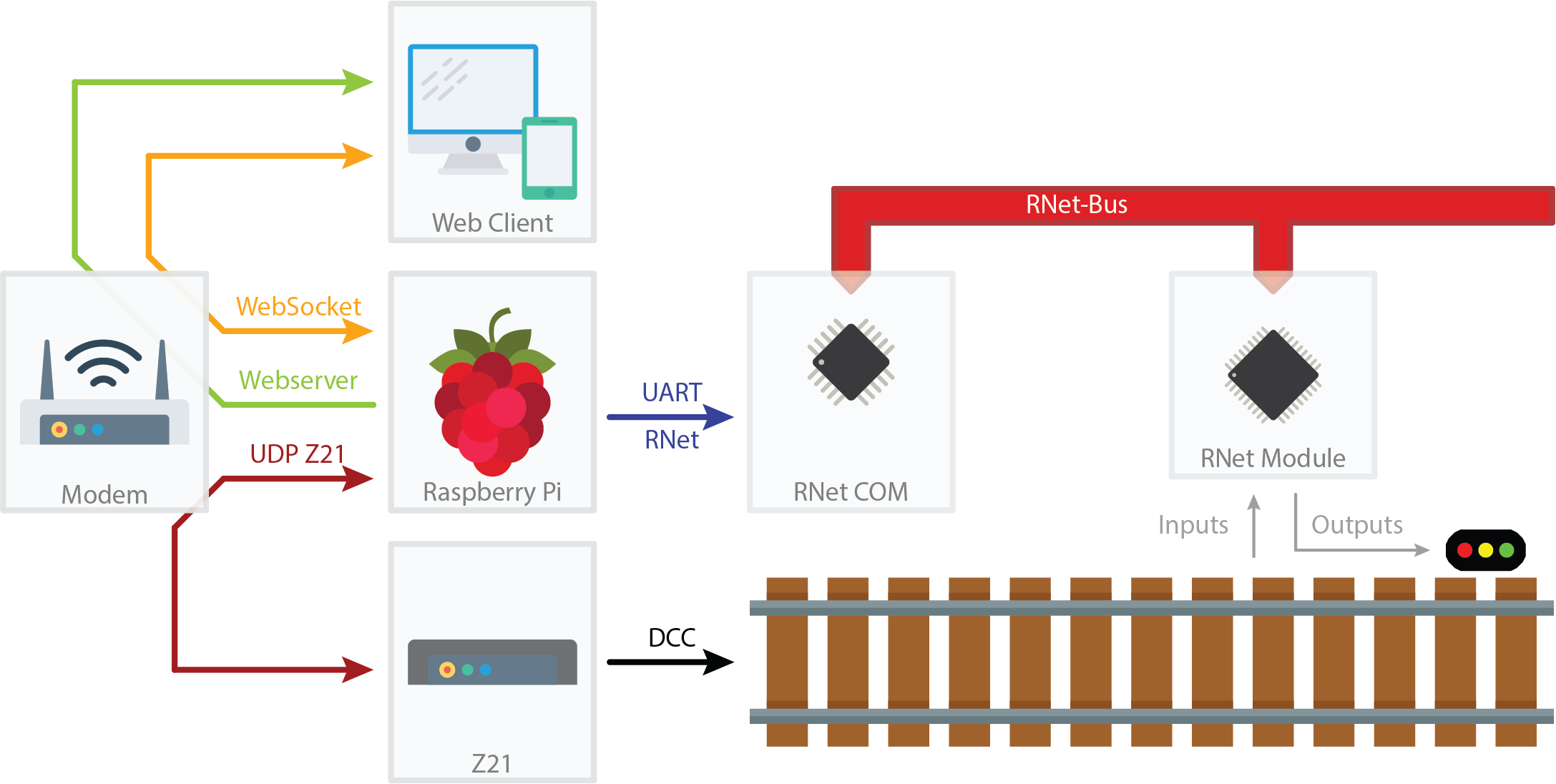
[Output 13](#_Toc516844543)

[Input 14](#_Toc516844544)

[Set Device Parameters 15](#_Toc516844545)

[EEPROM on Arduino 16](#_Toc516844546)

# General Overview



# Requirements

# RNet (v1)

Every device on the network can start transmitting. CSMA/CD (Carrier Sense Multiple Access / Collision Detection) as well as RS-485 are used to interconnect Modules.

A modules has a master node and some slave output and slave input nodes.

These nodes are interconnected using SPI shift register.

There is only two addresses list for the IO, one for input and one for output.

# RNet (v2)

Every device on the network can start transmitting. CSMA/CD (Carrier Sense Multiple Access / Collision Detection) as well as RS-485 are used to interconnect Modules.

A modules has one master node and some slave nodes.

These nodes are interconnected using i2c.

Each port of an node can be an output as well as an input. An output can be ON/BLINKING 1/BLINKING 2/OFF, Servo (0/1), PWM (0-255). An input can only be ON/OFF/PUSH.

Master node has address 1, slaves have counting numbers from 2.

A master node has a power supply for all local slave nodes.

There is only one addresses list for the IO.

## Websocket

The websocket shall work on port 9000, with the protocol defined below

## Trains

A train consists of multiple types of rolling stock, such as cars and engines. It has a duplicate list for all engines. It has a route, name, current block, current speed, max speed and timer.

An engine has a name, image path, icon path, dcc id, length, control type, direction, current\_speed\_step, max\_speed\_steps, stopped\_flags and max\_speed.  
A car has a name, image path, icon path, identifier number, length and max speed.

An engine and a car could have a function list, which has a type, momentary\_flag and state\_flag.

Each piece of rolling stock has a type.

A car can be used multiple times

An engine can only be used once, when in use: use\_flag is set.

Be able to add an engine, car or train.

Be able to add an engine, car or train from config files.

Be able to add route to train.

Be able to retrieve data from Z21 concerning several engines.

Be able to change speed, direction and functions of a train/engine.

Only a train can be on the rails, not an engine or car alone.

The engine, car configs are stored in non-volatile memory.

A train will be linked to a follow id, that is stored on the blocks.

A train will be linked to a dcc id, that is stored in memory.

## Website

## Track

### Unit

A unit is a group of blocks, switches, stations and signals with some input\_registers, output\_register, blink\_mask and an ID.

The input\_regs and output\_regs contain the number of how many 8-byte regs there are  
The InRegs, OutRegs and BlinkMask contain the GPIO values.

The unit configs consist of a contents config and a javascript json layout file. These configs are located in configs/units/<unit\_id>.<conf/json>

A Units has one or more connections to another unit. A connector can consist of one or more rails, those are only stored in the configs.

A Unit has a flag set when an output has changed.

A Unit has an input io list referring to the objects.

### Blocks

A block has several types: Main track, station track, shunting track, siding and special detection.

A block has a ioadr which corresponds with the gpio pin.

A block can be in several states: Blocked, Danger, Restricted, Caution, Proceed, Reserved, Reserved\_switch, Unknown.

A block has a speed limit, direction, reversed\_flag, oneway\_flag, blocked\_flag, changed\_flag, train\_follow\_id, length.

The direction of a block could be forward, reversed or switching. A switching block is used on a piece of rail between forward block and a reversed block.

The reversed flag is for polarity switching.  
A train is only allowed in the track direction if the oneway flag is set.  
The changed flag is used to which block has changed.

A block is linked to the next and previous block using pointers.  
A special detection block has no next and previous

A block has a pointer to a station if it is a station block.

A block can point to all (ms)switches in the same block.

The block loaded from configs have no rail\_link pointer yet, these are created with *Connect\_Segments();* . A next or previous link could be a unit connector, module is the connector number and id is the rail number of the connector.

### Switches

A switch has one “Approach” on one side and a “Diverging” and “Straight” on the other side. The state corosponds which path is selected: 0 is Approach<>Straight, 1 is Approach<>Diverging.

A switch has a detection block. Could be ether the same as the approach connection or a special block for the switch or switch group.

A switch has a default state. It will return on reset to this state. The default state will be 0 on init.

A switch could be coupled to other switches, using the linked\_switch

A switch could have a preference list for directing different kinds of trains in separate

A switch has a hold flag.

A switch has a list of output pins and for each state a list of enabled outputs.

A switch could have feedback Inputs.

### MSSwitch

A multistate\_switch (MSswitch) has a defined length states and thus a defined number of rail in- and outputs.

A MSswitch has a detection block. Could be ether the same as the approach connection or a special block for the switch or switch group.

MSswitch side A is on the M side, side B is on the m side. So with a rail\_link with type M should check side B.

A MSswitch has a default state. It will return on reset to this state. The default state will be 0 on init.

A MSswitch could be coupled to other switches, using the linked\_switch

A MSswitch could have a preference list for directing different kinds of trains in separate ways.

A MSswitch has a hold flag.

A MSswitch has a list of output pins and for each state a list of enabled outputs.

A MSswitch could have feedback IO.

### Station

A Station consist of one or more blocks, a name, a type and switches.

Station type: person, cargo or yard.

Switches should be added to a station if a switch is between the begin and end point.

### Signals

A signal has a module number, id and io number. It has a rail\_state state,

A signals can have 8 output pins maximum.

The output signal are defined for all states.

The output flash signals are defined for all states.

## Algorithm

# UDP Z21

The Z21 uses its own predefined protocol. It can be studied from the “Z21\_LAN\_Protokoll V1.05.pdf” file, it is only published in German.

# WebSocket Protocol

*WebSocket is a computer communications protocol, providing full-duplex communication channels over a single TCP connection. The WebSocket protocol was standardized by the IETF as RFC 6455 in 2011, and the WebSocket API in Web IDL is being standardized by the W3C.*

Al binary representation have the MSB at the left side, otherwise it is explicitly stated.

Each Websocket message is send as a binary packet.

You can register to certain topics as a Websocket client using the protocol properties. 25510 (1111 11112) registers you to all the topics, e.g. “Sec-WebSocket-Protocol: 255”

The flags are setup as following:

ABCD EFGH

A – C = Unused  
D = Admin subscription  
E = Messages  
F = Switches  
G = Track  
H = Trains

The messages are divided into 4 different categories. The system checks for the first set bit. The packet belong to a category if the first bit equals the flag.

## Admin Controls

All messages that has a format of 1xxx xxxx (0x80 flag) is reconised as an admin control packet

### Clear track

*Message from client to server and from the server to every client*Delete all the track from current layout, or notify all members that the track has been cleared.

|  |
| --- |
| 0x80 |

### Reload track

Rescan the track for modules. They still need to be ‘joined’ together.

0x81

### Track scan progress

Message from server to client

|  |  |  |  |
| --- | --- | --- | --- |
| 0x82 | Uint8 A | Uint8 B | Module ID |

A Number of connected points  
 B Total number of connection points  
Message from client to server

|  |  |  |  |
| --- | --- | --- | --- |
| 0x82 | Func | Data Length | Data |

Func

0x01 Stop connecting (no data length and data byte(s))  
0x02 Reload previous setup

### Track Layout Update

*Only from server to client*

*Send a partial layout / one module.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0x83 | AAAA AAAA | B1 | B2 | (B3) |

AAAA AAAA = Module number

B1 (Bx) = Connected modules to A, x = anchor number, all anchor should be send except non-existing

Track info

*Message from client to server*Request data from the track

|  |
| --- |
| 0x83 |

*Message from server to client*

Get the info of the track: voltage, current, uptime, downtime. NOT IMPLEMENTED

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0x83 | Int16 A | Uint16 B | Uint16 C | Uint16 D |

A Track Voltage (mV) 16 bit Integer  
 B Track Current (mA) 16 bit Unsinged integer  
 C Uptime (sec) 16 bit Unsigned  
 D Downtime (sec) 16 bit Unsigned

### Reset Switches

Reset all switches to default

0x84

### Force Switch

Force a switch to a state independent of reserved state.

0x85

### Toggle (Light) Output

Toggle output of modules on or off.

0x86

### All trains back to depot

Send all trains back to depot, only if depot space is available.

0x87

### Emergency Stop, Admin authority

Set emergency stop. Can only be release with admin authority (the send code).

0x90 Admin Code 2 bytes (0-65534)

### Emergency Release, Admin authority

Release Emergency stop.

0x91 Admin Code 2 bytes (0-65534)

### Disable Admin authority for this connection

*From client to server*Logout from the admin services. The server will return with 0x15 (Broadcast change) message to confirm the request.

0xFE

### Enable Admin authority for this connection

*From client to server*Login as admin to the services. The server will return with 0x15 (Broadcast change) message, confirming or denying the request.

0xFF Password hash length Password hash

## Train (0x40 Flag)

### Link train

*Message from client to server*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0x41 | AAAA AAAA | BBBB BBBB | CxxD DDDD | DDDD DDDD |

AAAA AAAA = Follow ID  
 BBBB BBBB = Real ID  
 Cxxx xxxx = Type (0 = Train, 1 = Engine)  
 xxxD DDDD DDDD DDDD = Message ID

### Speed control

*Data from client to server*

|  |  |  |
| --- | --- | --- |
| 0x42 | AAAA AAAA | BCCC CCCC |

AAAA AAAA = Follow ID  
 Bxxx xxxx = Direction, set = forward, unset = reverse  
 xCCC CCCC = Speed

### Function control

*Data from client to server*

0x43

### Operation change

*Data from client to server*

0x44

### Train data from Z21

*Data from server to client*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0x45 | AAAA AAAA | DB0 | … | DBn |

AAAA AAAA = Train ID

DB0 – DBn = Z21 Databits *LAN\_X\_LOCO\_INFO*

### Set Route

Both ways

|  |  |  |  |
| --- | --- | --- | --- |
| 0x46 | AAAA AAAA | BBBB BBBB | CCCC CCCC |

AAAA AAAA = Train ID

BBBB BBBB = Destination Module

CCCC CCCC = Destination Station/Platform number

### Add engine to library

Client to server

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x50 | Uint16 A | Uint16 B | Uint16 C | D | E | F | H | strings |

Uint16 A = DCC ID (little endian)

Uint16 B = Max speed

Uint16 C = Length (mm)

D = Type

E = Name length

F = Image path

H = Icon path

Server to client

|  |  |  |
| --- | --- | --- |
| 0x50 | Uint16 A | B |

Uint16 A = DCC ID (little endian)

B = Return value (-1 = DCC address already in use, 0 = Failed, 1 = Success)

### Engines Library

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x51 | Uint16 A | Uint16 B | Uint16 C | D | E | F | H | strings |

Uint16 A = DCC ID (little endian)

Uint16 B = Max speed

Uint16 C = Length (mm)

D = Type

E = Name length

F = Image path

H = Icon path

Repeated for each engine

### Add car to library

Client to server

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x52 | Uint32 A | Uint16 B | Uint16 C | D | E | F | H | strings |

Uint16 A = Part Nr (little endian)

Uint16 B = Max speed

Uint16 C = Length (mm)

D = Type

E = Name length

F = Image path

H = Icon path

### Cars Library

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x53 | Uint16 A | Uint16 B | Uint16 C | D | E | F | H | strings |

Uint16 A = Part Nr (little endian)

Uint16 B = Max speed

Uint16 C = Length (mm)

D = Type

E = Name length

F = Image path

H = Icon path

Repeated for each car

### Add train to library

0x54

Add option to store for next uses

### Train Library

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x55 | Uint16 A | Uint16 B | xxxx DCCC | E | F | name | Links | |
| Type | ID |

Uint16 A = Max speed

Uint16 B = Length (mm)

xCCC = Type

Dxxx = On track

E = Name length

F = Nr of engines/cars – and thus links

x don’t care

Repeated for each train

## Track (0x20 Flag)

### Set switch

*from client to server  
Set a (MS)Switch to a specific state. 0 = Straight, 1 = Diverging.*

|  |  |  |  |
| --- | --- | --- | --- |
| 0x20 | AAAA AAAA | BCCC CCCC | DDDD DDDD |

AAAA AAAA = Module number of switch  
 Bxxx xxxx = Set if MSSwitch  
 xCCC CCCC = (MS)Switch ID  
 DDDD DDDD = New state

### Set Multiple switches

*from client to server  
Set multiple (MS)Switches to a specific state. 0 = Straight, 1 = Diverging.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0x21 | AAAA AAAA | BBBB BBBB | CDDD DDDD | EEEE EEEE | BBBB BBBB |

AAAA AAAA = Number of switches  
 BBBB BBBB = Module number of switch  
 Cxxx xxxx = Set if MSSwitch  
 xDDD DDDD = (MS)Switch ID  
 EEEE EEEE = New state

### Set switch reserved

0x22

### Change reserved switch

0x23

### Set switches for route

*Mostly from client to server  
Server Calculates a route from point A to point B and set the switches accordingly and reserves them.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0x25 | AAAA AAAA | BBBB BBBB | CCCC CCCC | DDDD DDDD |

AAAA AAAA = Module number of point A  
 BBBB BBBB = Block of point A  
 CCCC CCCC = Module number of point B  
 DDDD DDDD = Block of point B

### Broadcast track occupation

*Only from server to client*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0x26 | AAAA AAAA | BBBB BBBB | CxxD EEEE | FFFF FFFF | AAAA AAAA… |

AAAA AAAA = Module number of block  
 BBBB BBBB = Block ID  
 Cxxx xxxx = Direction of block, set when block is reversed  
 xxxD xxxx = Set when block is occupied  
 xxxx EEEE = State of block: 0000 free, 0001 amber, 0010 red, 0011 unknown, 0100 ghost, 0101 blue  
 FFFF FFFF = ID of train, 0 if not occupied  
A – F is repeated for all blocks

### Broadcast states of switches

*Only from server to client*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0x27 | AAAA AAAA | BCCC CCCC | DDDD DDDD | EEEE EEEE | AAAA AAAA… |

AAAA AAAA = Module number of switch  
 Bxxx xxxx = Set if it is a Multi-state switch  
 xCCC CCCC = (MS)Switch ID  
 DDDD DDDD = New State  
 EEEE EEEE = Number of states, only for a MSSwitch, skip for a normal switch  
A – E is repeated for all (ms)switches

### Track Layout Setup

*Only from server to client*

*Send the total layout to the client*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0x30 | AAAA AAAA | B1 | B2 | (B3) | A… |

AAAA AAAA = Module number

B1 (Bx) = Connected modules to A, x = anchor number, all anchor should be send except non-existing

A – Bx is repeated for all modules

### Station Library

0x31

## General (0x10 Flag)

### Emergency stop

*Can be send or received*Server sends this message when the stop button has been pressed. Or client sends this when the button is pressed, server will confirm with same opcode.

|  |
| --- |
| 0x10 |

### Short Circuit stop

*Always from server to client  
Server sends this message at a short circuit detection, and stopping all traffic.*

|  |
| --- |
| 0x11 |

### Emergency release

*Can be send or received  
Server sends this message when the stop button has been released. Or client sends this when the button is released, server will confirm with same opcode.*

|  |
| --- |
| 0x12 |

### New Message

*Always from server to client  
Server sends a message to all clients to inform them of something.*

|  |  |  |  |
| --- | --- | --- | --- |
| 0x13 | AAAB BBBB | BBBB BBBB | Data |

AAAx xxxx = Message type  
 xxxB BBBB BBBB BBBB = Message ID

Data is depended on the type:   
*A new train has been put on the tracks (type 0)*

|  |  |  |  |
| --- | --- | --- | --- |
| Header | CCCC CCCC | DDDD DDDD | EEEE EEEE |

CCCC CCCC = and has this follow ID  
 DDDD DDDD = Train found on this module  
 EEEE EEEE = on this block   
*A train has split in a yard tracks (type 1)*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Header | CCCC CCCC | DDDD DDDD | EEEE EEEE | FFFF FFFF | GGGG GGGG |

CCCC CCCC = follow ID of train  
 DDDD DDDD = Part A on this module  
 EEEE EEEE = on this block  
 FFFF FFFF = Part B on this module  
 GGGG GGGG = on this block   
*A train has split on the main tracks (type 2)*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Header | CCCC CCCC | DDDD DDDD | EEEE EEEE | FFFF FFFF | GGGG GGGG |

CCCC CCCC = follow ID of train  
 DDDD DDDD = Part A on this module  
 EEEE EEEE = on this block  
 FFFF FFFF = Part B on this module  
 GGGG GGGG = on this block

Message update

0x14 Message ID uint16\_t NewMessage data

### Message Clear

0x15 Message ID uint16\_t

### Change Broadcast flags

*From client to server*Client request a change in the flags. Server will send current status if ‘New flags’ is set to zero. This doesn’t change the actual broadcast flags

|  |  |
| --- | --- |
| 0x16 | New Flags |

*From server to client*Server confirms/updates the change in flags settings.

|  |  |
| --- | --- |
| 0x16 | Flags |

### Server state

From server to client only

|  |  |
| --- | --- |
| 0x17 | Flags |

0x8000 STATE\_Z21\_FLAG

0x4000 STATE\_WebSocket\_FLAG

0x2000 STATE\_COM\_FLAG

0x1000 STATE\_Client\_Accept0x0200 STATE\_TRACK\_DIGITAL

0x0100 STATE\_RUN0x0008 STATE\_Modules\_Coupled

0x0004 STATE\_Modules\_Loaded

0x0001 STATE\_Trains\_Loaded

### Canvas Data

From server to client only

|  |  |  |
| --- | --- | --- |
| 0x1F | ModuleID | Data |

# Circuit of RailNet

There are 4 RNet devices in the network: COM interface, RailNet controller, RailNet output module, RailNet input module.

# Protocol of RailNet

A message must contain an opcode and a checksum. Some packet do not have a predefined length, then it is mandatory to send a length byte after the opcode.

## General

### Report ID

*From device to master*Each device sends this message at startup, so that the controller/master know which devices are in the network.

|  |  |  |
| --- | --- | --- |
| 0x00 | DevID | Checksum |

### Set Emergency STOP

*From device to maste*

A device that has an emergency button connect can send this message to call for an emergency stop on the layout.

|  |  |
| --- | --- |
| 0x01 | Checksum |

### Release Emergency STOP

*From device to master*A device that has an emergency button connect can send this message to release an emergency stop on the layout.

|  |  |
| --- | --- |
| 0x02 | Checksum |

### Set Power ON

*From device to master*A device that has a layout power switch connected can send this message to enable the track power.

|  |  |
| --- | --- |
| 0x03 | Checksum |

### Set Power OFF

*From device to master*A device that has a layout power switch connected can send this message to disable the track power.

|  |  |
| --- | --- |
| 0x04 | Checksum |

### Reset All Devices

From master to devices  
Resets the device to the initialization state, and resends the DeviceID packet.

|  |  |
| --- | --- |
| 0x05 | Checksum |

### Set Acknowledge

*From device to master*The device sends this message to acknowledge the changes in the device parameters.

|  |  |  |
| --- | --- | --- |
| 0x7F | DevID | Checksum |

## Output

### Toggle Single Address

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0x10 | DevID | Output L | Output H | Checksum |

Output Low: ID & 0x7F  
Output High: (ID >> 7) & 0x7F

Toggles the output of the specified ID.

### Pulse Single Address

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0x11 | DevID | Output L | Output H | Checksum |

Output Low: ID & 0x7F  
Output High: (ID >> 7) & 0x7F

Send a pulse on the output of the specified ID.

### Toggle Blink Single Address

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0x12 | DevID | Output L | Output H | Checksum |

Output Low: ID & 0x7F  
Output High: (ID >> 7) & 0x7F

Toggles the output of the specified ID.

### Toggle Multiple Addresses

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0x13 | Length | DevID | Output1 L | Output1 H | … | Checksum |

Output Low: ID & 0x7F  
Output High: (ID >> 7) & 0x7F

Toggles multiple addresses

### Set All output Addresses

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0x14 | Length | DevID | Output1 | Output2 | … | Checksum |

Output1: Output & 0x7F  
Output2: (Output >> 7) & 0x7F  
Output3: (Output >> 14) & 0x7F

Set all output to the specified bytes. Output 1 contains the ID of 0-6, output 2 contains 7-13, and so on. NOT IMPLEMENTED

### Set Blink mask of all addresses

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0x15 | Length | DevID | Output1 | Output2 | … | Checksum |

Output1: Output & 0x7F  
Output2: (Output >> 7) & 0x7F  
Output3: (Output >> 14) & 0x7F

Set the blink mask to the specified bytes. Output 1 contains the ID of 0-6, output 2 contains 7-13, and so on. NOT IMPLEMENTED

### Post all Output

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0x16 | Length | DevID | Output1 | Output2 | … | Checksum |

### Post Blink Mask

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0x17 | Length | DevID | Output1 | Output2 | … | Checksum |

### Request Read all Output states and Blink Mask

|  |  |  |
| --- | --- | --- |
| 0x47 | DevID | Checksum |

***Response:*** Post all Output (0x07)

## Input

### Post Single Input Address

|  |  |  |  |
| --- | --- | --- | --- |
| 0x06 | DevID | Output | Checksum |

DevID: ID & 0x7F

Output Low: ID & 0x7F  
Output High: (ID >> 7) & 0x7F

### Post Multiple Addresses

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0x07 | Length | DevID | Output1 | Output2 | … | Checksum |

DevID: ID & 0x7F

Output Low: ID & 0x7F  
Output High: (ID >> 7) & 0x7F

### Post All input Addresses

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0x08 | Length | DevID | OutReg1 | OutReg2 | … | Checksum |

DevID: ID

### Request Read all Inputs

|  |  |  |
| --- | --- | --- |
| 0x4C | DevID | Checksum |

DevID: ID & 0x7F

***Response:*** Post all Input (0x07)

## Set Device Parameters

***The device acknowledges all the changes to its parameters with the acknowledge packet***

### Change Device ID

|  |  |  |  |
| --- | --- | --- | --- |
| 0x50 | Old ID | New ID | Checksum |

### Set number of input modules and output modules

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0x51 | Dev ID | Input Modules | Output Modules | Checksum |

### Set blink Interval

|  |  |  |  |
| --- | --- | --- | --- |
| 0x52 | Dev ID | Interval scalar | Checksum |

Blink interval (ms) =

### Set Pulse length

|  |  |  |  |
| --- | --- | --- | --- |
| 0x53 | Dev ID | Pulse scalar | Checksum |

Pulse duration (ms) =

### Set Check input Interval

|  |  |  |  |
| --- | --- | --- | --- |
| 0x54 | Dev ID | Interval scalar | Checksum |

Check interval (ms) =

### Post All EEPROM variables

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0x55 | Length | Dev ID | Byte 0 | Byte 1 | … | Checksum |

### Request All EEPROM Values

|  |  |  |
| --- | --- | --- |
| 0x59 | Dev ID | Checksum |

## EEPROM on ATMega328P

|  |  |  |
| --- | --- | --- |
|  | 0x00 | 0x10 |
| 0x0 |  |  |
| 0x1 | Device ID |  |
| 0x2 | Input Devices |  |
| 0x3 | Output Devices |  |
| 0x4 | Blink scalar |  |
| 0x5 | Pulse scalar |  |
| 0x6 | Check scalar |  |
| 0x7 |  |  |
| 0x8 |  |  |
| 0x9 |  |  |
| 0xA |  |  |
| 0xB |  |  |
| 0xC |  |  |
| 0xD |  |  |
| 0xE |  |  |
| 0xF |  |  |