PiRail

A raspberry pi controlled model railway

Contents

[1 General Overview 2](#_Toc529996895)

[2 Requirements 3](#_Toc529996896)

[2.1 RNet (v1) 3](#_Toc529996897)

[2.2 RNet (v2) 4](#_Toc529996898)

[2.3 Websocket 4](#_Toc529996899)

[2.4 Rolling Stock 5](#_Toc529996900)

[2.5 Website 7](#_Toc529996901)

[2.6 Track 7](#_Toc529996902)

[2.7 Algorithm 10](#_Toc529996903)

[3 UDP Z21 11](#_Toc529996904)

[4 WebSocket Protocol 11](#_Toc529996905)

[4.1 System Controls 12](#_Toc529996906)

[4.2 Train (0x40 Flag) 16](#_Toc529996907)

[4.3 Track (0x20 Flag) 20](#_Toc529996908)

[4.4 Client / General (0x10 Flag) 22](#_Toc529996909)

[5 Circuit of RailNet 23](#_Toc529996910)

[6 Protocol of RailNet 24](#_Toc529996911)

[6.1 General 24](#_Toc529996912)

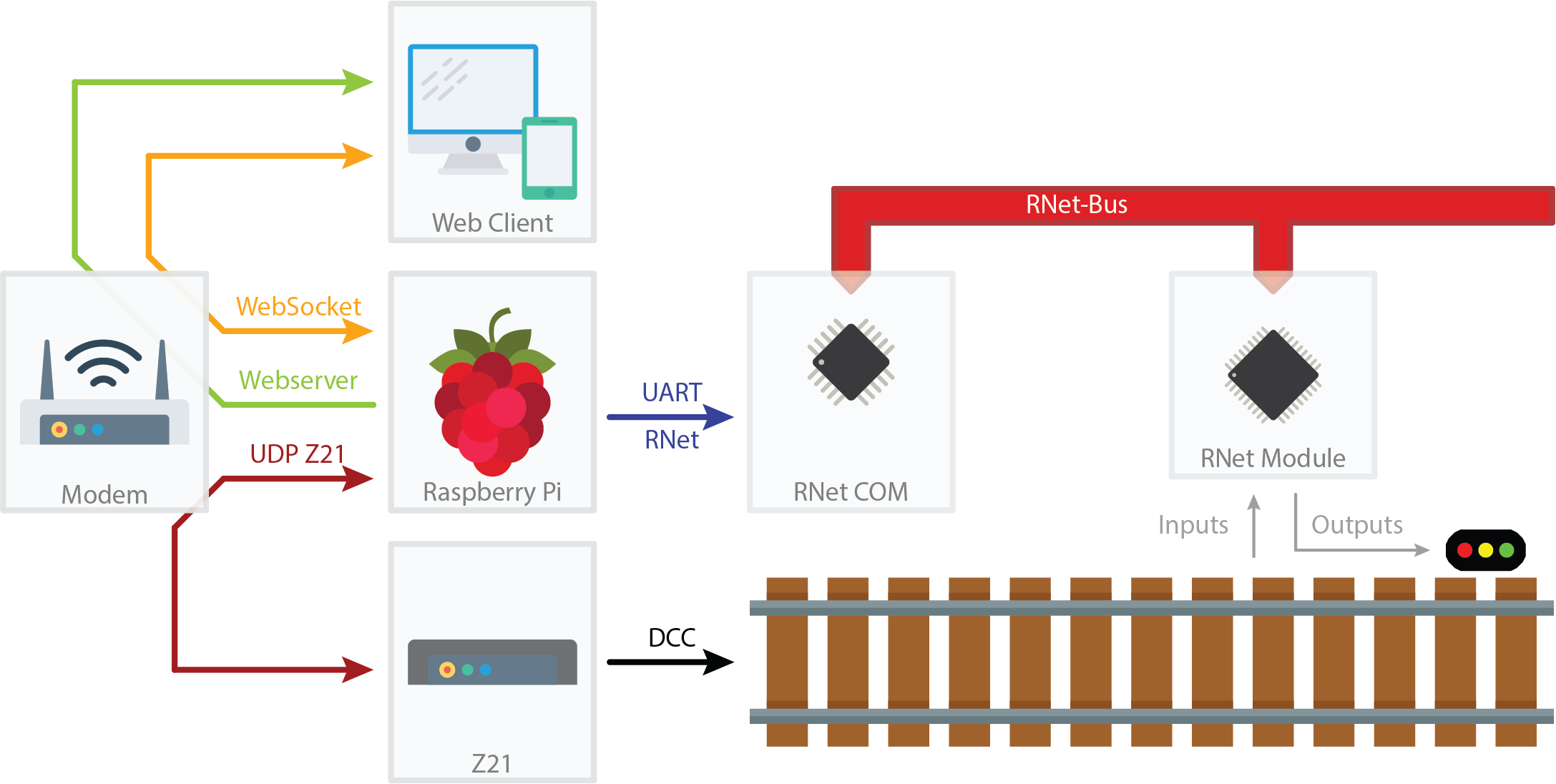
[6.2 IO 25](#_Toc529996913)

[6.3 Set Device Parameters 26](#_Toc529996914)

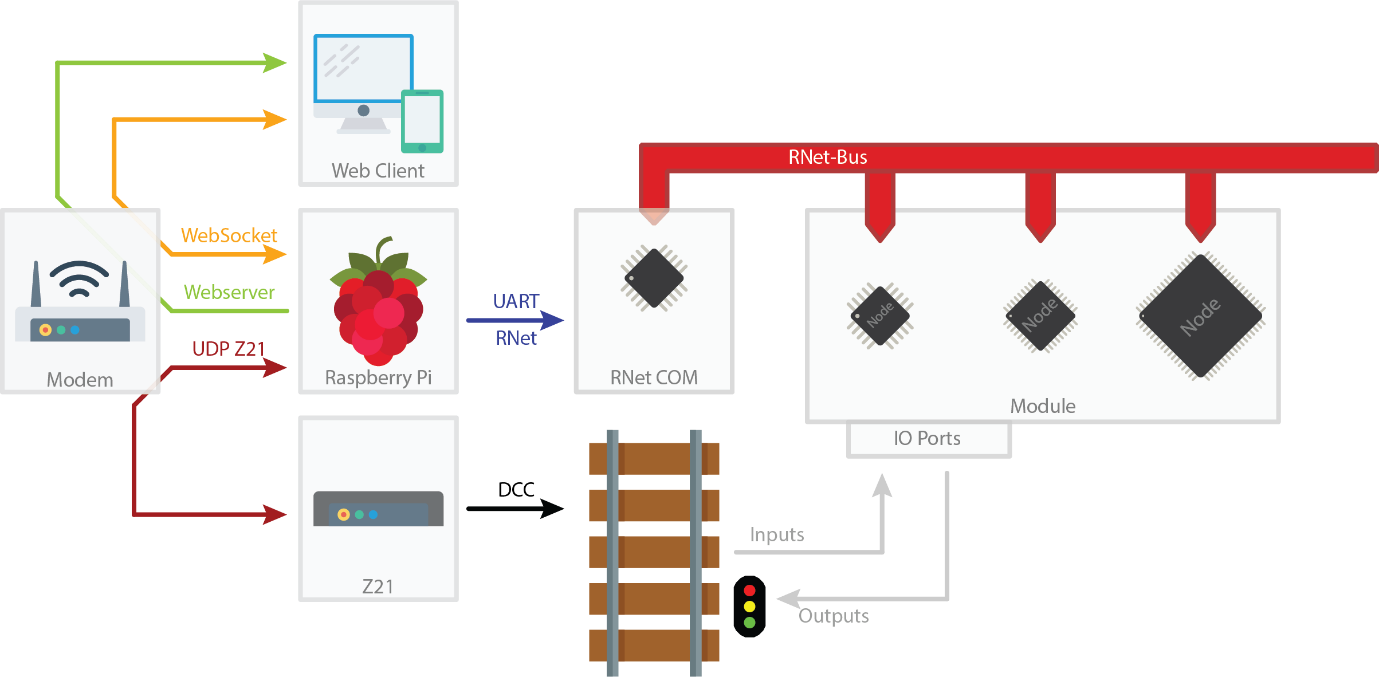
[6.4 EEPROM on ATMega328P 27](#_Toc529996915)

# General Overview

RNet(v1)



RNet(v2)



# 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.

To be compatible with v2/v2.1:

Inputs are first and Output will be offset => 16 inputs, 8 output => IO 16 is first output, 23 the last

Addressing scheme will be like Module:1:Port (Only one node)

## 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.

There are several different sizes of the boards. Ranging from 10 IO until 60 IO.

Each board has an interface to the RNet bus.

Each port of an node can be an output as well as an input. An output can be a regular output, Servo, PWM. An input can only be TOGGLE/PUSH.

Addressing scheme will be like Module:Node:Port. There can be 254 Modules max, (0 = Master, 255 = Broadcast). There can only be 16 Nodes max.

An servo could have four position (four states) and speed.  
The servo mode will change from one position to the other position with the set speed

An output could be High, Blinking pattern 1, Blinking pattern 2, pulse and off

A PWM output could have four values (four states) and speed.  
The PWM mode will change from one value to the other position with the set speed

A toggle input responds on both a rising edge as a falling edge.  
A push input responds only on a rising edge.

Blinking pattern 1, 2 and the pulse are programmable.  
PWM and Servo states are programmable.

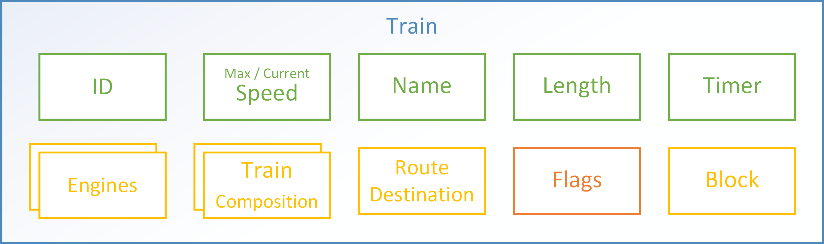
An IO port that is in PWM or Servo mode will notify when it has reached the desired state value.

## Websocket

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

## Rolling Stock

### Trains



A train consists of multiple types of rolling stock, such as cars and engines.

Each train has a route, name, current block, current speed step, max speed step and a timer.

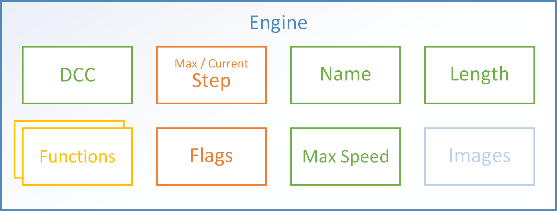
A Train could have two categories Person or Cargo:

The user can create his/her own collection of types in those categories, for example:

* Person
  + High-speed
  + Intercity
  + Regional
  + History
* Cargo
  + Coal
  + Tankers
  + Boxes
  + Containers

A train has a route to a destination.

### Engine



An engine has a name, 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.

The flags consist out of:  
Stopped 0 = stopped, 1 = waiting at station, 2 = waiting for signal  
Type Personal or cargo catagory.  
Dir 0 = forward, 1 = reverse  
Control 0 = manual, 1 = semi-automatic, 2 = full AI

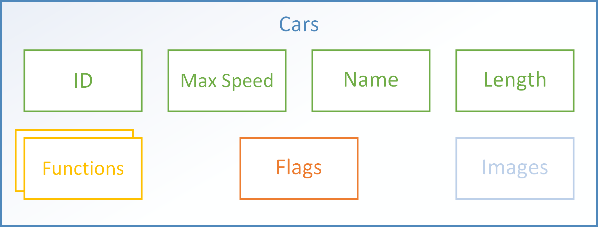
An engine and a car could have a function list, which has a type, momentary\_flag and state\_flag.

An engine can only be used once.

The image and icon of an engine will be stored like: {dcc}\_{name}.{filetype}, 1821\_traxx.png

An engine has a speed step to real speed curve.

### Cars



A car can be used multiple times

A car has a type: high-speed, personal, cargo or no-type.

The image and icon of a car will be stored like: {name}\_{nr}.{filetype}, ns\_icr\_15915.png

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.

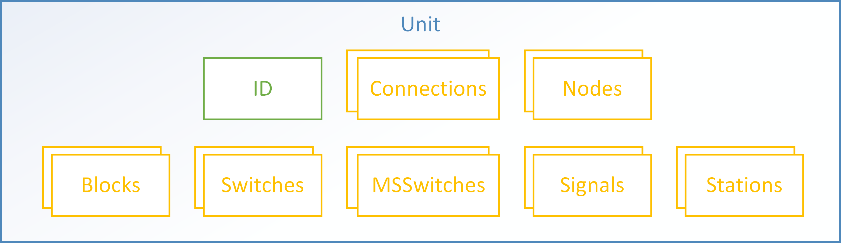
A train will be linked to a follow id, that is stored on the blocks.

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

## Website

## Track

### Unit

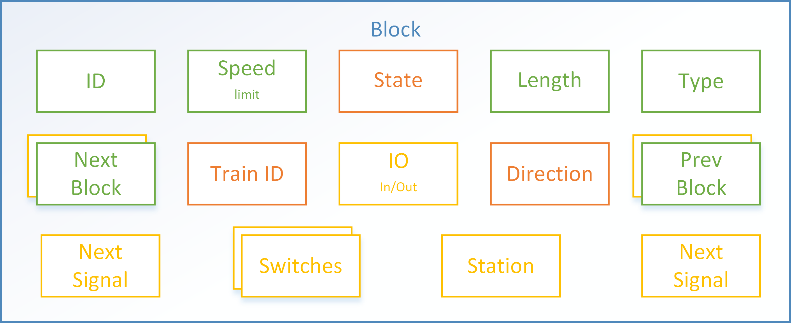


A unit is a group of blocks, (MS)switches, stations, signals, connections, IO\_nodes and an ID.

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

### Blocks



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

A block has can have two IO addresses assigned. One input for the detection and one ouput for direction dependend polarity switching.

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 direction can be reversed by changing the reverse\_flag. 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 show block has changed since previous itterations.

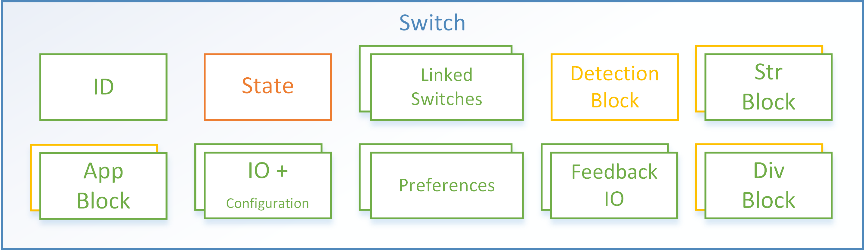
A block is linked to the next and previous block using pointers.  
A special detection block has no next and previous

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

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

The default state (on reset) will be 0.

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

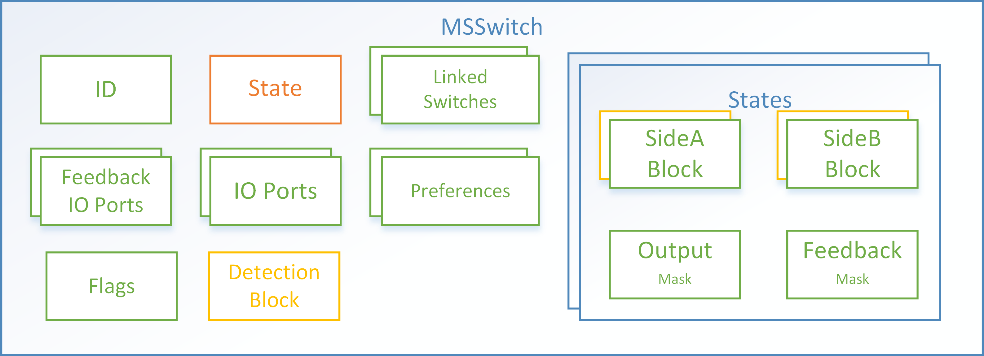
A switch could have a preference list for directing different kinds of trains in separate

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

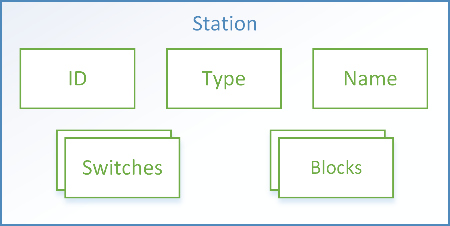
A MSswitch could have a preference list for directing different kinds of trains in separate ways.

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.

Allowed types

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

The algorithm shall select neighboring blocks for over 3 times the minimum algor\_block\_length. It is divided into 3 groups: PPP PP P CB N NN NNN.

Blocks are only check if they changed from FREE to BLOCKED, from BLOCKED to FREE or are forced to be checked.

### States

The block shall be set to DANGER if the next block is BLOCKED and current block is not BLOCKED.  
If current block is blocked and previous is not Blocked, previous blocks shall be set to DANGER  
The Previous block shall be set to CAUTION if the current block is clear and the next block is BLOCKED and the previous block is not blocked.  
The 2nd Previous block shall be set to PROCEED if the current block is clear and the next block is BLOCKED and the 2nd previous block is not blocked. Further back should be clear.

Add stating for switches, stations, yards …

### Signals

For each selected block:  
Change the state of the signal if the Next state is not equal to the NextSignals state.  
Change the state of the signal if the Prev state is not equal to the PrevSignals state.

If no next block is available, set nextSignal to DANGER

# 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.

• Client to server

• Server to client

• Un-implemented

## System Controls

All messages that has a format of 1xxx xxxx (0x80 flag) is recognized as an system control packet. Opcodes with the flag x1xx xxxx (0x40 flag) is an admin 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 Configuration Setup • •

*Only from server to client*

*Send the total layout to the client*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0x84 | 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

### Z21 Track info •• •

Data from the Z21 concerning track voltage, current, temperature and control states. It is broadcasted every 15 seconds.

*Message from client to server*

|  |
| --- |
| 0x86 |

*Message from server to client*

|  |  |
| --- | --- |
| 0x86 | Dataset |

|  |  |  |  |
| --- | --- | --- | --- |
| Offset | Type | Name | Unit |
| 0 | Int16 | MainCurrent | mA |
| 2 | Int16 | MainFilterdCurrent | mA |
| 4 | Int16 | ProgCurrent | mA |
| 6 | Uint16 | MainVoltage | mV |
| 8 | Uint16 | SupplyVoltage | mV |
| 10 | Uint16 | Temperature | °C |
| 12 | 16-bitmask | CentralState |  |

16-Bitmask:

csEmergencyStop 0x0001 // Emergency stop enabled

csTrackVoltageOff 0x0002 // Track Power turned off

csShortCircuit 0x0004 // Short Circuit

csProgrammingModeActive 0x0020 // Active Programming

cseHighTemperature 0x0100 // Temperature to high

csePowerLost 0x0200 // Supplyvoltage to low

cseShortCircuitExternal 0x0400 // On external Booster-Output

cseShortCircuitInternal 0x0800 // On Main or Program track

### Z21 Settings •• •

*Message from client to server*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0x87 | A | B | C | D |

IP-address: A.B.C.D

*Message from server to client*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0x87 | A | B | C | D | F | G |

IP-address: A.B.C.D  
Firmware-version: F major, G minor

### Reset Switches • •

Reset all switches to default

0x8C

### All trains back to depot •• •

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

0x8F

### Enable sub-module •

Enable a sub system (Z21, UART, Train Control, Layout Control, Websocket clients)

|  |  |
| --- | --- |
| 0x90 | Component |

Components flags see **4.1.11 - Disable sub-module •**, Reply’s with **4.1.12 - Sub-module state •**

### Disable sub-module •

Disable a sub system (Z21, UART, Train Control, Layout Control, Websocket clients)

|  |  |
| --- | --- |
| 0x91 | Component |

Components: Websocket 0x80  
 Z21 0x40  
 UART 0x20  
 LC 0x10  
 TC 0x08  
 0x07 Reserved

Reply’s with **4.1.12 - Sub-module state •**

Some components require other components:  
TC requires: LC, Z21  
LC requires: UART

### Sub-module state •

|  |  |  |
| --- | --- | --- |
| 0x92 | AABB CCDD | EEEx xxxx |

AA = Websocket, BB = Z21, CC = UART, DD = TC, EEE = LC  
States: 0b00 = Off 0b100 = Finding Modules  
 0b01 = Initializing 0b101 = Connecting Modules  
 0b10 = Running 0b110 =   
 0b11 = Failure 0b111 =

### Restart application • •

Restart the control application

0x9F

### Admin

### Emergency Stop, Admin authority • •

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

0xC0 Admin Code 2 bytes (0-65534)

### Emergency Release, Admin authority • •

Release Emergency stop.

0xC1 Admin Code 2 bytes (0-65534)

### Disable Admin authority for this connection (LOGOUT) •

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

0xCE

### Enable Admin authority for this connection (LOGIN) •

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

*From client to server*

0xCF Password hash length Password hash

*From client to server, Login required otherwise disconnect*

0xCF

## 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 Update • •

*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 | C | D | E | F | data |

Uint16 A = DCC ID (little endian)

Uint16 B = Length (mm)

C = Flags (Speedsteps/Type)

D = Image/icon filetype (0x00: png/png, 0x01: png/jpg, 0x10: jpg/png, 0x11: jpg/jpg

E = Name length

F = Nr of calibrated speedsteps

Data contains name and speedsteps

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)

### Update engine in library •

Client to server

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x51 | ABBB BBBB | BBBB BBBB | Uint16 C | Uint16 D | E | F | G | H | data |

Axxx xxxx = 0 Edit / 1 Remove

B = uid

Next not supplied if A is remove

Uint16 C = DCC ID (little endian)

Uint16 D = Length (mm)

E = Flags (Speedsteps/Type)

F = Image/icon filetype (0x00: png/png, 0x01: png/jpg, 0x10: jpg/png, 0x11: jpg/jpg

G = Name length

H = Nr of calibrated speedsteps

Data contains name and speedsteps

### Engines Library •

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

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

Data contains name, paths and speedsteps

Repeated for each engine

### Add car to library ••

Client to server

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x53 | Uint32 A | Uint16 B | Uint16 C | D | E | F | H | data |

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

Data = contains Name

Server to client

|  |  |  |
| --- | --- | --- |
| 0x53 | Uint16 A | B |

Uint16 A = Nr (little endian)

B = Return value (0 = Failed, 1 = Success)

### Update car in library • •

Client to server

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x54 | ABBB BBBB | BBBB BBBB | Uint16 C | Uint16 D | E | F | G | H | data |

Axxx xxxx = 0 Edit / 1 Remove

B = uid

Next not supplied if A is remove

Uint16 C = DCC ID (little endian)

Uint16 D = Length (mm)

E = Flags (Speedsteps/Type)

F = Image/icon filetype (0x00: png/png, 0x01: png/jpg, 0x10: jpg/png, 0x11: jpg/jpg

G = Name length

H = Nr of calibrated speedsteps

Data contains name and speedsteps

Server to client

|  |  |  |
| --- | --- | --- |
| 0x53 | Uint16 A | B |

Uint16 A = Nr (little endian)

B = Return value (0 = Failed, 1 = Success)

### Cars Library •

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x55 | Uint16 A | Uint16 B | Uint16 C | D | E | F | H | data |

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 ••

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0x56 | A | BCCC CCCC | D | data |

A = Name length

Bxxx xxxx = Save

xCCC CCCC = Number of Rolling stock in train

D = Type

Data contains name and train composition

Server to client

|  |  |
| --- | --- |
| 0x56 | A |

A = Return value (0 = Failed, 1 = Success)

### Update train in library • •

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0x57 | ABBB BBBB | BBBB BBBB | D | EFFF FFFF | G | data |

Axxx xxxx = 0 Edit / 1 Remove

B = uid

Next not supplied if A is remove

D = Name length

Exxx xxxx = Save

xFFF FFFF = Number of Rolling stock in train

G = Type

Data contains name and train composition

Server to client

|  |  |
| --- | --- |
| 0x57 | A |

A = Return value (0 = Failed, 1 = Success)

### Train Library •

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x58 | Uint16 A | Uint16 B | C | D | E | F | name | Links | |
| Type | ID |

Uint16 A = Max speed

Uint16 B = Length (mm)

C = Type

xxxx xxxD = 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 Data • •

From server to client only

Send the visual layout to the client

|  |  |  |
| --- | --- | --- |
| 0x30 | ModuleID | Data |

### Station Library • •

0x32

## Client / 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

xxxB BBBB BBBB BBBB = Message ID

### Message Clear •

0x15 Message ID uint16\_t

AAAx xxxx = return code (0 = failed, 1 = success, other = errorcode)  
 xxxB BBBB BBBB BBBB = Message ID

### 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 |

x8000 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

# 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 master*

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

|  |  |  |
| --- | --- | --- |
| 0x01 | DevID | 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 | DevID | 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 |

## IO

### Set Address

From master to device

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0x10 | DevID | BBBB BBBB | CCCC DDDD | Checksum |

A = Length flag  
Address = B + (C << 8)  
D = State

### Notify Updated Input

From device to master

Notify a state change on a certain address.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0x11 | DevID | BBBB BBBB | CCCC xxxD | Checksum |

A = Length flag  
Address = B + (C << 8)  
D = State

### Request Read all States

|  |  |  |  |
| --- | --- | --- | --- |
| 0x13 | DevID | NodeID | Checksum |

***Response:*** Read all states (0x07)

### Response Read all states

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0x12 | Length | DevID | AAAA BBBB | CCCC DDDD | … | Checksum |

A = Node ID, B = Address 1, C = Address 2, D = Address 3, …

## 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 |

### Change Slave ID

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0x51 | Dev ID | Old sID | New sID | Checksum |

### Set blink pattern

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0x52 0x53 | Dev ID | Node | Timing On1 | Timing Off1 | Timing On2 | Timing Off2 | Checksum |

Timing consist of a sequence where the output is turned on and off.  
So 10, 10, 0, 0 will be a one Hertz blink (500ms on, 500ms off)  
So 10, 10, 20, 20 will be a blink of 500ms on, 500ms off, 1000ms on, 1000ms off

ID 0x52 for pattern 1, ID 0x53 for pattern 2

### 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

|  |  |  |
| --- | --- | --- |
| Device ID | Uint8 |  |
| Node id | Uint8 |  |
| IO | Struct IO\_EEPROM | Repeat IO nr |

Struct IO\_EEPROM

Type: output/servo/pwm/toggle\_in/push\_in

Only output Blinking A, blinking B, pulse

Only servo/pwm state1, state2, state3, state4