



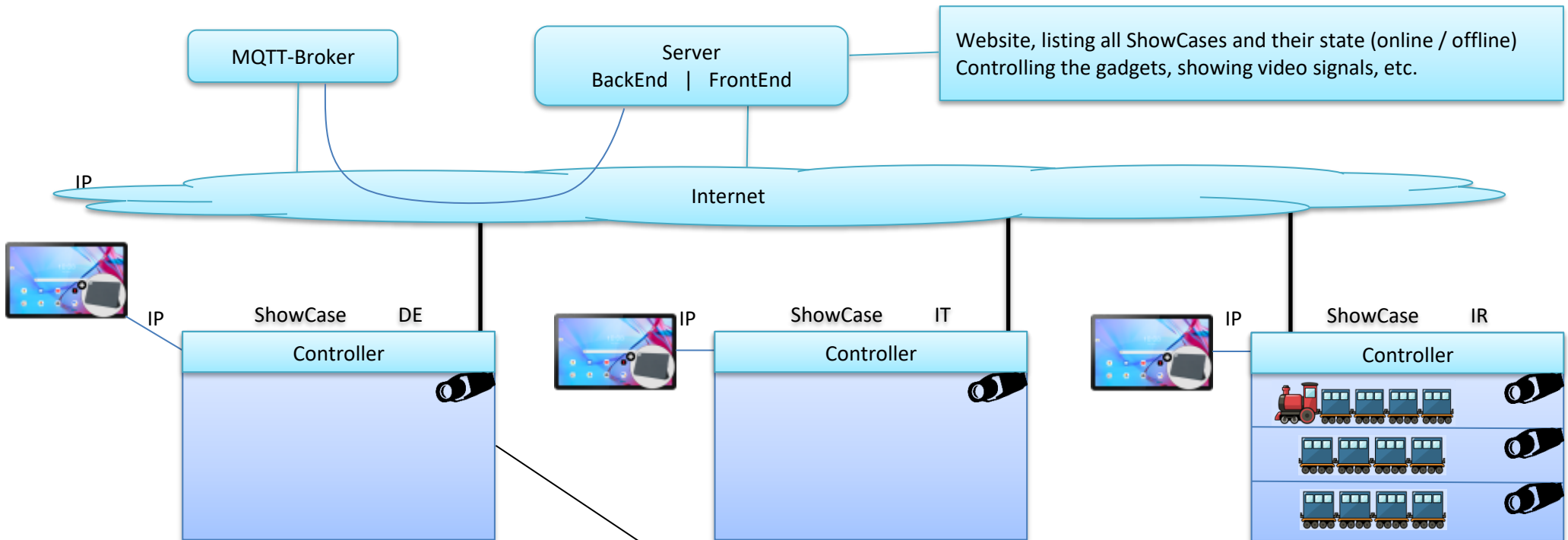
**European CNC-Network**  
**Train for Europe – Digital Revolution 4.0**  
**Technical Specification**

# 1 Overview

The idea of the project is to bring some animations (Led, motors, servos, whatever) into the already existing wagons of the „Train for Europe“.

## Inhalt

1	Overview.....	2
1.1	Details:.....	5
1.2	Server.....	6
1.3	MQTT-Broker.....	6
2	Hardware.....	7
2.1	Why PCF8575 replacement.....	7
2.2	Overview.....	7
2.3	Schematic.....	8
2.4	Board.....	9
2.5	Bill of material.....	10
2.6	Wiring.....	10
2.6.1	Bill of material.....	10
2.7	Pin assignments PCF8575 and Arduino.....	12
2.8	RS485-Node.....	13
3	Software.....	14
3.1	MQTT-Message flow.....	14
3.1.1	Setup.....	14
3.1.2	Handshaking.....	16
3.1.3	Starting and stopping any animation at a wagon.....	17
3.2	Addressing scheme.....	18
3.2.1	Mapping.....	18
3.3	Animations.....	19
3.3.1	Steps to build your own animation.....	20
4	Wagons.....	21
4.1	Used wagons in the train.....	23
4.2	Wagon DK (MP3-Player).....	23
4.3	Wagon BE.....	24
4.4	Wagon SE.....	24
4.5	Wagon RO.....	24
4.6	Wagon DE.....	24
5	Sources (Hardware + Software).....	25
6	Sources (Datasheets).....	26



MQTT-Client talking to MQTT-Broker

Global – Consensus to all the global issues required

Local – No consensus required after the ESP at the loco / first wagon

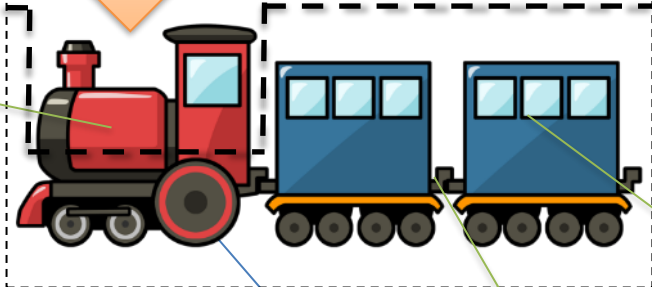
Loco Controller talking with ShowCase controller via MQTT and all wagons e.g.: Arduino, ESP, ESP32...

per wagon  $\mu$ C, talking with loco, controlling the gadget e.g.: Arduino, ESP, MCP23016, ...

Power supply:  
 opt.1 (mobile): Battery in Loco: e.g. 12V / 7Ah  
 opt.2 (stationary): Power Supply e.g. 12V / 2A

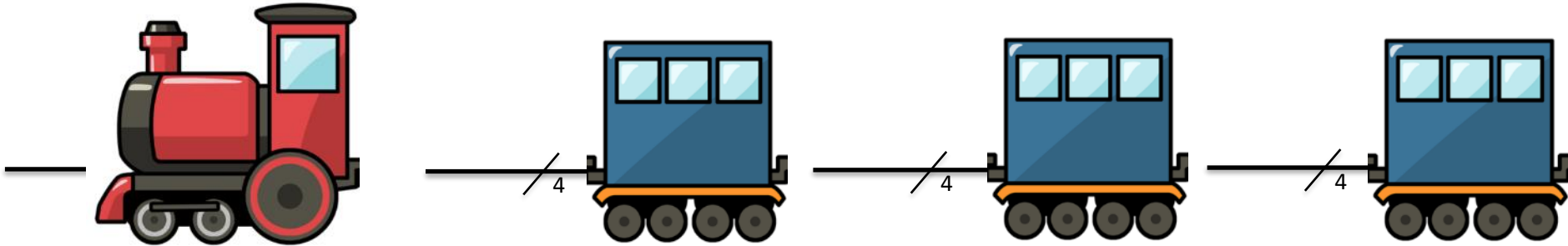
power & communication (4 wire) e.g. CAN2.0 | I<sup>2</sup>C | ... any other 2-wire-communication

MQTT-Client



## 1.1 Details:

Different options have been discussed, how to connect the wagons to the loco resp. to the ESP32 at the first wagon. Here is a summary of the discussion.



### Option #1: suggested by Team Italy

ESP32 at the loco (or the first wagon, if there is no loco) connected to the Showcase-Controller using MQTT via Wi-Fi  
Connected to the other wagons using I<sup>2</sup>C-Bus

I<sup>2</sup>C-Bus  
+ 5V, GND  
means 4 wires  
are sufficient

I-O-Extender like PCF8574, PCF5875, MCP23016 or similar to control the electronic gadget

I<sup>2</sup>C-Bus  
+ 5V, GND  
means 4  
wires are  
sufficient

I-O-Extender like PCF8574, PCF5875, MCP23016 or similar to control the electronic gadget

I<sup>2</sup>C-Bus  
+ 5V, GND  
means 4  
wires are  
sufficient

I-O-Extender like PCF8574, PCF5875, MCP23016 or similar to control the electronic gadget

Pros: simple, cheap

Cons: can be problematic, if the train is long, because I<sup>2</sup>C was not designed for long wires (should be tested with a prototype), can be solved with bus-extenders (e.g. LTC4311)  
restricted address range to 7 or 8 devices  
no advanced timing possible at the wagon (at least difficult)

### Option #2: suggested by Team Germany

ESP32 at the loco (or the first wagon, if there is no loco) connected to the Showcase-Controller using MQTT via Wi-Fi  
Connected to the other wagons using RS485-Bus and ModBus-Protocol

RS485-Bus  
+ 12V, GND  
means 4 wires  
are sufficient

- Arduino Nano Every  
- MAX481 Bus-Transceiver  
- ModBus-Client here  
- own  $\mu$ C to realize any kind of electronic gadget

RS485-Bus  
+ 12V, GND  
means 4  
wires are  
sufficient

- Arduino Nano Every  
- MAX481 Bus-Transceiver  
- ModBus-Client here  
- own  $\mu$ C to realize any kind of electronic gadget

RS485-Bus  
+ 12V,  
GND  
means 4  
wires are  
sufficient

- Arduino Nano Every  
- MAX481 Bus-Transceiver  
- ModBus-Client here  
- own  $\mu$ C to realize any kind of electronic gadget

Pros: no length restrictions due to the RS485-Bus  
no restrictions to the number of devices  
own DC-DC-Converter at the Arduino-Boards reduces noise at the power supply lines

Cons: more expensive because of the additional  $\mu$ C in each wagon  
higher complexity due to multiple controllers

## 1.2 Server

Beside the website itself, ...

Webserver ??

MongoDB (version ??)

Go-Program from Denis

## 1.3 MQTT-Broker

Actually, we are using <https://www.hivemq.com/> as MQTT broker. There is no special requirement to this broker, so any other will also fit our needs.

We also tested: <https://www.emqx.io/> as well as a local instance of [Mosquito](#).

We are using anonymous access, so there is no need for a certificate at the ESP.

```
const char *mqtt_broker    = "broker.hivemq.com";
const char *mqtt_username  = "";
const char *mqtt_password  = "";
const int   mqtt_port      = 1883;
```

See section 3.1 for a detailed description of the message flow between the ESP and the broker.

## 2 Hardware

### 2.1 Why PCF8575 replacement

In the first meeting, the group decided to use PCF8575 port expanders inside the wagons.

The PCF8575 is a 16Bit port expander for the I<sup>2</sup>C-Bus. A maximum of 8 devices can be used per I<sup>2</sup>C bus. More devices require the use of an I<sup>2</sup>C multiplexer. Another option is to replace the port expander by a microcontroller like Arduino. The processing capabilities will allow much more complex animations and the addressing can be from 0... 255.

The port expanders are very limited and not suitable for more complex animations. They can be used for very simple animations only.

### 2.2 Overview

Because of the lack of internal configuration registers (like MCP23016), the device is quite simple. After the initial write sequence transporting the device address, every subsequent byte written will be placed in the output latch A bevor B. The operation is module 2. Writing 6 bytes will have the same result then writing only the last 2 bytes.

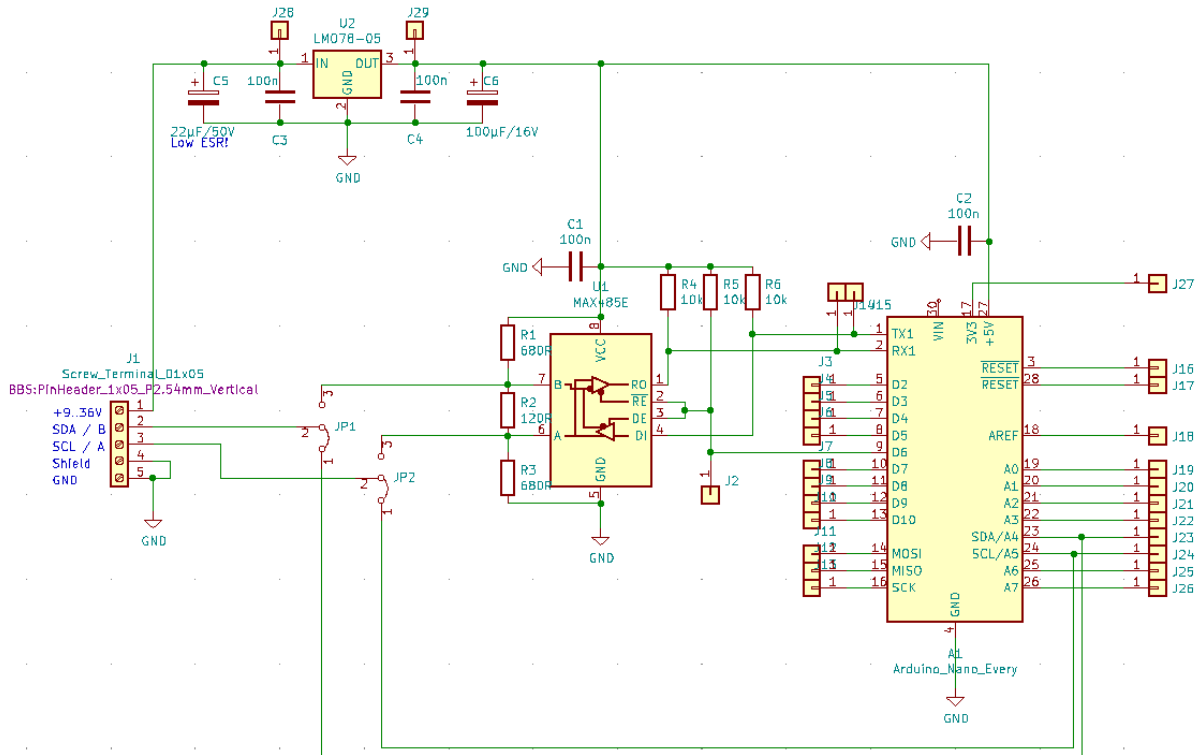
The firmware to emulate the PCF8575 is based on the WIRE library, which comes with Arduino. The idea is to be able to replace a PCF8575 without making any changes to the I<sup>2</sup>C-Master. Just setting the correct address at the slave. The address must be set in the source code and cannot be changed by external settings like the PCF8575 can.

The controller board used here comes with real push-pull-outputs, where the PCF8575 uses only low-side switches. The Controller used is an Arduino Nano Every. It is cheap, has a small footprint and supports 2 serial ports by hardware, so one can be used for serial monitor and the other is used for RS485 device transceiver. The two flavors of the board are:

- a) As a replacement of the PCF8575 16 Bit I<sup>2</sup>C bus expander. In this mode, the RS485-Transceiver will not be used. The socket for this chip can be left open, R1... R6 are not needed.
- b) As a node on a RS485-Network. Based on this physical layer, a ModBus-Client will be placed. ModBus is a very famous protocol, widely used in industry. Libraries for Arduino are also available. It allows a much more sophisticated control of the wagon, than a simple bus expander can provide.

The following description covers both scenarios. If all parts are placed, you can simply change the usage by changing the two jumpers JP1 and JP2. See section board for details.

## 2.3 Schematic



### Changes to Rev.0:

- DC-DC-Converter replaced by a cheaper device with smaller footprint
- Additional holes to connect to +U<sub>b</sub> and +5V for other equipment inside the wagon
- Add 3 Mounting holes
- Add Version number at the soldering side



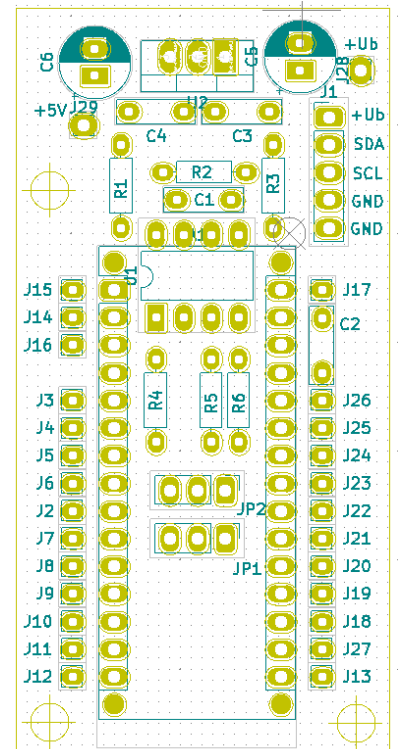
## 2.4 Board

To allow a small footprint, most of the parts were placed underneath the controller board. We do not use SMD packages to keep soldering simple.

The dimensions of the pcb are: 34,5 x 68,6 mm

Layout changes slightly to avoid vias. Now there are only 2 vias without a part. If you produce the pcb yourself, take care of these two vias and solder a short piece of wire on both sides to establish a connection through the hole.

If you order the pcb, the manufacturer will connect through all holes.



FS1	A Step-Down-Converter with a 5V / 500mA output and a wide input-range from 6,5V..36V. Using a DC-DC-Converter per node will eliminate noise coming along the main power supply from the loco. One power supply will fit all needs.
JP1, JP2	These two jumpers connects terminal 2/3 to either the RS485 device or the SDL/SCL lines of the Controller. The jumpers must only be changed, if the firmware at the node changes. So if you do not plan to change the operation mode, you can use 2 short wires as well. 1-2 = I <sup>2</sup> C mode 2-3 = RS485 mode
Address	To spare I/O-pins, the address of the device must be set via source code. By doing so, any address in the range from 0x01 to 0xFF is possible. In RS485-mode, it will be possible to change the address via ModBus protocol.
R1,R3	They provide a save potential at the lines A and B, if no sender is active. These resistors are only for safe operation and must not be placed several times. Place them at the first node in the chain and only, if you are using RS485 mode.
R2	Terminating resistor. Must correspond with the impedance of the wires. 120 Ohm will fit most needs. Needed only at the beginning and the end of the bus and only, if you are using RS485 mode.

Schematic as well as the board were made with KiCAD, with is available for free (see <https://www.kicad.org/> ).

## 2.5 Bill of material




All parts can for this pcb can be ordered at Reichelt. See: <https://www.reichelt.de/my/1984952>

Reichelt is a German distributor. Maybe you prefer to order at Amazon o.e.

## 2.6 Wiring

We agreed to use small magnetic connectors.

### 2.6.1 Bill of material

Nr	Source	Picture
1	<a href="#">Magnetic connector</a> , 1 pair per wagon  These connectors allow a max. current of 2 A. If one wagon draws the full current allowed by the DC-DC-Converter (0,5 A), we are able to run 10 animations at the same time. This seems to be sufficient.	
2	<a href="#">Wires for the power lines</a>  A closer look at the requirements for the power supply cables resulted in the above mentioned conductor diameter. With fewer wagons, the diameter can be smaller. There is a small Excel file to address this issue (→ Wiring.xlsx).	
3	<a href="#">Wire for Data lines</a>  When used as an RS485 node, the data line must be twisted in pairs. The conductor diameter, on the other hand, is of no particular importance.	

### **Warning:**

A short cut between the power supply pin (+12V) and one of the ESP's data pins (SDA, SCL) will immediately destroy the ESP!

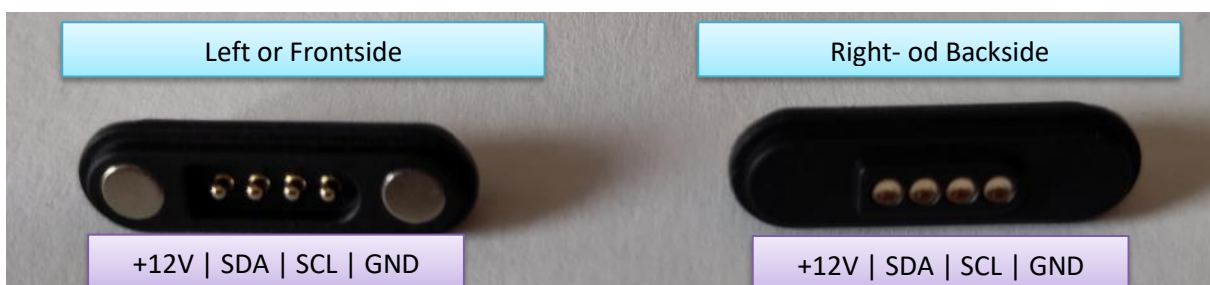
These are suggestions only. The explanatory text explains only briefly how we came to the selection.

To ensure interoperability, we also need to agree on a common physical layer. This means primarily orientation and signals on each pin of the connector. The next picture shows our choice.

### Note:

This interoperability between wagons of different schools is nice to have, but not really necessary, because it will typically not happen that wagons of different schools are mixed within one showcase.

The used assignment by the german team:

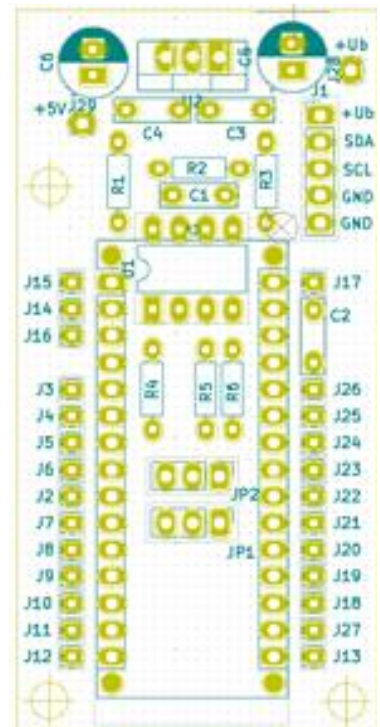




## 2.7 Pin assignments PCF8575 and Arduino

The Arduino-Pins are assigned to the bits of the PCF8575 and the PCB connectors as follows:

PCF8575 Register A	Arduino Pin-Nr	Connector @ PCB	Connector @ PCB	Arduino Pin-Nr	PCF8575 Register B
		J15 (TX1)	J17(/Reset)		
		J14 (RX1)			
		J16(/Reset)			
Bit 0	2	J3 (D2)	J26 (A7)	10	Bit 0
Bit 1	3	J4 (D3)	J25 (A6)	11	Bit 1
Bit 2	4	J5 (D4)	J24 (A5/SCL)	12	Bit 2
Bit 3	5	J6 (D5)	J23 (A4/SDA)	13	Bit 3
Bit 4	6	J2 (D6)	J22 (A3)	A0	Bit 4
Bit 5	7	J7 (D7)	J21 (A2)	A1	Bit 5
Bit 6	8	J8 (D8)	J20 (A1)	A3	Bit 6
Bit 7	9	J9 (D9)	J19 (A0)	A3	Bit 7
		J10 (D10)	J18 (AREF)		
		J11 (MOSI)	J27 (+3V3)		
		J12 (MISO)	J13 (SCK)		



The assignment is important for wiring the device. It can be easily changed in the firmware. The Pin numbers are collected in two arrays, named `portA` and `portB`, see → `Test_PCF8575_Emu`

```
uint8_t portA[] = {2, 3, 4, 5, 6, 7, 8, 9};
uint8_t portB[] = {10, 11, 12, 13, A0, A1, A2, A3};
```

This will allow to simulate part of the functionality of a port expander with this board. See out [Git](#) for some test program to demonstrate this.

## 2.8 RS485-Node

RS485 is a differential 2-wire bus system widely used in industry.

The functionality was tested but will not be used in this project.

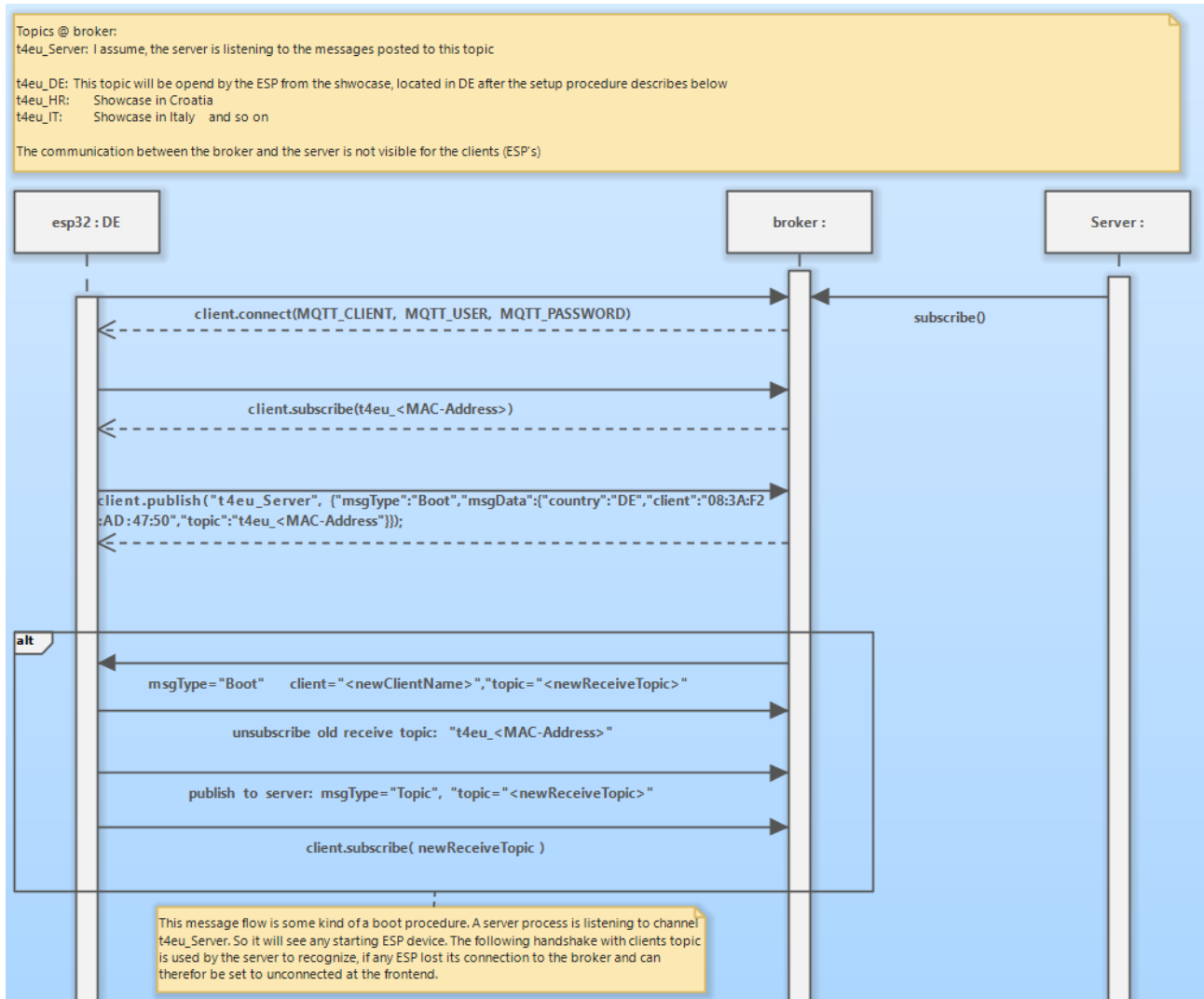
The I<sup>2</sup>C bus seems to work over the distances we need.

### 3 Software

#### 3.1 MQTT-Message flow

Message exchange between all subsystems is done via MQTT. There is no need for a special broker. Any free service can be used. There is a message flow sequence on application level to establish a connection between the ESP and the broker / server. This flow is as follows.

##### 3.1.1 Setup



##### 3.1.1.1 Prerequisites

If the ESP is powered on, he first establishes a connection to the configured Wi-Fi and, if successful, establishes a connection to the preconfigured MQTT broker. We assume, this steps succeeds.

- In the following example we set the location of the ESP to DE (Germany).  
Location = DE
- Sometimes, the MAC-Address of the Wi-Fi interface of the ESP is needed. In this test case, it is:  
MAC = C8:C9:A3:C5:E5:A4

##### 3.1.1.2 Connection

- ESP subscribes to a topic. The name of this topic is built by the prefix t4eu\_ and the MAC-Address of the Wi-Fi interface of the ESP itself.  
topic = t4eu\_C8:C9:A3:C5:E5:A4

- b) ESP sends a JSON encoded message to the predefined topic t4eu\_Server. The message is as follows:

```
{  "msgType": "Boot",
  "msgData": {
    "country": "DE",
    "client": "C8:C9:A3:C5:E5:A4",
    "topic": "t4eu_C8:C9:A3:C5:E5:A4"
  }
}
```

- c) A server process is also listening to this topic and will receive the message. This server process will send a message back to the ESP with some new topic. The ESP will receive this message the topic he subscribed in step a)

Return message looks like this:

```
{  "msgType": "Boot",
  "msgData": {
    "country": "DE",
    "client": "t4e_DE",
    "topic": "t4eu_DE"
  }
}
```

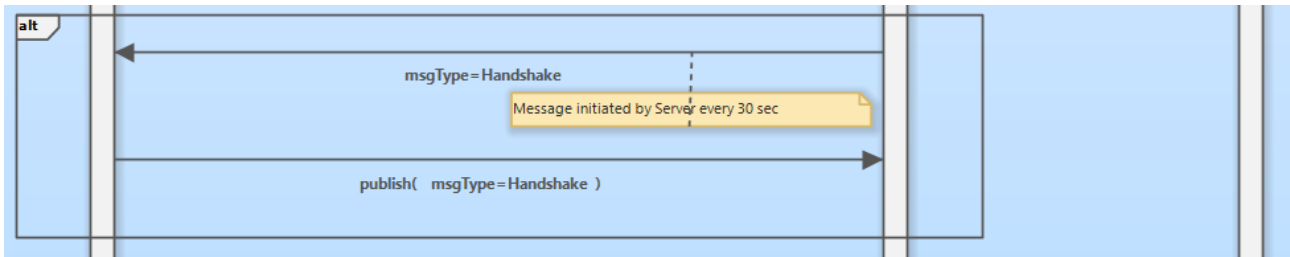
- d) ESP now unsubscribes from the topic, it enrolled in step a) and subscribes to the topic provided by the server process. Next step is to inform the server, that these steps are done. So the ESP publishes a JSON message to the topic t4eu\_Server. The message looks like this:

```
{  "msgType": "Topic",
  "msgData": {
    "country": "DE",
    "client": "t4eu_DE",
    "topic": "t4eu_DE"
  }
}
```

After this kind of a 3-way-handshake, the connections between the ESP, the broker and the Server are established.

### 3.1.2 Handshaking

After the connection is established, a handshake process will take place. The server will send a message every 30 s. The client must answer this message to signal, he is still alive.



After running through the process discussed before, the server is publishing a message to the topic `t4eu_DE` every 30 seconds to see, if the ESP is still alive.

ESP receives a message from the Server via topic `t4eu_DE`:

```
{
  "msgType": "Handshake",
  "msgData": {
    "sender": "server"
  }
}
```

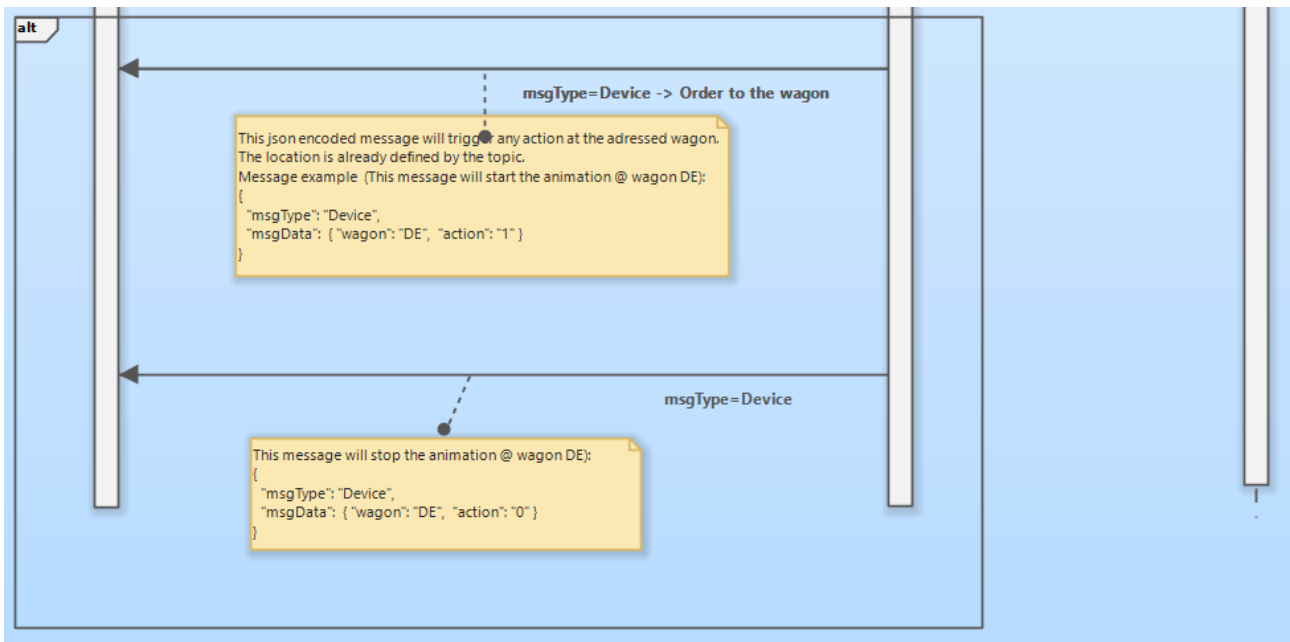
ESP answers with a message like this published to the topic `t4eu_Server`

```
{
  "msgType": "Handshake",
  "msgData": {
    "sender": "client",
    "country": "DE"
  }
}
```



### 3.1.3 Starting and stopping any animation at a wagon

Any animation is triggered by the server with a Device – message.



This type of message is sent to the topic t4eu\_DE to trigger an action on a specific wagon of the train located in DE. The topic itself already determines the addressed showcase / country.

Example message to start an animation:

```
{
  "msgType": "Device",
  "msgData": {
    "wagon": "DE",
    "action": "1"
  }
}
```

Example message to stop the animation:

```
{
  "msgType": "Device",
  "msgData": {
    "wagon": "DE",
    "action": "0"
  }
}
```

## 3.2 Addressing scheme

Actually, the parameters send to the ESP via MQTT are as follows:

wagon	a 2-letter country code addressing the wagon in the train and not the location of the train / showcase. This is already done by the topic.
action	a value between [0 1] is allowed 0 = Stop the animation 1 = Start the animation

The server generates two messages. The first one with `action=1` will start the animation. The animation will run as long as the server sends another message with `action=0`. This will stop the animation. The time between these two messages is now 15 s.

Maybe an upper time limit will be helpful, if the second message is lost for whatever reason. The German ESP implementation uses a second parameter called `val`. Provide an upper time limit here.

### 3.2.1 Mapping

Animations in our implementation for the ESP uses the following variables.

<code>uint16_t wagon</code>	The value for wagon is derived from the 2-letter country code (e.g. IT, HR, DE) by a simple algorithm: <ul style="list-style-type: none"><li>- Convert each letter to its ASCII code</li><li>- Shift the value of the first letter to the left by 8 bit positions</li></ul> By this, each wagon is identified by a 16bit value, which is easier to deal with than a string.
<code>uint8_t cmd</code>	The actual implementation maps the parameter <code>action</code> directly to <code>cmd</code> . This was done to prevent multiple changes to the animations just to change the name of the parameter. Only 2 values have a fixed meaning. 0 = Stop the running animation 1 = Start an animation 2 . . 255 = The meaning of these values depends on the special animation.
<code>uint8_t val</code>	The actual implementation will carry the time limit, if the second MQTT request from the server is lost. This will prevent the animation to run forever.

### 3.3 Animations

Due to the lack of processing capabilities by the port expander, every animation using the PCF8575 have to be done by the ESP. To keep things simple for other programmers with their own ideas of animation, we implemented a base class `TAnimationBasic`, where the basic stuff is done and derived some special classes from this base class for the concrete animation. There are some simple animations available. They should give an idea, how things are working.

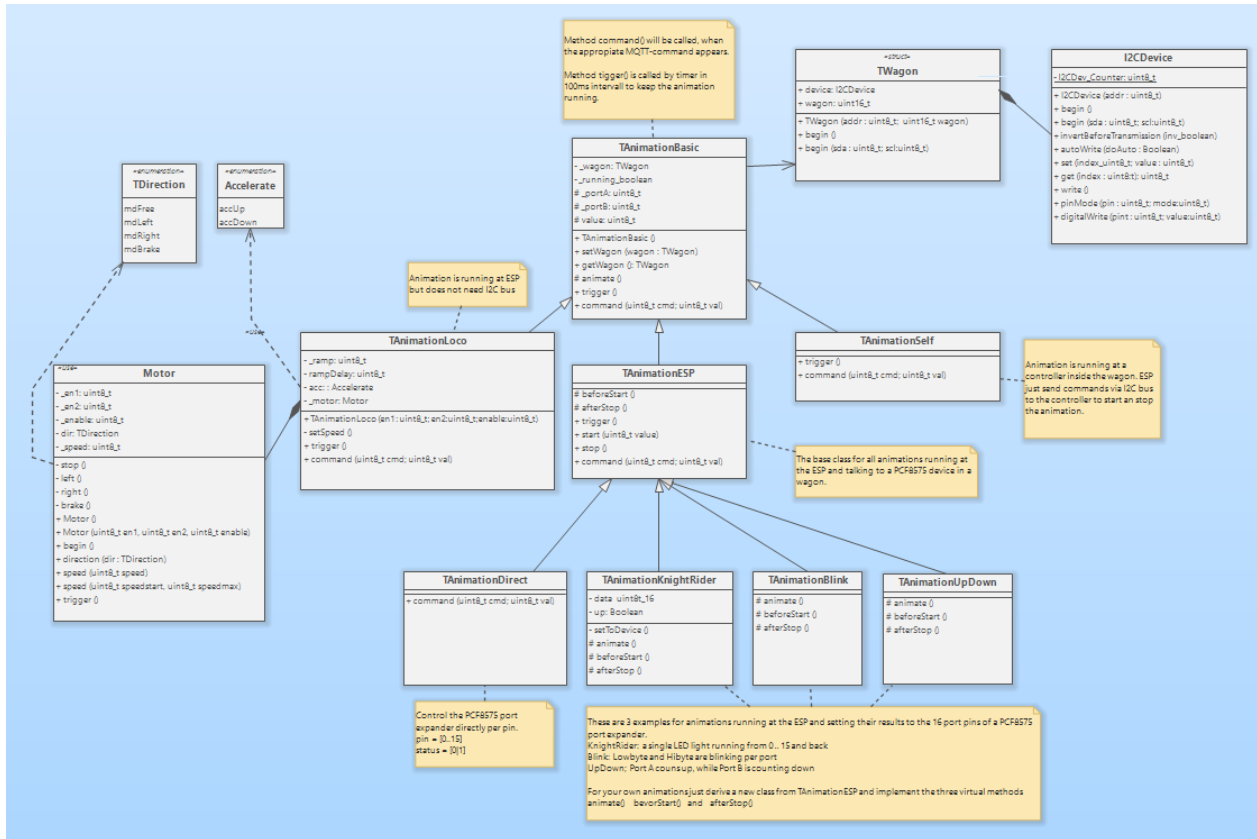


Abbildung 1: Class diagram, see `./git/loco/Loco_01/Loco_01.simp`

This diagram was made with [Software Ideas Modeller](#), which is free for non-commercial use. You find the file in the folder `./doc` in the [git repository](#).

- `TAnimationBasic`      The base class for all animations, so that we can store all those objects in a list or an array.
- `TAnimationLoco`      An example for a class of animation, which is running at the ESP but do not use the I<sup>2</sup>C bus. It uses an object motor, which can control a DC motor.
- `TAnimationESP`      This is the base class for all animations running at the ESP and using a PCF 8575 port expander inside a wagon. 4 Example are available:
  - `TAnimationDirect`      The appropriate pin at the port expander is switched on/off <sup>1</sup>
  - `TAnimationKnightRider`      One light is running from one end to the other and back.
  - `TAnimationBlink`      Low nibble and high nibble of each port are switched on and off alternatively.

<sup>1</sup> With the actual version of the protocol, this animation is deprecated because there is no parameter "pin" and "status" any longer.

TAnimationUpDown      Both ports are treated as 8 bit counters. Port A counts upwards while Port counts downwards.

TAnimationSelf      This is the base class for all animations running on a special microcontroller inside a wagon. The two bytes `cmd` and `val` are transparently transported via the I<sup>2</sup>C bus to the wagon's microcontroller. There is no meaning for these two bytes outside the addressed controller.

### 3.3.1 Steps to build your own animation

#### 3.3.1.1 Based on a PCF8575 port expander

- Derive a new class from TAnimationESP
- Implement these three virtual methods;
  - `beforeStart()`    - preparing all you need, like `setup()`
  - `animate()`        - will be called regularly every 100ms as long as the animation is running.  
Do your stuff here like you normally do in `loop()`
  - `afterStop()`      - clean up the scene, when your animations ends, e.g. switch off all Leds etc.

#### 3.3.1.2 Based on a microcontroller in a wagon

Like Arduino nano, Arduino nano Every or similar

- Normally no special things needed at the ESP.  
Just use the class TAnimationSelf. You just have to derive your own class if you need some more interaction with the ESP like a data transmission from the wagon back to the ESP.

#### 3.3.1.3 Based on the ESP directly

- Take the class TAnimationLoco as an example and derive your own class directly from TAnimationBasic. At least you have to override the method `command()` to receive the values from the ESP via I<sup>2</sup>C. Keep in mind, that this animation is directly running on the ESP and needs therefore some resources from it.

## 4 Wagons

Our train consists of 24 wagons. This no is defined in File: wagon.h

You only have to increase this value, if you have more than 25 objects in the train (1 loco and 24 wagons).

```
#define NR_OF_WAGONS 25
```

The enumeration of all available country codes is just for a better reading of the code.

```
enum CountryCode {  
    L1,  
    IT, NO, IR, HR, LT, DE,  
    DK, PT, CH, X1, X2, NL,  
    TR, AT, PL, FR, SK, SI,  
    BE, SE, RO, X3, X4, X5  
};
```

L1 means Loco 1, X1...X4 are unknown at the moment.

The array countries hold the 2 letter country codes. The nr of elements as well as the sequence of the elements have to match with the enum mentioned before.

```
const char countries[NR_OF_WAGONS][3] = {  
    "L1",  
    "IT", "NO", "IR", "HR", "LT", "DE",  
    "DK", "PT", "CH", "X1", "X2", "NL",  
    "TR", "AT", "PL", "FR", "SK", "SI",  
    "BE", "SE", "RO", "X3", "X4", "X5"  
};
```

The array allWagons[] contains the assigned I<sup>2</sup>C bus addresses and the country code converted to a 16 bit value. See File: wagon.cpp

```
TWagon allWagons[ NR_OF_WAGONS ] = {  
    {0x10, convertCountryCode( countries[L1] )},  
    ...  
    {0x00, convertCountryCode( countries[LT] )},  
    {0x31, convertCountryCode( countries[DE] )},  
    {0x32, convertCountryCode( countries[DK] )},  
    {0x00, convertCountryCode( countries[PT] )},  
    ...  
    {0x00, convertCountryCode( countries[SI] )},  
    {0x20, convertCountryCode( countries[BE] )},  
    {0x22, convertCountryCode( countries[SE] )},  
    {0x24, convertCountryCode( countries[RO] )},  
    ...  
    {0x00, convertCountryCode( countries[X5] )}  
};
```

0x00 means: not used at the moment

Any other value is the corresponding I<sup>2</sup>C bus address of the device, for example 0x20 ... 0x24 for the PCF 8575 port expanders

The step before just assigned a country code to a I<sup>2</sup>C bus address.

The last step is the assignment of an animation to a specific I<sup>2</sup>C bus address. This is done in File `main.cpp` in the following method:

```
void assignAnimations() {
    animation[ BE ] = new TAnimationKnightRider(); //create an animation object
    animation[ BE ]->setWagon( &allWagons[ BE ] ); //assign a wagon to this animatio

    animation[ SE ] = new TAnimationBlink();
    animation[ SE ]->setWagon( &allWagons[ SE ] );
    ...
}
```

Only 2 lines of code are required to create an animation object and assign this object to a specific wagon.

No more changes are needed to adopt the ESP firmware to your needs.

For example:

The wagon DE is assigned with the animation `TAnimationLoco`. This animation does not communicate with the I<sup>2</sup>C bus and does therefore not need an I<sup>2</sup>C address. It is set to `0x00`.

The wagon BE, SE and RO are used in the simulation with a port expander. So we need an I<sup>2</sup>C address here. The following table lists all the wagons.

The animation for each wagon is defined in the method `assignAnimations()` in `main.cpp`. You can assign any available animation to any listed wagon.

## 4.1 Used wagons in the train

The intention of this table is to provide an overview of the I<sup>2</sup>C addresses already in use. It corresponds to the german showcase.

	Device	Adress	
0	L1	0x10	Loco 1 Mapped to wagon L1
	L2		Loco 2
		0x3C	Display @ the Loco
1	IT		Olivetti, Dilara
2	NO		Segelboot mit Fahne, Yannick
3	IR		Irland
4	HR		Kroatien, Radio, Plexiglasschild, Jana
5	LT		Litauen
6	DE	0x31	Deutschland, Gutenberg Druckerpresse
7	DK	0x32	Dänemark, Musik mp3
8	PT		Portugal, Segelboot, Marvin
9	CH		Schweiz, Uhr, Kreuz, Krystian
10	X1		Karusell, Zoe
11	X2		Modellzug, Darwin
12	NL		Niederlande, Musik
13	TR		Türkei, Brücke
14	AT		Österreich
15	PL		Polen
16	FR		Frankreich, Satellit
17	SK		Slowakei, Schlagbaum
18	SI		Slowenien,
19	BE	0x20	Belgien, Bierfass
20	SE	0x22	Schweden
21	RO	0x24	Rumänien
22	X3		
23	X4 IT		2. Waggon, Turm Pisa
24	X5 NO		2. Wagon Bohrinsel

## 4.2 Wagon DK (MP3-Player)

I2C-adress	0x32	Arduino Nano Every
wagon	DK	
cmd	0:	Stop the animation
	1:	Start the animation, sets val=255
	2..255:	tbd
val		Duration for the animation to run
		0 = no time limit
		1..255 = timelimit in steps of 100ms

### 4.3 Wagon BE

I2C-adress	0x20 Port expander PCF8575
wagon	BE (Belgium) Simulation – a PCF8575 port expander fully equipped with 16 Leds
cmd	0: Stop the animation 1: Start the animation, sets val=255 2..255: dc
val	Duration for the animation to run 0 = no time limit 1..255 = timelimit in steps of 100ms

dc = don't care

### 4.4 Wagon SE

I2C-adress	0x22 Port expander PCF8575
wagon	SE (Sweden) Simulation – a PCF8575 port expander fully equipped with 16 Leds
cmd	0: Stop the animation 1: Start the animation, sets val=255 2..255: dc
val	Duration for the animation to run 0 = no time limit 1..255 = timelimit in steps of 100ms

dc = don't care

### 4.5 Wagon RO

I2C-adress	0x24 Port expander PCF8575
wagon	RO (Romania) Simulation – a PCF8575 port expander fully equipped with 16 Leds
cmd	0: Stop the animation 1: Start the animation, sets val=255 2..255: dc
val	Duration for the animation to run 0 = no time limit 1..255 = timelimit in steps of 100ms

dc = don't care

### 4.6 Wagon DE

I2C-adress	0x31 MicroController Arduino Nano Every
wagon	DE (Germany) Gutenberg Letterpress
cmd	0: Stop the animation 1: Start the animation, sets val=255 2..255: tbd
val	Duration for the animation to run 0 = no time limit 1..255 = timelimit in steps of 100ms

dc = don't care



## 5 Sources (Hardware + Software)

### Hardware

PCF8575 datasheet

<https://www.ti.com/lit/gpn/PCF8575>

MCP 23016 datasheet

<https://ww1.microchip.com/downloads/en/DeviceDoc/20090C.pdf>

RS485 Bus transceiver

<https://datasheets.maximintegrated.com/en/ds/MAX1487-MAX491.pdf>

Arduino Nano Every

<https://docs.arduino.cc/hardware/nano-every>

### Software

Possible useable library for PCF8575

[https://github.com/xreef/PCF8575\\_library](https://github.com/xreef/PCF8575_library)

We do not use this library, because of its limitations. We use the Wire library instead.

Arduino-Client for MQTT

<https://github.com/knolleary/pubsubclient>

JSON-Library for Arduino

<https://github.com/bblanchon/ArduinoJson>

Our repository at github

<https://github.com/T4EU-Rev4>

Italian teams repository for server and frontend as well as there ESP implementation (not public)

<https://github.com/CristianAlasotto/TrainForEurope>

## 6 Sources (Datasheets)

Camera:

<https://publicdomainvectors.org/de/tag/Kamera>

Train:

<http://clipart-library.com/clipart/8ixn5bx4T.htm>

PCF8575 datasheet

<https://www.ti.com/lit/gpn/PCF8575>

MCP 23016 datasheet (alternative to PCF8575, not used here)

<https://ww1.microchip.com/downloads/en/DeviceDoc/20090C.pdf>

RS485 Bus transceiver

<https://datasheets.maximintegrated.com/en/ds/MAX1487-MAX491.pdf>

Arduino Nano Every

<https://docs.arduino.cc/hardware/nano-every>

