Z21 LAN Protocol Specification

Legal notices, exclusion of liability

Modelleisenbahn GmbH expressly states that it shall under no circumstances be legally liable for the content in this document or for any additional information specified in this document.

The legal responsibility exclusively lies with the user of the data provided or with the publisher of the additional information in question.

The company Modelleisenbahn GmbH (of Plainbachstrasse 4, A-5101 Bergheim, Austria) expressly accepts no liability for any and all damages caused by the use or by the non-use of the information provided.

Modelleisenbahn GmbH, Plainbachstrasse 4, A-5101 Bergheim, Austria, accepts no responsibility for the up-to-dateness, correctness, completeness or quality of the information provided. No liability claims relating to damages of a material, immaterial or conceptual nature due to the use or non-use of the information shall be accepted.

Modelleisenbahn GmbH, Plainbachstrasse 4, A-5101 Bergheim, Austria, reserves the right to modify, supplement or delete the information provided without prior notice.

All brand names and trademarks mentioned in the document and where applicable protected by third parties, are subject without restriction to the provisions of the applicable trademark law and the ownership rights of the respective registered owners.

Should parts or individual formulations of the liability disclaimer not, no longer or not completely comply with the applicable legal position, the remaining parts of the disclaimer remain unaffected in their content and validity.

Publishing info

Publisher: Modelleisenbahn GmbH, Plainbachstrasse 4, A-5101 Bergheim, Austria

Revision history

Date	Document version	Change
06.02.2013	1.00	Description of LAN interface for Z21 FW Version 1.10, 1.11 and SmartRail FW Version 1.12
20.03.2013	1.01	**Z21 FW Version 1.20** LAN_SET_BROADCASTFLAGS: new Flags LAN_GET_HWINFO: new command LAN_SET_TURNOUTMODE: MM format LocoNet: Gateway functionality **SmartRail FW Version 1.13** LAN_GET_HWINFO: new command
29.10.2013	1.02	**Z21 FW Version 1.22:** Reading and writing Decoder CVs POM Read and Accessory Decoder: new commands **LocoNet Dispatch and Track Occupancy Detector** *LAN_LOCONET_DISPATCH_ADDR*: new Reply LAN_SET_BROADCASTFLAGS: new Flags LAN_LOCONET_DETECTOR: new message
12.02.2014	1.03	**Z21 FW Version 1.23** Correction of long vehicle address in Chapter 4 Driving LAN_X_MM_WRITE_BYTE LAN_LOCONET_DETECTOR: Extension for LISSY
25.03.2014	1.04	**Z21 FW Version 1.24** LAN_SET_BROADCASTFLAGS: Flag 0x00010000 Chapter 5 Switching: Explanation Turnout addressing LAN_X_GET_TURNOUT_INFO: Queue-Bit Extension LAN_X_DCC_WRITE_REGISTER
21.01.2015	1.05	**Z21 FW Version 1.25 und 1.26** Chapter 4 Driving: Explanations about speed steps and format LAN_X_DCC_READ_REGISTER LAN_X_DCC_WRITE_REGISTER LAN_LOCONET_Z21_TX Binary State Control Instruction
05.04.2016	1.06	**Z21 FW Version 1.28** Chapter 2 System Status Versions: z21start LAN_GET_HW_INFO LAN_GET_CODE
19.04.2017	1.07	**Z21 FW Version 1.29 und 1.30** Chapter 8 RailCom Chapter 10 CAN
15.01.2018	1.08	Chapter 9 LocoNet : Lissy Examples
23.05.2019	1.09 en	First english edition Chapter 4 Driving: added speed step coding infos Chapter 7 R-BUS: added 10808 and 10819 Chapter 9.3.1 fixed Binary State Control Instruction
28.01.2021	1.10 en	**Z21 FW Version 1.40** Chapter 4 LAN_GET_HWINFO: HW-Types Chapter 5 Switching: Extended Accessories DCCext Chapter 11 zLink
11.08.2021	1.11 en	**Z21 FW Version 1.41** Chapter 10 CAN: Booster
28.02.2022	1.12 en	**Z21 FW Version 1.42** Chapter 2.18 SystemState: cseRCN213, Capabilities Chapter 4: DCC functions ≥ F29, binary states Chapter 6: fixed typo POM Read Byte „111001MM“ (0xE4) Chapter 10.2 and 11.2: Booster Management
08.07.2022	1.13 en	**Z21 FW Version 1.43** Chapter 4 Driving: Motorola-Bit in LAN_X_LOCO_INFO Chapter 4 Driving: added commands for purging and E-STOP Chapter 12: Fastclock

Table of contents

1 Basics

1.1 Communication

Communication with the command station Z21 is done via UDP by using port 21105 or 21106. Control applications on the client (PC, App, ...) should primarily use port 21105.

Communication is always asynchronous, i.e. broadcast messages can occur between a request and the corresponding response.

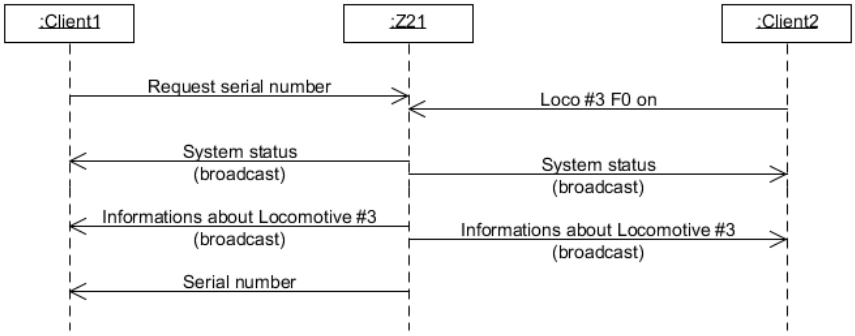

Figure 1 Example Sequence: Communication

Each client is expected to communicate with the Z21 once per minute, otherwise it will be removed from the list of active participants. If possible, a client should log off from the command station with the command LAN_LOGOFF.

1.2 Z21 Dataset

1.2.1 Structure

A Z21 data record, i.e. a request or response, is structured in the following way:

DataLen (2 Byte)	Header (2 Byte)	Data (n Bytes)

- **DataLen** (little endian):
  Total length over the entire data set including DataLen, Header and Data, i.e. DataLen = 2+2+n.
- **Header** (little endian):
  Describes the Command and the Protocol's group.
- **Data**:
  Structure and number depend on the command. For a detailed description, see the respective command.

Unless otherwise specified, the byte order is little-endian, i.e. first the low byte, then the high byte.

1.2.2 X-BUS Protocol tunneling

Requests and responses *based* on the X-BUS protocol are transmitted with the Z21-LAN-Header **0x40 (*LAN_X*_xxx)**. This only refers to the protocol, because these commands have nothing to do with the physical X-BUS of the Z21, but are exclusively addressed to the LAN clients or the Z21.

The actual X-BUS command is located inside the Data field within the Z21 data record. The last byte is a checksum and is calculated as XOR via the X-BUS command. Example:

DataLen		**Header**		**Data**			
				X-Header	**DB0**	**DB1**	**XOR-Byte**
0x08	0x00	**0x40**	0x00	**h**	**x**	**y**	h XOR x XOR y

1.2.3 LocoNet tunneling

From Z21 FW Version 1.20.

With the Z21-LAN headers 0xA0 and 0xA1 (LAN_LOCONET_Z21_RX, LAN_LOCONET_Z21_TX), messages received or sent by the Z21 on the LocoNet bus are forwarded to the LAN client. The LAN client can subscribe to these LocoNet messages by using the ***2.16*** LAN_SET_BROADCASTFLAGS.

The LAN client can write messages to the LocoNet bus via the Z21-LAN header **0xA2 (LAN_LOCONET_FROM_LAN**).

This way the Z21 can be used as an Ethernet/LocoNet gateway, where the Z21 is also the LocoNet master managing the refresh slots and generating the DCC packets.

The actual LocoNet message is located inside the Data field within the Z21 data record.

Example Loconet message OPC_MOVE_SLOTS <0><0> („DISPATCH_GET“) is sent by the Z21:

DataLen		**Header**		**Data**			
				OPC	**ARG1**	**ARG2**	**CKSUM**
0x08	0x00	**0xA0**	0x00	**0xBA**	**0x00**	**0x00**	**0x45**

More information about the LocoNet Gateway can be found in section ***9*** LocoNet.

1.3 Combining datasets in one UDP packet

In the payload data of a UDP packet, several independent Z21 data sets can also be sent together to one recipient. Each recipient must be able to interpret these combined Z21 dataset packets.

Example

Following combined Z21 datasets in one UDP packet...

UDP Paket				
IP Header	**UDP Header**	**UDP Payload**		
		Z21 Dataset 1	**Z21 Dataset 2**	**Z21 Dataset 3**
		LAN_X_GET_TOURNOUT_INFO #4	LAN_X_GET_TOURNOUT_INFO #5	LAN_RMBUS_GETDATA #0

... is equivalent to these three UDP packets:

UDP Paket 1		
IP Header	**UDP Header**	**UDP Payload**
		Z21 dataset
		LAN_X_GET_TOURNOUT_INFO #4

UDP Paket 2		
IP Header	**UDP Header**	**UDP Payload**
		Z21 dataset
		LAN_X_GET_TOURNOUT_INFO #5

UDP Paket 3		
IP Header	**UDP Header**	**UDP Payload**
		Z21 dataset
		LAN_RMBUS_GETDATA #0

The UDP packet must fit into an Ethernet MTU, i.e. considering IPv4 header and UDP header there is a maximum of 1500-20-8 = 1472 bytes of payload data.

2 System, Status, Versions

2.1 LAN_GET_SERIAL_NUMBER

Reading the serial number of the Z21.

Request to Z21:

DataLen		Header		Data
0x04	0x00	**0x10**	0x00	-

Reply from Z21:

DataLen		Header		Data
0x08	0x00	**0x10**	0x00	32 Bits Serial number (little endian)

2.2 LAN_LOGOFF

Logging off the client from the Z21.

Request to Z21:

DataLen		Header		Data
0x04	0x00	**0x30**	0x00	-

Reply from Z21:
none

Use the same port number when logging out as when logging in.

Note: the login is implicitly done with the first command of the client (e.g. LAN_SYSTEM_STATE_GETDATA, ...).

2.3 LAN_X_GET_VERSION

The X-Bus version of the Z21 can be read out with the following command.

Request to Z21:

DataLen		Header		Data		
				X-Header	**DB0**	**XOR-Byte**
0x07	0x00	**0x40**	0x00	**0x21**	**0x21**	0x00

Reply from Z21:

DataLen		Header		Data				
				X-Header	**DB0**	**DB1**	**DB2**	**XOR-Byte**
0x09	0x00	**0x40**	0x00	0x63	0x21	**XBUS_VER**	**CMDST_ID**	0x60

XBUS_VER X-Bus protocol version (0x30 = V3.0, 0x36 = V3.6, 0x40 = V4.0, …)
CMDST_ID Command station ID (0x12 = Z21 device family)

2.4 LAN_X_GET_STATUS

This command can be used to request the Z21 status.

Request to Z21:

DataLen		Header		Data		
				X-Header	**DB0**	**XOR-Byte**
0x07	0x00	**0x40**	0x00	**0x21**	**0x24**	0x05

Reply from Z21:
see *2.12* LAN_X_STATUS_CHANGED

This command station status is identical to the CentralState, which is delivered in the system status, see *2.18* LAN_SYSTEMSTATE_DATACHANGED.

2.5 LAN_X_SET_TRACK_POWER_OFF

This command switches off the track voltage.

Request to Z21:

DataLen		Header		Data		
				X-Header	**DB0**	**XOR-Byte**
0x07	0x00	**0x40**	0x00	**0x21**	**0x80**	0xa1

Reply from Z21:
see *2.7* LAN_X_BC_TRACK_POWER_OFF

2.6 LAN_X_SET_TRACK_POWER_ON

This command switches on the track voltage, or terminates either the emergency stop or the programming mode.

Request to Z21:

DataLen		Header		Data		
				X-Header	**DB0**	**XOR-Byte**
0x07	0x00	**0x40**	0x00	**0x21**	**0x81**	0xa0

Reply from Z21:
see *2.8* LAN_X_BC_TRACK_POWER_ON

2.7 LAN_X_BC_TRACK_POWER_OFF

The following packet is sent from the Z21 to the registered clients when
- a client has sent command *2.5* LAN_X_SET_TRACK_POWER_OFF.
- or the track voltage has been switched off by some input device (multiMaus).
- and the relevant client has activated the corresponding broadcast, see ***2.16*** LAN_SET_BROADCASTFLAGS, Flag 0x00000001

Z21 to Client:

DataLen		**Header**		**Data**		
				X-Header	**DB0**	**XOR-Byte**
0x07	0x00	**0x40**	0x00	**0x61**	**0x00**	0x61

2.8 LAN_X_BC_TRACK_POWER_ON

The following packet is sent from the Z21 to the registered clients when
- a client has sent command *2.6* LAN_X_SET_TRACK_POWER_ON.
- or the track voltage has been switched on by some input device (multiMaus).
- and the relevant client has activated the corresponding broadcast, see ***2.16*** LAN_SET_BROADCASTFLAGS, Flag 0x00000001

Z21 to Client:

DataLen		**Header**		**Data**		
				X-Header	**DB0**	**XOR-Byte**
0x07	0x00	**0x40**	0x00	**0x61**	**0x01**	0x60

2.9 LAN_X_BC_PROGRAMMING_MODE

The following packet is sent from the Z21 to the registered clients if the Z21 has been put into CV programming mode by ***6.1*** LAN_X_CV_READ or ***6.2*** LAN_X_CV_WRITE and the respective client has activated the corresponding broadcast, see ***2.16*** LAN_SET_BROADCASTFLAGS, Flag 0x00000001

Z21 to Client:

DataLen		**Header**		**Data**		
				X-Header	**DB0**	**XOR-Byte**
0x07	0x00	**0x40**	0x00	**0x61**	**0x02**	0x63

2.10 LAN_X_BC_TRACK_SHORT_CIRCUIT

The following packet is sent from the Z21 to the registered clients if a short circuit has occurred and the relevant client has activated the corresponding broadcast, see ***2.16*** LAN_SET_BROADCASTFLAGS, Flag 0x00000001

Z21 to Client:

DataLen		**Header**		**Data**		
				X-Header	**DB0**	**XOR-Byte**
0x07	0x00	**0x40**	0x00	**0x61**	**0x08**	0x69

2.11 LAN_X_UNKNOWN_COMMAND

The following packet is sent from the Z21 to the client in response to an invalid request.

Z21 to Client:

DataLen		**Header**		**Data**		
				X-Header	**DB0**	**XOR-Byte**
0x07	0x00	**0x40**	0x00	**0x61**	**0x82**	E3

2.12 LAN_X_STATUS_CHANGED

The following packet is sent from the Z21 to the client if the client explicitly sets the status to *2.4* LAN_X_GET_STATUS*.*

Z21 to Client:

DataLen		**Header**		**Data**			
				X-Header	**DB0**	**DB1**	**XOR-Byte**
0x08	0x00	**0x40**	0x00	**0x62**	**0x22**	**Status**	XOR-Byte

DB1 … command station status

Bitmask for command station status:

```
#define csEmergencyStop          0x01   // The emergency stop is switched on
#define csTrackVoltageOff        0x02   // The track voltage is switched off.
#define csShortCircuit           0x04   // Short-circuit
#define csProgrammingModeActive  0x20   // The programming mode is active
```

This command station status is identical to the SystemState.CentralState, see *2.18* LAN_SYSTEMSTATE_DATACHANGED*.*

2.13 LAN_X_SET_STOP

With this command the emergency stop is activated, i.e. the locomotives are stopped but the track voltage remains switched on.

Request to Z21:

DataLen		**Header**		**Data**	
				X-Header	**XOR-Byte**
0x06	0x00	**0x40**	0x00	**0x80**	0x80

Reply from Z21:
see *2.14* LAN_X_BC_STOPPED

2.14 LAN_X_BC_STOPPED

The following packet is sent from the Z21 to the registered clients when

- a client has sent command *2.13* LAN_X_SET_STOP.
- or the emergency stop was triggered by some input device (multiMaus).
- and the relevant client has activated the corresponding broadcast, see ***2.16*** LAN_SET_BROADCASTFLAGS, Flag 0x00000001

Z21 to Client:

DataLen		**Header**		**Data**		
				X-Header	**DB0**	**XOR-Byte**
0x07	0x00	**0x40**	0x00	**0x81**	0x00	0x81

2.15 LAN_X_GET_FIRMWARE_VERSION

The firmware version of the Z21 can be read with this command.

Request to Z21:

DataLen		**Header**		**Data**		
				X-Header	**DB0**	**XOR-Byte**
0x07	0x00	**0x40**	0x00	**0xF1**	0x0A	0xFB

Reply from Z21:

DataLen		**Header**		**Data**				
				X-Header	**DB0**	**DB1**	**DB2**	**XOR-Byte**
0x09	0x00	**0x40**	0x00	0xF3	0x0A	**V_MSB**	**V_LSB**	XOR-Byte

DB1 … MSB of the Firmware version
DB2 … LSB of the Firmware version

The version is specified in BCD format.
Example:
`0x09 0x00 0x40 0x00 0xf3 0x0a` **`0x01 0x23`** `0xdb` … means: „Firmware Version **1.23**“

2.16 LAN_SET_BROADCASTFLAGS

Set the broadcast flags in the Z21. These flags are set per client (i.e. per IP + port number) and must be set again the next time you log on.

Request to Z21:

DataLen		Header		Data
0x08	0x00	**0x50**	0x00	32 Bits Broadcast-Flags (little endian)

Broadcast flags are an OR-combination of the following values:

0x00000001 Broadcasts and info messages concerning driving and switching are delivered to the registered clients automatically.
The following messages are concerned:
2.7 LAN_X_BC_TRACK_POWER_OFF
2.8 LAN_X_BC_TRACK_POWER_ON
2.9 LAN_X_BC_PROGRAMMING_MODE
2.10 LAN_X_BC_TRACK_SHORT_CIRCUIT
2.14 LAN_X_BC_STOPPED
4.4 LAN_X_LOCO_INFO (loco address must be subscribed too)
5.3 LAN_X_TURNOUT_INFO

0x00000002 Changes of the feedback devices on the R-Bus are sent automatically.
Z21 Broadcast messages see ***7.1*** LAN_RMBUS_DATACHANGED

0x00000004 Changes of RailCom data of subscribed locomotives are sent automatically.
Z21 Broadcast messages see **8.1** LAN_RAILCOM_DATACHANGED

0x00000100 Changes of the Z21 system status are sent automatically.
Z21 Broadcast messages see ***2.18*** LAN_SYSTEMSTATE_DATACHANGED

From Z21 FW Version 1.20:

0x00010000 Extends flag 0x00000001; client now gets LAN_X_LOCO_INFO LAN_X_LOCO_INFO without having to subscribe to the corresponding locomotive addresses, i.e. for all controlled locomotives!
Due to the high network traffic, this flag may only be used by adequate PC railroad automation software and **is NOT intended for mobile hand controllers** under any circumstances.
From FW V1.20 bis V1.23: LAN_X_LOCO_INFO is sent for **all** locomotives.
From **FW V1.24**: LAN_X_LOCO_INFO is sent for **all modified** locomotives.

0x01000000 Forwarding messages from LocoNet bus to LAN client without locos and switches.

0x02000000 Forwarding locomotive-specific LocoNet messages to LAN Client:
OPC_LOCO_SPD, OPC_LOCO_DIRF, OPC_LOCO_SND, OPC_LOCO_F912, OPC_EXP_CMD

0x04000000 Forwarding switch-specific LocoNet messages to LAN client:
OPC_SW_REQ, OPC_SW_REP, OPC_SW_ACK, OPC_SW_STATE

See also chapter ***9*** LocoNet.

From Z21 FW Version 1.22:

0x08000000 Sending status changes of LocoNet track occupancy detectors to the LAN client.
See **9.5** LAN_LOCONET_DETECTOR

From Z21 FW Version 1.29:

0x00040000 Sending changes of RailCom data to the LAN Client.
Client gets LAN_RAILCOM_DATACHANGED without having to subscribe to the corresponding locomotive addresses, i.e. for all controlled locomotives! Due to the high network traffic, this flag may only be used by adequate PC railroad automation software and is NOT intended for mobile hand controllers under any circumstances.
Z21 Broadcast messages see **8.1** LAN_RAILCOM_DATACHANGED

From Z21 FW Version 1.30:
0x00080000 Sending status changes of CAN-Bus track occupancy detectors to the LAN client.
See **10.1** LAN_CAN_DETECTOR

From Z21 FW Version 1.41:
0x00020000 Forward CAN-Bus booster status messages to LAN Client.
See **10.2.3**LAN_CAN_BOOSTER_SYSTEMSTATE_CHGD

From Z21 FW Version 1.43:
0x00000010 Send fast clock time messages to LAN client.
See **12.2** LAN_FAST_CLOCK_DATA

Reply from Z21:
none

When preparing the settings for the broadcast flags, always consider the effects on the network load. This applies in particular to the broadcast flags 0x00010000, 0x00040000, 0x02000000 and 0x04000000! The IP packets may be deleted by the router in case of overload and UDP does not offer any detection mechanisms for this! For example, before using flag 0x00000100 (system status) it is worth considering whether 0x00000001 with the corresponding LAN_X_BC_xxx broadcast messages would be a more suitable alternative. Not every application needs to be regularly informed in detail about the latest voltage, current and temperature values of the Z21.

2.17 LAN_GET_BROADCASTFLAGS

Reading the broadcast flags in the Z21.

Request to Z21:

DataLen		**Header**		**Data**
0x04	0x00	**0x51**	0x00	-

Reply from Z21:

DataLen		**Header**		**Data**
0x08	0x00	**0x51**	0x00	Broadcast-Flags 32 Bit (little endian)

Broadcast-Flags see above.

2.18 LAN_SYSTEMSTATE_DATACHANGED

Reports a change in the system status from the Z21 to the client.

This message is asynchronously reported to the client by the Z21 when the client

- activated the corresponding broadcast, see ***2.16*** LAN_SET_BROADCASTFLAGS, Flag 0x00000100.
- explicitly requested the system status, see ***2.19*** LAN_SYSTEMSTATE_GETDATA.

Z21 to Client:

DataLen		**Header**		**Data**
0x14	0x00	**0x84**	0x00	**SystemState** (16 Bytes)

SystemState is structured as follows (the 16-bit values are little endian):

Byte Offset	**Typ**	**Name**		
0	INT16	MainCurrent	mA	Current on the main track
2	INT16	ProgCurrent	mA	Current on programming track
4	INT16	FilteredMainCurrent	mA	smoothed current on the main track
6	INT16	Temperature	°C	command station internal temperature
8	UINT16	SupplyVoltage	mV	supply voltage
10	UINT16	VCCVoltage	mV	internal voltage, identical to track voltage
12	UINT8	CentralState	bitmask	see below
13	UINT8	CentralStateEx	bitmask	see below
14	UINT8	reserved		
15	UINT8	Capabilities	bitmask	see below, from Z21 FW Version 1.42

Bitmask for CentralState:

```
#define csEmergencyStop          0x01   // The emergency stop is switched on
#define csTrackVoltageOff        0x02   // The track voltage is switched off
#define csShortCircuit           0x04   // Short-circuit
#define csProgrammingModeActive  0x20   // The programming mode is active
```

Bitmask for CentralStateEx:

```
#define cseHighTemperature       0x01   // temperature too high
#define csePowerLost             0x02   // Input voltage too low
#define cseShortCircuitExternal  0x04   // S.C. at the external booster output
#define cseShortCircuitInternal  0x08   // S.C. at the main track or programming track
```

From Z21 FW Version 1.42:

```
#define cseRCN213                0x20   // turnout addresses according to RCN-213
```

From Z21 FW Version 1.42:

Bitmask for Capabilities:

```
#define capDCC                   0x01   // capable of DCC
#define capMM                    0x02   // capable of MM
//#define capReserved            0x04   // reserved for future development
#define capRailCom               0x08   // RailCom is activated
#define capLocoCmds              0x10   // accepts LAN commands for locomotive decoders
#define capAccessoryCmds         0x20   // accepts LAN commands for accessory decoders
#define capDetectorCmds          0x40   // accepts LAN commands for detectors
#define capNeedsUnlockCode       0x80   // device needs activate code (z21start)
```

SystemState.Capabilities provides an overview of the device's range of features.
If SystemState.Capabilities == 0, then it can be assumed that the device has an older firmware version.
SystemState.Capabilities should not be evaluated when using older firmware versions!

2.19 LAN_SYSTEMSTATE_GETDATA

Request the current system status.

Request to Z21:

DataLen		Header		Data
0x04	0x00	**0x85**	0x00	-

Reply from Z21:
see above ***2.18*** LAN_SYSTEMSTATE_DATACHANGED

2.20 LAN_GET_HWINFO

From Z21 FW Version 1.20 and SmartRail FW Version V1.13.

Read the hardware type and the firmware version of the Z21.

Request to Z21:

DataLen		Header		Data
0x04	0x00	**0x1A**	0x00	-

Reply from Z21:

DataLen		Header		Data	
0x0C	0x00	**0x1A**	0x00	HwType 32 Bit (little endian)	FW Version 32 Bit (little endian)

HwType:

```
#define D_HWT_Z21_OLD      0x00000200   // „black Z21” (hardware variant from 2012)
#define D_HWT_Z21_NEW      0x00000201   // „black Z21”(hardware variant from 2013)
#define D_HWT_SMARTRAIL    0x00000202   // SmartRail (from 2012)
#define D_HWT_z21_SMALL    0x00000203   // „white z21” starter set variant (from 2013)
#define D_HWT_z21_START    0x00000204   // „z21 start” starter set variant (from 2016)

#define D_HWT_SINGLE_BOOSTER     0x00000205   // 10806 „Z21 Single Booster” (zLink)
#define D_HWT_DUAL_BOOSTER       0x00000206   // 10807 „Z21 Dual Booster” (zLink)

#define D_HWT_Z21_XL       0x00000211   // 10870 „Z21 XL Series”  (from 2020)
#define D_HWT_XL_BOOSTER   0x00000212   // 10869 „Z21 XL Booster” (from 2021, zLink)

#define D_HWT_Z21_SWITCH_DECODER 0x00000301   // 10836 „Z21 SwitchDecoder” (zLink)
#define D_HWT_Z21_SIGNAL_DECODER 0x00000302   // 10836 „Z21 SignalDecoder” (zLink)
```

The **FW version** is specified in BCD format.

Example:
0x0C 0x00 0x1A 0x00 0x00 0x02 0x00 0x00 0x20 0x01 0x00 0x00
means: „Hardware Type **0x200,** Firmware Version **1.20**“

To read out the version of an older firmware, use the alternative command
2.15 LAN_X_GET_FIRMWARE_VERSION. Apply following rules for older firmware versions:

- V1.10 ... Z21 (hardware variant from 2012)
- V1.11 ... Z21 (hardware variant from 2012)
- V1.12 ... SmartRail (from 2012)

2.21 LAN_GET_CODE

Read the software feature scope of the Z21 (and z21 or z21start of course).

This command is of particular interest for the hardware variant "z21 start", in order to be able to check whether driving and switching via LAN is blocked or permitted.

Request to Z21:

DataLen		Header		Data
0x04	0x00	**0x18**	0x00	-

Reply from Z21:

DataLen		Header		Data
0x05	0x00	**0x18**	0x00	Code (8 Bit)

Code:

```
#define Z21_NO_LOCK         0x00  // all features permitted
#define z21_START_LOCKED    0x01  // „z21 start”: driving and switching is blocked
#define z21_START_UNLOCKED 0x02  // „z21 start”: driving and switching is permitted
```

3 Settings

The following settings described here are stored in the Z21 persistently.
These settings can be reset by the user to the factory settings by keeping the STOP button on the Z21 pressed until the LEDs flash violet.

3.1 LAN_GET_LOCOMODE

Read the output format for a given locomotive address.

In the Z21, the output format (DCC, MM) is persistently stored for each locomotive address. A maximum of 256 different locomotive addresses can be stored. Each address >= 256 is DCC automatically.

Request to Z21:

DataLen		**Header**		**Data**
0x06	0x00	**0x60**	0x00	16 bits Loco-Address (**big endian**)

Reply from Z21:

DataLen		**Header**		**Data**	
0x07	0x00	**0x60**	0x00	16 bits Loco-Address (**big endian**)	Mode 8 bit

Loco Address 2 Bytes, **big endian**, i.e. first comes high byte, followed by low byte.

Mode 0 ... DCC Format
1 ... MM Format

3.2 LAN_SET_LOCOMODE

Set the output format for a given locomotive address. The format is stored in the Z21persistently.

Request to Z21:

DataLen		**Header**		**Data**	
0x07	0x00	**0x61**	0x00	Loco address 16 Bit (**big endian**)	Modus 8 bit

Reply from Z21:
none

Meaning of the values: see above.

Note: each locomotive address >= 256 is and remains "Format DCC" automatically.

Note: the speed steps (14, 28, 128) are also stored in the command station persistently. This automatically happens with the loco driving command, see ***4.2*** LAN_X_SET_LOCO_DRIVE.

3.3 LAN_GET_TURNOUTMODE

Read the settings for a given accessory decoder address ("Accessory Decoder" RP-9.2.1).

In the Z21, the output format (DCC, MM) is persistently stored for each accessory decoder address. A maximum of 256 different accessory decoder addresses can be stored. Each address >= 256 automatically is DCC.

Request to Z21:

DataLen		**Header**		**Data**
0x06	0x00	**0x70**	0x00	16 bits Accessory Decoder Address (**big endian**)

Reply from Z21:

DataLen		**Header**		**Data**	
0x07	0x00	**0x70**	0x00	16 bits Accessory Decoder Address (**big endian**)	Mode 8 bit

Accessory Decoder Address 2 Bytes, **big endian,** i.e. first comes high byte, followed by low byte.

Mode 0 ... DCC Format
1 ... MM Format

At the LAN interface and in the Z21, the accessory decoder addresses are addressed from 0, but in the visualization in the apps or on the multiMaus from 1. This is only a decision of the visualization.
Example: multiMaus switch address #3, corresponds to address 2 on the LAN and in Z21.

3.4 LAN_SET_TURNOUTMODE

Set the output format for a given accessory decoder address. The format is stored in the Z21 persistently.

Request to Z21:

DataLen		**Header**		**Data**	
0x07	0x00	**0x71**	0x00	16 bits Accessory Decoder Address (**big endian**)	Mode 8 bit

Reply from Z21:
none

Meaning of the values: see above.

MM accessory decoders are supported by Z21 firmware version 1.20 and higher.
MM accessory decoders are not supported by SmartRail.

Note: Each accessory decoder >= 256 is and remains DCC automatically.

4 Driving

This chapter describes the messages that are required for driving with locomotive decoders.

A client can subscribe to locomotive infos with *4.1* LAN_X_GET_LOCO_INFO in order to be automatically informed about changes to this locomotive address caused also by other clients or handsets. Furthermore the corresponding broadcast must also be activated for the client, see ***2.16*** LAN_SET_BROADCASTFLAGS, Flag 0x00000001.

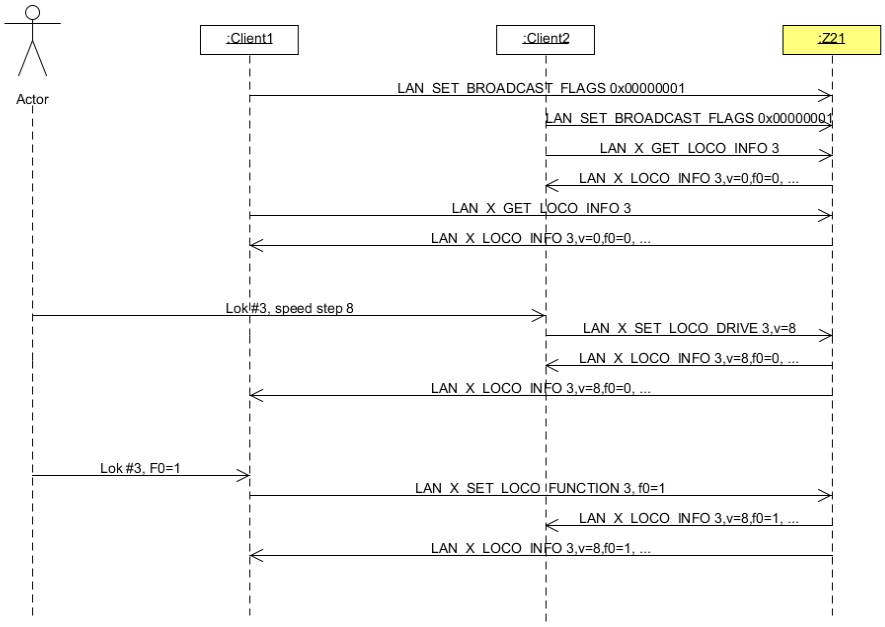

Figure 2 Example sequence: locomotive control

In order to keep network traffic within reasonable limits, a maximum of 16 locomotive addresses per client can be subscribed to (FIFO). You could also poll the locos, but always consider the network load: the IP packets may be deleted by the router in case of overload and UDP does not offer any detection mechanisms.

4.1 LAN_X_GET_LOCO_INFO

The following command can be used to poll the status of a locomotive. At the same time, the client also "subscribes" to the locomotive information for this locomotive address (only in combination with LAN_SET_BROADCASTFLAGS, Flag 0x00000001).

Request to Z21:

DataLen		**Header**		**Data**				
				X-Header	**DB0**	**DB1**	**DB2**	**XOR-Byte**
0x09	0x00	**0x40**	0x00	**0xE3**	**0xF0**	**Adr_MSB**	**Adr_LSB**	XOR-Byte

Note: loco address = (**Adr_MSB** & 0x3F) << 8 + **Adr_LSB**
For locomotive addresses ≥ 128, the two highest bits in DB1 must be set to 1:
DB1 = (**0xC0** | **Adr_MSB**). For locomotive addresses < 128, these two highest bits have no meaning.

Reply from Z21:
see *4.4* LAN_X_LOCO_INFO

4.2 LAN_X_SET_LOCO_DRIVE

Change the speed and direction of a locomotive.

Request to Z21:

DataLen		**Header**		**Data**					
				X-Header	**DB0**	**DB1**	**DB2**	**DB3**	**XOR-Byte**
0x0A	0x00	**0x40**	0x00	**0xE4**	**0x1S**	**Adr_MSB**	**Adr_LSB**	**RVVVVVVV**	XOR-Byte

Note: loco address = (**Adr_MSB** & 0x3F) << 8 + **Adr_LSB**
For locomotive addresses ≥ 128, the two highest bits in DB1 must be set to 1:
DB1 = (**0xC0** | **Adr_MSB**). For locomotive addresses < 128, these two highest bits have no meaning.

0x1**S** Number of speed steps, depending on the rail format set
S=0: DCC 14 speed steps, or MMI with 14 speed steps and F0
S=2: DCC 28 speed steps, or MMII with 14 real speed steps and F0-F4
S=3: DCC 128 speed steps (aka “126 speed steps” when not counting the stops), or MMII with 28 real speed steps (using light-trit) and F0-F4

RVVVVVVV R ... Direction: 1=forward
V ... Speed: depending on the speed steps S. Coding see below.
If the format MM is configured for the locomotive, the conversion of the given DCC speed stage into the real MM speed stage takes place automatically in the Z21.

The coding of the speed is similar to NMRA S 9.2 and S 9.2.1.
“**Stop**” means “normal stop” or “step 0”. “**E-Stop**” means “immediate emergency stop”.

Coding speed for “DCC 14”:

R000 VVVV	**Speed**	**R000 VVVV**	**Speed**	**R000 VVVV**	**Speed**	**R000 VVVV**	**Speed**
R000 0000	**Stop**	R000 0100	Step 3	R000 1000	Step 7	R000 1100	Step 11
R000 0001	**E-Stop**	R000 0101	Step 4	R000 1001	Step 8	R000 1101	Step 12
R000 0010	Step 1	R000 0110	Step 5	R000 1010	Step 9	R000 1110	Step 13
R000 0011	Step 2	R000 0111	Step 6	R000 1011	Step 10	R000 1111	Step 14 **max**

Coding speed for “DCC 28” (like “DCC 14”, but with additional intermediate speed step in the fifth bit **V_5**):

R00V_5 VVVV	**Speed**	**R00V_5 VVVV**	**Speed**	**R00V_5 VVVV**	**Speed**	**R00V_5 VVVV**	**Speed**
R000 0000	**Stop**	R000 0100	Step 5	R000 1000	Step 13	R000 1100	Step 21
R001 0000	**Stop**[1]	R001 0100	Step 6	R001 1000	Step 14	R001 1100	Step 22
R000 0001	**E-Stop**	R000 0101	Step 7	R000 1001	Step 15	R000 1101	Step 23
R001 0001	**E-Stop**[1]	R001 0101	Step 8	R001 1001	Step 16	R001 1101	Step 24
R000 0010	Step 1	R000 0110	Step 9	R000 1010	Step 17	R000 1110	Step 25
R001 0010	Step 2	R001 0110	Step 10	R001 1010	Step 18	R001 1110	Step 26
R000 0011	Step 3	R000 0111	Step 11	R000 1011	Step 19	R000 1111	Step 27
R001 0011	Step 4	R001 0111	Step 12	R001 1011	Step 20	R001 1111	Step 28 **max**

Coding speed for “DCC 128”:

RVVV VVVV	**Speed**
R000 0000	**Stop**
R000 0001	**E-Stop**
R000 0010	Step 1
R000 0011	Step 2
R000 0100	Step 3
R000 0101	Step 4
…	…
R111 1110	Step 125
R111 1111	Step 126 **max**

[1] Usage not recommended

Reply from Z21:
No standard reply, *4.4* LAN_X_LOCO_INFO to subscribed clients.

Note: the number of speed steps (14/28/128) is automatically stored for the given loco address in the command station persistently.

4.3 Functions for locomotive decoder

Function commands from F0 up to and including F12 are sent periodically (priority dependent) on the main track, just like the speed and direction.

Function commands F13 and above, on the other hand, are sent three times on the main track after a change and then, regarding the available bandwidth on the track and according with RCN-212, are no longer sent until the next change of the function state.

4.3.1 LAN_X_SET_LOCO_FUNCTION

Change a function of a locomotive.

Request to Z21:

DataLen		**Header**		**Data**					
				X-Header	**DB0**	**DB1**	**DB2**	**DB3**	**XOR-Byte**
0x0A	0x00	**0x40**	0x00	**0xE4**	**0xF8**	**Adr_MSB**	**Adr_LSB**	**TTNN NNNN**	XOR-Byte

Note: loco address = (**Adr_MSB** & 0x3F) << 8 + **Adr_LSB**
For locomotive addresses ≥ 128, the two highest bits in DB1 must be set to 1:
DB1 = (**0xC0** | **Adr_MSB**). For locomotive addresses < 128, these two highest bits have no meaning.

TT switch type: 00=off, 01=on, 10=toggle,11=not allowed
NNNNNN Function index, 0x00=F0 (light), 0x01=F1 etc.

With Motorola MMI only F0 can be switched. With MMII, F0 to F4 can be used.
With DCC, F0 to F28 can be switched here. **From Z21 FW version 1.42** the extended range from **F0 to F31** can be used here.

Reply from Z21:
No standard reply, *4.4* LAN_X_LOCO_INFO to subscribed clients.

4.3.2 LAN_X_SET_LOCO_FUNCTION_GROUP

With the following command, a whole function group of a locomotive decoder can be switched. Thus, up to 8 functions can be switched with a single command. **From Z21 FW version 1.42**, DCC functions can be switched up to F31, and with some restrictions even up to F68.

The client should constantly monitor the status of all functions of the controlled locomotive to avoid accidentally switching off a function when sending this command, which may have been switched on before by another LAN client or handheld controller. For this reason, this command is more suitable for PC railroad automation software, because it should keep track of all vehicles anyway.

Request to Z21:

DataLen		**Header**		**Data**					
				X-Header	**DB0**	**DB1**	**DB2**	**DB3**	**XOR-Byte**
0x0A	0x00	**0x40**	0x00	**0xE4**	**Group**	**Adr_MSB**	**Adr_LSB**	**Functions**	XOR-Byte

Note: loco address = (**Adr_MSB** & 0x3F) << 8 + **Adr_LSB**
For locomotive addresses ≥ 128, the two highest bits in DB1 must be set to 1:
DB1 = (**0xC0** | **Adr_MSB**). For locomotive addresses < 128, these two highest bits have no meaning.

Groups and **functions** are structured as follows:

Number	**Group**	**Functions**								**Remarks**
	HEX	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**	
1	**0x20**	0	0	0	F0	F4	F3	F2	F1	(A)
2	**0x21**	0	0	0	0	F8	F7	F6	F5	
3	**0x22**	0	0	0	0	F12	F11	F10	F9	
4	**0x23**	F20	F19	F18	F17	F16	F15	F14	F13	(B)
5	**0x28**	F28	F27	F26	F25	F24	F23	F22	F21	(B)
6	**0x29**	F36	F35	F34	F33	F32	**F31**	**F30**	**F29**	**(C)** (D) (E)
7	**0x2A**	F44	F43	F42	F41	F40	F39	F38	F37	(D) (E)
8	**0x2B**	F52	F51	F50	F49	F48	F47	F46	F45	(D) (E)
9	**0x50**	F60	F59	F58	F57	F56	F55	F54	F53	(D) (E)
10	**0x51**	F68	F67	F66	F65	F64	F63	F62	F61	(D) (E)

Remarks:
(A) With Motorola MMI only F0 can be used, with MMII F0 up to F4 can be used.
(B) DCC F13 to F28 with this command only **from Z21 FW V1.24** and higher.
(C) DCC F29 to F31 **from Z21 FW V1.42**, including feedback to the LAN clients, see also below.
(D) DCC F32 to F68 **from Z21 FW V1.42**, however, there is no feedback to the LAN clients. The DCC function commands are only sent on the track.
(E) We cannot guarantee that the DCC function commands from F29 and higher will actually be understood by all decoders! Currently (2022) only very few DCC decoder types understand the function commands from F29 (F29 to F31 were tested successfully with "Loksound 5" decoder). Nowadays, some manufacturers also offer sound functions on F29, F30 or F31, but they often do not work with DCC in practice, because their multi-protocol decoders do not yet understand the corresponding new DCC commands.

Reply from Z21:
No standard reply, for function **F0 to F31** the feedback *4.4* LAN_X_LOCO_INFO is sent to subscribed clients.

4.3.3 LAN_X_SET_LOCO_BINARY_STATE

From Z21 FW Version 1.42, a DCC "Binary State" command can be sent to a locomotive decoder with the following command.

Request to Z21:

DataLen		**Header**		**Data**						
				X-Header	**DB0**	**DB1**	**DB2**	**DB3**	**DB4**	**XOR-Byte**
0x0A	0x00	**0x40**	0x00	**0xE5**	**0x5F**	**AH**	**AL**	**FLLL LLLL**	**HHHH HHHH**	XOR-Byte

Note: loco address = (**Adr_MSB** & 0x3F) << 8 + **Adr_LSB**
For locomotive addresses ≥ 128, the two highest bits in DB1 must be set to 1:
DB1 = (**0xC0** | **Adr_MSB**). For locomotive addresses < 128, these two highest bits have no meaning.

F The most significant bit **F** determines whether the binary state is on or off.
LLLLLLL The low-order **seven** (!) bits of the binary state address.
HHHHHHHH The high eight bits of the binary state address.

Note: The following applies: the 15-bit binary state address = (**HHHHHHHH** << **7**) + (**LLLLLLL** & 0x7F)

The binary states address range from 29 to 32767 is permitted.
Only binary state addresses ≥ 29 may be used for general switching functions.
The binary state addresses from 1 to 28 are reserved for special applications.
Binary state address 0 is reserved as broadcast.

Binary state addresses < 128 (i.e., if **HHHHHHHH** == 0) are **automatically** issued on the track as DCC "binary state control command **short form**" according to RCN-212, from ≥ 128 as DCC "binary state control command **long form**".

DCC binary state control commands are sent three times on the main track, and according to RCN-212, thereafter no more repeated regularly.

There is no response to the caller and no notification to other clients.

Reply from Z21:
None.

4.4 LAN_X_LOCO_INFO

This message is sent from the Z21 to the clients in response to the command *4.1* LAN_X_GET_LOCO_INFO. However, it is also unsolicitedly sent to an associated client if

- the locomotive status has been changed by one of the (other) clients or handset controls
- and the associated client has activated the corresponding broadcast, see ***2.16*** LAN_SET_BROADCASTFLAGS, Flag 0x00000001
- and the associated client has subscribed to the locomotive address with *4.1* LAN_X_GET_LOCO_INFO.

Z21 to Client:

DataLen		**Header**		**Data**									
				X-Header	**DB0**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**XOR-Byte**
7 + ***n***	0x00	**0x40**	0x00	0xEF	**Locomotive Information**								XOR-Byte

The actual packet length *n* may vary depending on the data actually sent, with ***7 ≤ n ≤ 14.***
From Z21 FW version 1.42 DataLen is ≥ 15 (**n ≥ 8**) for also transferring the status of F29, F30 and F31!

The data for locomotive information is structured as follows:

Position	**Data**	**Meaning**
DB0	**Adr_MSB**	The two highest bits in Adr_MSB must be ignored.
DB1	**Adr_LSB**	Loco address = (**Adr_MSB** & 0x3F) << 8 + **Adr_LSB**
DB2	0000**BKKK**	**M**=1 … **From Z21 FW version 1.43** identifies loco with MM output format **B**=1 ... the locomotive is controlled by another X-BUS handset controller ("busy") **KKK** ... Speed steps information: 0=14, 2=28, 4=128 0: DCC 14 speed steps, or MMI with 14 speed steps and F0 2: DCC 28 speed steps, or MMII with 14 real speed stages and F0-F4 **4**: DCC 128 speed steps, or MMII with 28 real speed stages (light-trit) and F0-F4
DB3	**RVVVVVVV**	**R** ... Directon: 1=forward **V** ... Speed. Coding also depends on the speed steps information KKK. See also above section *4.2* LAN_X_SET_LOCO_DRIVE. If the format MM is configured for the locomotive, then the conversion of the real MM speed step into the presented DCC speed step has already been done in the Z21.
DB4	0**DSLFGHJ**	**D** ... double traction: 1=Loco included in a double traction **S** ... Smartsearch **L** ... F0 (Licht) **F** ... F4 **G** ... F3 **H** ... F2 **J** ... F1
DB5	F5-F12	Function F5 is bit0 (LSB)
DB6	F13-F20	Function F13 is bit0 (LSB)
DB7	F21-F28	Function F21 is bit0 (LSB)
DB8	F29-F31	**From Z21 FW version 1.42** and if DataLen ≥ 15; Function F29 is bit0 (LSB)
DB***n***		optional, for future extensions

4.5 LAN_X_SET_LOCO_E_STOP

From Z21 FW version 1.43, a locomotive can be stopped with the following command. In the case of a DCC locomotive, the speed step "**E-STOP**" ("emergency stop" according to RCN-212) is then sent in the DCC speed command onto the track, i.e., the decoder should stop the engine as quickly as possible. In the case of an MM locomotive, the speed step 0 ("Stop") is sent onto the track.

Request to Z21:

DataLen		**Header**		**Data**			
				X-Header	**DB0**	**DB2**	**XOR-Byte**
0x08	0x00	**0x40**	0x00	**0x92**	**Adr_MSB**	**Adr_LSB**	XOR-Byte

Note: loco address = (**Adr_MSB** & 0x3F) << 8 + **Adr_LSB**
For locomotive addresses ≥ 128, the two highest bits in DB1 must be set to 1:
DB1 = (**0xC0** | **Adr_MSB**). For locomotive addresses < 128, these two highest bits have no meaning.

Reply from Z21:
No standard reply, *4.4* LAN_X_LOCO_INFO to subscribed clients.

4.6 LAN_X_PURGE_LOCO

From Z21 FW version 1.43, a locomotive can be removed from the Z21 with the following command. This also cancels the sending of the loco commands for this locomotive on the track. Sending will start again as soon as a new drive or function command is sent to the same locomotive address.

In this way, it is possible, for example, for a PC railroad automation software to influence the number of locomotives in the system and thus also the data throughput on the track.

Request to Z21:

DataLen		**Header**		**Data**				
				X-Header	**DB0**	**DB1**	**DB2**	**XOR-Byte**
0x09	0x00	**0x40**	0x00	**0xE3**	**0x44**	**Adr_MSB**	**Adr_LSB**	XOR-Byte

Note: loco address = (**Adr_MSB** & 0x3F) << 8 + **Adr_LSB**
For locomotive addresses ≥ 128, the two highest bits in DB1 must be set to 1:
DB1 = (**0xC0** | **Adr_MSB**). For locomotive addresses < 128, these two highest bits have no meaning.

There is no response to the caller and no notification to other clients.

Reply from Z21:
None.

5 Switching

This chapter deals with messages which are required for switching accessory decoders ("Accessory Decoder" according RP-9.2.1, e.g. decoder for turnouts, ...).

The visualization of the turnout number on the user interface is differently solved in some DCC systems and can significantly differ from the real DCC accessory decoder address plus port actually used in the track signal. According to DCC, there are four ports with two outputs each per accessory decoder address. One turnout can be connected per port. Usually one of the following options is used to visualize the turnout number:

1. Numbering from 1 with DCC address at 1 starting with 4 ports each (ESU, Uhlenbrock, ...)
   Switch #1: DCC-Addr=1 Port=0; Switch #5: DCC-Addr=2 Port=0; Switch #6: DCC-Addr=2 Port=1

2. Numbering from 1 with DCC address at 0 starting with 4 ports each (Roco)
   Switch #1: DCC-Addr=0 Port=0; Switch #5: DCC-Addr=1 Port=0; Switch #6: DCC-Addr=1 Port=1

3. Virtual switch number with freely configurable DCC address and port (Twin Center)

4. Displaying real DCC-address and port number (Zimo)

None of these visualization options can be described as “wrong” due to lack of specification in RP-9.2.1, where the visualization to the user is not mentioned at all. For the user, however, this can mean in consequence getting used to the fact that one and the same turnout at an ESU control panel is controlled under number 1, while it is switched on the Roco multiMaus and Z21 under number 5 ("shift by 4").

In order to be able to implement the visualization of your choice in your application, it helps to know how the Z21 converts the input parameters for the switching commands (**FAdr_MSB**, **FAdr_LSB**, **A**, **P**, see below) into the corresponding DCC accessory command:

DCC basic accessory decoder packet format: {preamble} 0 10AAAAAA 0 1aaaCDDd 0 EEEEEEEE 1

UINT16 *FAdr* = (**FAdr_MSB** << 8) + **FAdr_LSB;**
UINT16 *Dcc_Addr* = *FAdr* >> 2;

aaaAAAAAA = (~*Dcc_Addr* & 0x1C0) | (*Dcc_Addr* & 0x003F); // DCC Address
C = **A**; // Activate or deactivate output
DD = *FAdr* & 0x03; // Port
d = **P**; // Switch to the left or to the right

Example:
FAdr=0 equals DCC-Addr=0 Port=0;
FAdr=3 equals DCC-Addr=0 Port=3;
FAdr=4 equals DCC-Addr=1 Port=0; etc.

On the other hand, for MM Format note: FAdr starts with 0, i.e. FAdr=0: MM-Addr=1; FAdr=1: MM-Addr=2; ...

A client can subscribe to accessory info in order to be automatically notified of changes to accessory decoders caused by other clients or handsets. For this purpose, the corresponding broadcast must be activated for the client, see ***2.16*** *LAN_SET_BROADCASTFLAGS,* Flag 0x00000001.

The actual position of the turnout depends on the cabling and possibly also on the configuration in the client's application. The command station cannot know anything about this, and that is why the following description deliberately omits the terms "*straight*" and "*branching*". Instead we will speak about “output 1” and “output 2”.

5.1 LAN_X_GET_TURNOUT_INFO

The following command can be used to poll the status of a turnout (or any accessory function).

Request to Z21:

DataLen		**Header**		**Data**			
				X-Header	**DB0**	**DB1**	**XOR-Byte**
0x08	0x00	**0x40**	0x00	**0x43**	**FAdr_MSB**	**FAdr_LSB**	XOR-Byte

Note: Function address = (**FAdr_MSB** << 8) + **FAdr_LSB**

Reply from Z21:
see *5.3* LAN_X_TURNOUT_INFO

5.2 LAN_X_SET_TURNOUT

A turnout (or any accessory function) can be switched with the following command.

Request to Z21:

DataLen		**Header**		**Data**				
				X-Header	**DB0**	**DB1**	**DB2**	**XOR-Byte**
0x09	0x00	**0x40**	0x00	**0x53**	**FAdr_MSB**	**FAdr_LSB**	10**Q**0**A**00**P**	XOR-Byte

Note: Function address = (**FAdr_MSB** << 8) + **FAdr_LSB**

1000**A**00**P** **A**=0 ... Deactivate turnout output
A=1 ... Activate turnout output
P=0 ... Select output 1 of the turnout
P=1 ... Select output 2 of the turnout
Q=0 … Execute command immediately
Q=1 … **From Z21 FW V1.24**: Insert turnout command into the queue of Z21 and deliver it as soon as possible to the track.

Reply from Z21:
No standard answer, *5.3* LAN_X_TURNOUT_INFO to subscribed clients.

From Z21 FW V1.24 the Q flag ("Queue") was introduced.

5.2.1 LAN_X_SET_TURNOUT with Q=0

With **Q=0** the Z21 behaves compatible to the previous versions: the turnout switching command is immediately sent on the track by being mixed into the running loco driving commands. **The Activate (A=1) is output until the LAN client sends the corresponding Deactivate. Only one switching command may be active at the same time**. This behavior corresponds, for example, to pressing and releasing the multiMaus key.

Please note that with Q=0 the correct sequence of the switching commands (i.e. Activate followed by Deactivate) must be observed strictly. Otherwise, undefined end positions may occur depending on the turnout decoder used.

The LAN client is responsible for the correct serialization and the timing of the switching duration!

Wrong:
Activate turnout #5/A2 (4,0x89); Activate turnout #6/A2 (5,0x89);
Activate turnout #3/A1 (2,0x88); Deactivate turnout #3/A1 (2,0x80);
Deactivate turnout #5/A2 (4,0x81); Deactivate turnout #6/A2 (5,0x81);

Correct:
Activate turnout #5/A2 (4,0x89); wait 100ms; deactivate turnout #5/A2 (4,0x81); wait 50ms;
Activate turnout #6/A2 (5,0x89); wait 100ms; deactivate turnout #6/A2 (5,0x81); wait 50ms;
Activate turnout #3/A1 (2,0x88); wait 100ms; deactivate turnout #3/A1 (2,0x80); wait 50ms;

Example:
Activate turnout #7 / A2 (6,0x89); wait 150ms; deactivate turnout #7 / A2 (6,0x81)

```
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  LOCO address=3 FG1 (0-4) F=Loooo
DCC  preamble=16  LOCO address=3 FG2 (5-8) F=o7oo
DCC  preamble=16  ACESSORY raw data AA=1 DD=5 C=1 , "Roco_lenz f=7 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  ACESSORY raw data AA=1 DD=5 C=1 , "Roco_lenz f=7 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 FG1 (0-4) F=Loooo
DCC  preamble=16  ACESSORY raw data AA=1 DD=5 C=1 , "Roco_lenz f=7 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 FG2 (5-8) F=o7oo
DCC  preamble=16  ACESSORY raw data AA=1 DD=5 C=1 , "Roco_lenz f=7 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  ACESSORY raw data AA=1 DD=5 C=1 , "Roco_lenz f=7 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 FG1 (0-4) F=Loooo
DCC  preamble=16  ACESSORY raw data AA=1 DD=5 C=1 , "Roco_lenz f=7 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 FG2 (5-8) F=o7oo
DCC  preamble=16  ACESSORY raw data AA=1 DD=5 C=1 , "Roco_lenz f=7 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  ACESSORY raw data AA=1 DD=5 C=1 , "Roco_lenz f=7 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 FG1 (0-4) F=Loooo
DCC  preamble=16  ACESSORY raw data AA=1 DD=5 C=1 , "Roco_lenz f=7 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 FG2 (5-8) F=o7oo
DCC  preamble=16  ACESSORY raw data AA=1 DD=5 C=1 , "Roco_lenz f=7 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  ACESSORY raw data AA=1 DD=5 C=1 , "Roco_lenz f=7 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 FG1 (0-4) F=Loooo
DCC  preamble=16  ACESSORY raw data AA=1 DD=5 C=1 , "Roco_lenz f=7 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 FG2 (5-8) F=o7oo
DCC  preamble=16  ACESSORY raw data AA=1 DD=5 C=1 , "Roco_lenz f=7 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  ACESSORY raw data AA=1 DD=5 C=1 , "Roco_lenz f=7 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 FG1 (0-4) F=Loooo
DCC  preamble=16  ACESSORY raw data AA=1 DD=5 C=0 , "Roco_lenz f=7 out=A  INACTIVE"
DCC  preamble=16  LOCO address=3 FG2 (5-8) F=o7oo
DCC  preamble=16  ACESSORY raw data AA=1 DD=5 C=0 , "Roco_lenz f=7 out=A  INACTIVE"
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  ACESSORY raw data AA=1 DD=5 C=0 , "Roco_lenz f=7 out=A  INACTIVE"
DCC  preamble=16  LOCO address=3 FG1 (0-4) F=Loooo
DCC  preamble=16  ACESSORY raw data AA=1 DD=5 C=0 , "Roco_lenz f=7 out=A  INACTIVE"
DCC  preamble=16  LOCO address=3 FG2 (5-8) F=o7oo
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  LOCO address=3 FG1 (0-4) F=Loooo
```

Figure 3 DCC Sniff on track with Q=0

5.2.2 LAN_X_SET_TURNOUT with Q=1

If **Q=1**, the following behavior occurs: in the Z21 the switching command is first put into an internal queue (FIFO). When generating the track signal, this queue is constantly checked whether a switching command is available for output. This switching command is then taken out of the queue and is written four times onto the track. This liberates the LAN client from the obligation of strict serialization, i.e. the switching commands may be sent mixed to the Z21 with Q=1 (very useful routes!). The LAN client only needs to take care of the Deactivate timing. Depending on the DCC decoder, the Deactivate may even be omitted. With MM you should not do without Deactivate, because e.g. the k83 and some older turnout decoders do not have an automatic shut-off.

Example:
Activate turnout #25 / A2 (24, 0xA9); Activate turnout #5 / A2 (4, 0xA9);
Wait 150ms;
Deactivate turnout #25 / A2 (24, 0xA1)

```
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  ACESSORY raw data AA=6 DD=1 C=1 , "Roco_lenz f=25 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  ACESSORY raw data AA=6 DD=1 C=1 , "Roco_lenz f=25 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  ACESSORY raw data AA=6 DD=1 C=1 , "Roco_lenz f=25 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  ACESSORY raw data AA=6 DD=1 C=1 , "Roco_lenz f=25 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  ACESSORY raw data AA=1 DD=1 C=1 , "Roco_lenz f=5 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  ACESSORY raw data AA=1 DD=1 C=1 , "Roco_lenz f=5 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  ACESSORY raw data AA=1 DD=1 C=1 , "Roco_lenz f=5 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  ACESSORY raw data AA=1 DD=1 C=1 , "Roco_lenz f=5 out=A  ACTIVE"
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  ACESSORY raw data AA=6 DD=1 C=0 , "Roco_lenz f=25 out=A  INACTIVE"
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  ACESSORY raw data AA=6 DD=1 C=0 , "Roco_lenz f=25 out=A  INACTIVE"
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  ACESSORY raw data AA=6 DD=1 C=0 , "Roco_lenz f=25 out=A  INACTIVE"
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  ACESSORY raw data AA=6 DD=1 C=0 , "Roco_lenz f=25 out=A  INACTIVE"
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
DCC  preamble=16  LOCO address=3 ss128=0 fwd Speed=Stop
```

Figure 4 DCC Sniff on track with Q=1

Never mix switching commands with Q=0 and switching commands with Q=1 in your application.

5.3 LAN_X_TURNOUT_INFO

This message is sent from the Z21 to the clients in response to the command *5.1* LAN_X_GET_TURNOUT_INFO. However, it is also sent to an associated client unsolicitedly if

- the function status has been changed by one of the (other) clients or a handset controller
- and the associated client has activated the corresponding broadcast, see ***2.16*** LAN_SET_BROADCASTFLAGS, Flag 0x00000001

Z21 to Client:

DataLen		**Header**		**Data**				
				X-Header	**DB0**	**DB1**	**DB2**	**XOR-Byte**
0x09	0x00	**0x40**	0x00	**0x43**	**FAdr_MSB**	**FAdr_LSB**	000000**ZZ**	XOR-Byte

Note: Function address = (**FAdr_MSB** << 8) + **FAdr_LSB**

000000**ZZ** **ZZ**=00 ... Turnout not switched yet
ZZ=01 ... Turnout is in position according to switching command "P=0", see *5.2* LAN_X_SET_TURNOUT
ZZ=10 ... Turnout is in position according to switching command "P=1", see *5.2* LAN_X_SET_TURNOUT
ZZ=11 ... Invalid combination

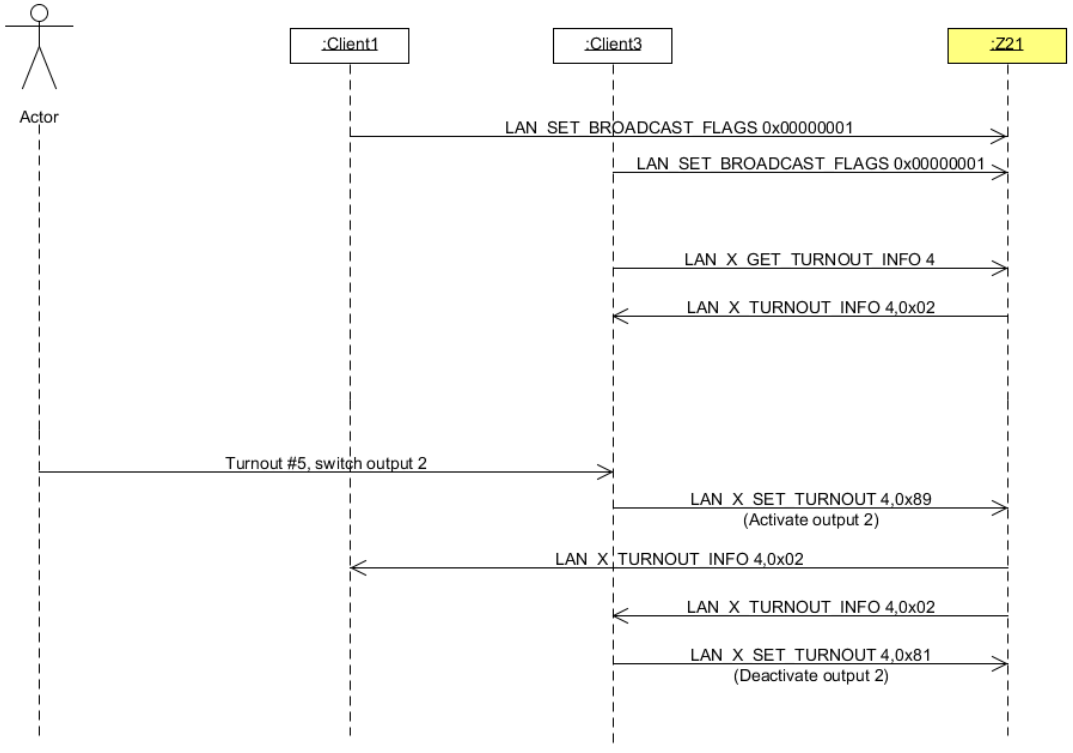

Figure 5 Example Sequence: Turnout switching

5.4 LAN_X_SET_EXT_ACCESSORY

From Z21 FW V1. 40, a DCC command in the "**extended accessory decoder package format**" (DCCext) can be sent to an **extended accessory decoder** with the following request. It allows to send even switching times for turnouts or complex signal aspects with just one single command. See also RCN-213 (Section 2.3).

Request to Z21:

DataLen		**Header**		**Data**					
				X-Header	**DB0**	**DB1**	**DB2**	**DB3**	**XOR-Byte**
0x0A	0x00	**0x40**	0x00	**0x54**	**Adr_MSB**	**Adr_LSB**	DDDDDDDD	0x00	XOR-Byte

Note: **RawAddress** = (**Adr_MSB** << 8) + **Adr_LSB**

RawAddress The RawAddress for the first extended accessory decoder is 4 according to RCN-213. This address is usually displayed as "Address 1" in user interfaces. The address calculation is strictly compliant with RCN-213, i.e. there is no longer any different address calculation compared with other compliant systems.

DDDDDDDD 256 different states can be transmitted via bits 0 to 7 in DB2.
The content is transferred to the decoder on the track in the ***extended accessory decoder package format*** according to RCN-213.

Note:
Ter **10836 Z21 switch DECODER** interprets DDDDDDDD as "switch decoder with reception of switching time" as **RZZZZZZZ**. The following applies:

- **ZZZZZZZ** defines the power-on time with a resolution of **100 ms.**
  - A value of 0 means that the output is switched off.
  - a value of 127 means that the output is switched on permanently, i.e. until the next command to this address.
- Bit 7 **R** is used to select the output within the output pair:
  - R=1 means "green" (straight).
  - R=0 means "red" (branched).

The **10837 Z21 signaldecoder** interprets DDDDDDDD as one of 256 theoretically possible signal aspects. The actual value range depends to a large extent on the signal type set in the signal decoder. Common values are, for example:

- 0 ... Stop
- 4 ... Clear with speed limit max 40 km/h
- 16 ... Clear
- 65 (0x41) ... shunting allowed
- 66 (0x42) ... turn all lights off (e.g. for distant signals)
- 69 (0x45) ... substitution (permission to pass a defect signal)

The suitable value for the desired signal aspect for a given signal can be found for the Z21 signal DECODER under https://www.z21.eu/en/products/z21-signal-decoder/signaltypen.

Reply from Z21:
No standard answer, or *5.6 LAN_X_EXT_ACCESSORY_INFO* to subscribed clients.

Example:
0x0A 0x00 0x40 0x00 0x54 0x00 0x04 0x05 0x00 0x55
meaning: "send to decoder with RawAddress=4 (this address is displayed as address 1 in user dialogs!) a value of DDDDDDDD=5."
If the receiver is a 10836 Z21 switch DECODER, then the output 1 "red" (clamp 1A) will be switched on and switched off again after 5*100ms automatically.

With this command, it is also possible to send the "emergency stop command for extended accessory decoders" according to RCN-213 (Section 2.4). This corresponds to the value 0 ("Stop") for the RawAddress=2047:
0x0A 0x00 0x40 0x00 0x54 0x07 0xFF 0x00 0x00 0xAC

5.5 LAN_X_GET_EXT_ACCESSORY_INFO

From Z21 FW V1. 40, the following request can be used to poll the last command transferred to an **extended accessory decoder**.

Request to Z21:

DataLen		**Header**		**Data**				
				X-Header	**DB0**	**DB1**	**DB2**	**XOR-Byte**
0x09	0x00	**0x40**	0x00	**0x44**	**Adr_MSB**	**Adr_LSB**	0x00	XOR-Byte

Note: **RawAddress** = (**Adr_MSB** << 8) + **Adr_LSB**

RawAddress The address of the accessory decoder according to RCN-213.
See section *5.4* LAN_X_SET_EXT_ACCESSORY

DB2 Reserved for future extensions, should remain initialized with 0 until further notice.

Reply from Z21:
see *5.6*LAN_X_EXT_ACCESSORY_INFO

5.6 LAN_X_EXT_ACCESSORY_INFO

This message is sent from the Z21 to the clients in response to command *5.5 LAN_X_GET_EXT_ACCESSORY_INFO*.
However, it is also sent to an associated client unsolicitedly if

- the accessory status has been changed by one of the (other) clients or a handset controller
- and the associated client has activated the corresponding broadcast, see ***2.16*** LAN_SET_BROADCASTFLAGS, Flag 0x00000001

Z21 to Client:

DataLen		**Header**		**Data**					
				X-Header	**DB0**	**DB1**	**DB2**	**DB3**	**XOR-Byte**
0x0A	0x00	**0x40**	0x00	**0x44**	**Adr_MSB**	**Adr_LSB**	**DDDDDDDD**	**Status**	XOR-Byte

Note: **RawAddress** = (**Adr_MSB** << 8) + **Adr_LSB**

RawAddress The address of the accessory decoder according to RCN-213.
See section *5.4*LAN_X_SET_EXT_ACCESSORY

DDDDDDDD Up to 256 possible states encoded in ***extended accessory decoder package format*** according to RCN-213.
See section *5.4* LAN_X_SET_EXT_ACCESSORY.

Status 0x00 … Data Valid
0xFF … Data Unknown

6 Reading and writing Decoder CVs

This chapter deals with messages required for reading and writing decoder CVs (Configuration Variable, RP-9.2.2, RP-9.2.3).

Whether the decoder is accessed bit-wise or byte-wise depends on the settings in the Z21.

6.1 LAN_X_CV_READ

Read a CV in direct mode.

Request to Z21:

DataLen		**Header**		**Data**				
				X-Header	**DB0**	**DB1**	**DB2**	**XOR-Byte**
0x09	0x00	**0x40**	0x00	**0x23**	**0x11**	**CVAdr_MSB**	**CVAdr_LSB**	XOR-Byte

Note: CV Address = (**CVAdr_MSB** << 8) + **CVAdr_LSB**, where 0=CV1, 1=CV2, 255=CV256, etc.

Reply from Z21:
2.9 LAN_X_BC_PROGRAMMING_MODE to subscribed clients, as well as the result
6.3 LAN_X_CV_NACK_SC*, 6.4* LAN_X_CV_NACK or *6.5* LAN_X_CV_RESULT*.*

6.2 LAN_X_CV_WRITE

Write a CV in direct mode.

Request to Z21:

DataLen		**Header**		**Data**					
				X-Header	**DB0**	**DB1**	**DB2**	**DB3**	**XOR-Byte**
0x0A	0x00	**0x40**	0x00	**0x24**	**0x12**	**CVAdr_MSB**	**CVAdr_LSB**	**Value**	XOR-Byte

Note: CV-Address = (**CVAdr_MSB** << 8) + **CVAdr_LSB**, where 0=CV1, 1=CV2, 255=CV256, etc.

Reply from Z21:
2.9 LAN_X_BC_PROGRAMMING_MODE to subscribed clients, as well as the result
6.3 LAN_X_CV_NACK_SC*, 6.4* LAN_X_CV_NACK or *6.5* LAN_X_CV_RESULT*.*

6.3 LAN_X_CV_NACK_SC

If the programming failed due to a short circuit on the track, this message is automatically sent to the client that initiated the programming by *6.1* LAN_X_CV_READ or *6.2* LAN_X_CV_WRITE.

Z21 to Client:

DataLen		**Header**		**Data**		
				X-Header	**DB0**	**XOR-Byte**
0x07	0x00	**0x40**	0x00	**0x61**	**0x12**	0x73

6.4 LAN_X_CV_NACK

If the ACK is missing from the decoder, this message is automatically sent to the client that initiated the programming by *6.1* LAN_X_CV_READ or *6.2* LAN_X_CV_WRITE.
When reading with byte-wise access, the time until *LAN_X_CV_NACK* can be very long.

Z21 to Client:

DataLen		**Header**		**Data**		
				X-Header	**DB0**	**XOR-Byte**
0x07	0x00	**0x40**	0x00	**0x61**	**0x13**	0x72

6.5 LAN_X_CV_RESULT

This message is also a "positive ACK" and is automatically sent to the client that initiated the programming by *6.1* LAN_X_CV_READ or *6.2* LAN_X_CV_WRITE.
When reading with byte-wise access, the time until *LAN_X_CV_RESULT* can be very long.

Z21 to Client:

DataLen		**Header**		**Data**					
				X-Header	**DB0**	**DB1**	**DB2**	**DB3**	**XOR-Byte**
0x0A	0x00	**0x40**	0x00	**0x64**	**0x14**	**CVAdr_MSB**	**CVAdr_LSB**	**Value**	XOR-Byte

Note: CV Address = (**CVAdr_MSB** << 8) + **CVAdr_LSB**, where 0=CV1, 1=CV2, 255=CV256, etc.

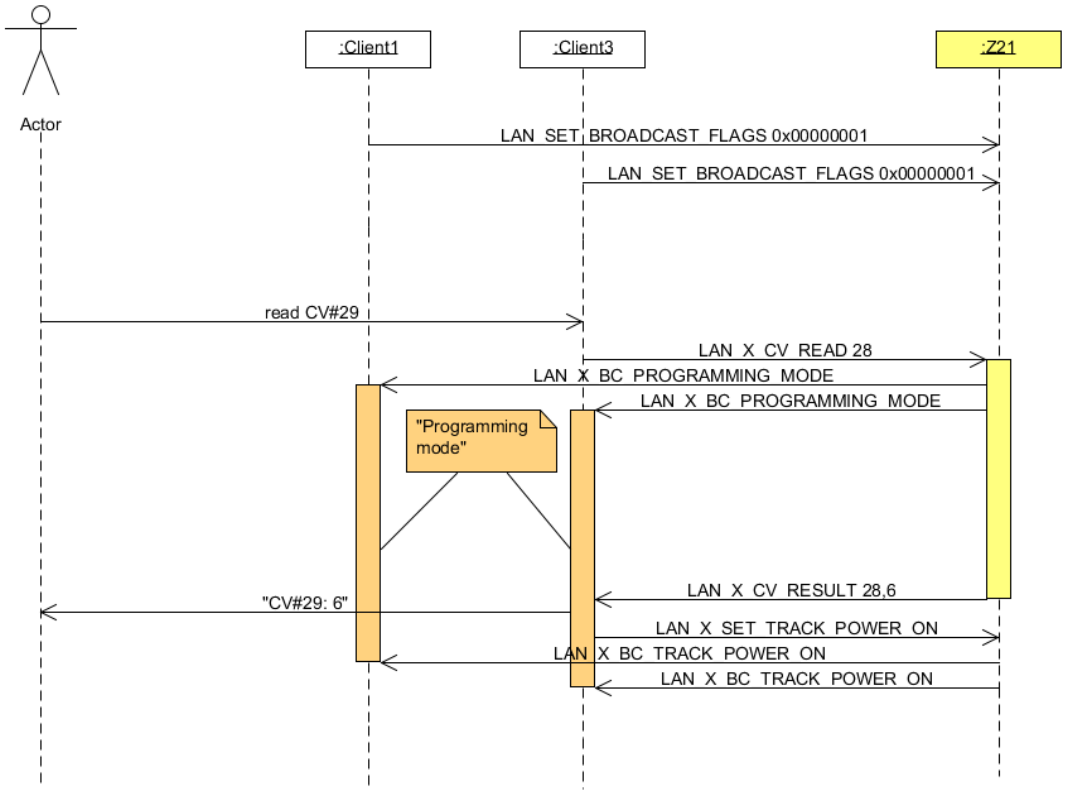

Figure 6 Example Sequence: CV Reading

6.6 LAN_X_CV_POM_WRITE_BYTE

With the following command a CV of a locomotive decoder (“Multi Function Digital Decoders” according to NMRA S-9.2.1 Section C; Configuration Variable Access Instruction - Long Form) can be written on the main track (POM "Programming on the Main"). This is done in normal operating mode, i.e. the track voltage must be already switched on and the service mode is not activated. There is no feedback.

Request to Z21:

DataLen		**Header**		**Data**							
				X-Header	**DB0**	**DB1**	**DB2**	**DB3**	**DB4**	**DB5**	**XOR-Byte**
0x0C	0x00	**0x40**	0x00	**0xE6**	**0x30**	**POM-Parameter**					XOR-Byte

The data for **POM parameters** is structured as follows:

Position	**Data**	**Meaning**
DB1	**Adr_MSB**	
DB2	**Adr_LSB**	Loco Address = (**Adr_MSB** & 0x3F) << 8 + **Adr_LSB**
DB3	**111011MM**	**Option** ... **0xEC** **MM** ... CVAdr_MSB
DB4	**CVAdr_LSB**	CV Address = (**MM** << 8) + **CVAdr_LSB** (0=CV1., 1=CV2, 255=CV256, etc.)
DB5	**Value**	New CV Value

Reply from Z21:
none

6.7 LAN_X_CV_POM_WRITE_BIT

With the following command one bit of a CV of a locomotive decoder (“Multi Function Digital Decoders” according to NMRA S-9.2.1 Section C; Configuration Variable Access Instruction - Long Form) can be written on the main track (POM). This is done in normal operating mode, i.e. the track voltage must be already switched on and the service mode is not activated. There is no feedback.

Request to Z21:

DataLen		**Header**		**Data**							
				X-Header	**DB0**	**DB1**	**DB2**	**DB3**	**DB4**	**DB5**	**XOR-Byte**
0x0C	0x00	**0x40**	0x00	**0xE6**	**0x30**	**POM-Parameter**					XOR-Byte

The data for **POM parameters** is structured as follows:

Position	**Data**	**Meaning**
DB1	**Adr_MSB**	
DB2	**Adr_LSB**	Loco Address = (**Adr_MSB** & 0x3F) << 8 + **Adr_LSB**
DB3	**111010MM**	**Option** ... **0xE8** **MM** ... CVAdr_MSB
DB4	**CVAdr_LSB**	CV Address = (**MM** << 8) + **CVAdr_LSB** (0=CV1., 1=CV2, 255=CV256, etc.)
DB5	0000**VPPP**	**PPP** ... Bit-Position in CV **V** ... New CV Value

Reply from Z21:
none

6.8 LAN_X_CV_POM_READ_BYTE

From Z21 FW Version 1.22.

With the following command a CV of a locomotive decoder (“Multi Function Digital Decoders” according to NMRA S-9.2.1 Section C; Configuration Variable Access Instruction - Long Form) can be read on the main track (POM). This is done in normal operating mode, i.e. the track voltage must be already switched on and the service mode is not activated. RailCom must be activated in the Z21. The vehicle decoder to be read must be capable of RailCom, CV28 bit 0 and 1 as well as CV29 bit 3 must be set to 1 in the locomotive decoder (Zimo).

Request to Z21:

DataLen		**Header**		**Data**							
				X-Header	**DB0**	**DB1**	**DB2**	**DB3**	**DB4**	**DB5**	**XOR-Byte**
0x0C	0x00	**0x40**	0x00	**0xE6**	**0x30**	**POM-Parameter**					XOR-Byte

The data for **POM parameters** is structured as follows:

Position	**Data**	**Meaning**
DB1	**Adr_MSB**	
DB2	**Adr_LSB**	Loco Address = (**Adr_MSB** & 0x3F) << 8 + **Adr_LSB**
DB3	**111001MM**	**Option** ... **0xE4** **MM** ... CVAdr_MSB
DB4	**CVAdr_LSB**	CV Address = (**MM** << 8) + **CVAdr_LSB** (0=CV1., 1=CV2, 255=CV256, etc.)
DB5	**0**	

Reply from Z21:
6.4 LAN_X_CV_NACK or *6.5* LAN_X_CV_RESULT.

6.9 LAN_X_CV_POM_ACCESSORY_WRITE_BYTE

From Z21 FW Version 1.22.

With the following command a CV of an accessory decoder (according to NMRA S-9.2.1 Section D, “Basic Accessory Decoder Packet address for operations mode programming”) can be written on the main track (POM). This happens in normal operating mode, i.e. the track voltage must be already switched on and the service mode is not activated. There is no feedback.

Request to Z21:

DataLen		**Header**		**Data**							
				X-Header	**DB0**	**DB1**	**DB2**	**DB3**	**DB4**	**DB5**	**XOR-Byte**
0x0C	0x00	**0x40**	0x00	**0xE6**	**0x31**	**POM-Parameter**					XOR-Byte

The data for **POM parameters** is structured as follows:

Position	**Data**	**Meaning**
DB1	**aaaaa**	Decoder_Address MSB
DB2	**AAAACDDD**	Note: **aaaaaAAAACDDD** = ((Decoder_Address & 0x1FF) << 4) | CDDD; In case **CDDD**=0000, then the CV refers to the whole decoder. In case **C**=1, then **DDD** is the number of the output to be programmed.
DB3	**111011MM**	**Option** ... **0xEC** **MM** ... CVAdr_MSB
DB4	**CVAdr_LSB**	CV Address = (**MM** << 8) + **CVAdr_LSB** (0=CV1, 1=CV2, 255=CV256, etc.)
DB5	**Value**	new CV value

Reply from Z21:
none

6.10 LAN_X_CV_POM_ ACCESSORY_WRITE_BIT

From Z21 FW Version 1.22.

With the following command a CV of an accessory decoder (according to NMRA S-9.2.1 Section D, “Basic Accessory Decoder Packet address for operations mode programming”) can be written on the main track (POM). This happens in normal operating mode, i.e. the track voltage must be already switched on and the service mode is not activated. There is no feedback.

Request to Z21:

DataLen		**Header**		**Data**							
				X-Header	**DB0**	**DB1**	**DB2**	**DB3**	**DB4**	**DB5**	**XOR-Byte**
0x0C	0x00	**0x40**	0x00	**0xE6**	**0x31**	**POM-Parameter**					XOR-Byte

The data for **POM parameters** is structured as follows:

Position	**Data**	**Meaning**
DB1	**aaaaa**	Decoder_Address MSB
DB2	**AAAACDDD**	Note: **aaaaaAAAACDDD** = ((Decoder_Address & 0x1FF) << 4) | CDDD; In case **CDDD**=0000, then the CV refers to the whole decoder. In case **C**=1, then **DDD** is the number of the output to be programmed.
DB3	**111010MM**	**Option** ... **0xE8** **MM** ... CVAdr_MSB
DB4	**CVAdr_LSB**	CV Address = (**MM** << 8) + **CVAdr_LSB** (0=CV1, 1=CV2, 255=CV256, etc.)
DB5	0000**VPPP**	**PPP** ... Bit position in CV **V** ... new CV value

Reply from Z21:
none

6.11 LAN_X_CV_POM_ ACCESSORY_READ_BYTE

From Z21 FW Version 1.22.

With the following command a CV of an accessory decoder (according to NMRA S-9.2.1 Section D, “Basic Accessory Decoder Packet address for operations mode programming”) can be read on the main track (POM). This happens in normal operating mode, i.e. the track voltage must be already switched on, the service mode is not activated. RailCom must be activated in the Z21. The accessory decoder to be read must be capable of RailCom.

Request to Z21:

DataLen		**Header**		**Data**							
				X-Header	**DB0**	**DB1**	**DB2**	**DB3**	**DB4**	**DB5**	**XOR-Byte**
0x0C	0x00	**0x40**	0x00	**0xE6**	**0x31**	**POM-Parameter**					XOR-Byte

The data for **POM parameters** is structured as follows:

Position	**Data**	**Meaning**
DB1	**aaaaa**	Decoder_Address MSB
DB2	**AAAACDDD**	Note: **aaaaaAAAACDDD** = ((Decoder_Address & 0x1FF) << 4) | CDDD; In case **CDDD**=0000, then the CV refers to the whole decoder. In case **C**=1, then **DDD** is the number of the output to be programmed.
DB3	**111001MM**	**Option** ... **0xE4** **MM** ... CVAdr_MSB
DB4	**CVAdr_LSB**	CV Address = (**MM** << 8) + **CVAdr_LSB** (0=CV1, 1=CV2, 255=CV256, etc.)
DB5	**0**	new CV value

Reply from Z21:
6.4 LAN_X_CV_NACK or *6.5* LAN_X_CV_RESULT.

6.12 LAN_X_MM_WRITE_BYTE

From Z21 FW Version 1.23.

With the following command a register of a Motorola decoder can be written on the programming track.

Request to Z21:

DataLen		**Header**		**Data**					
				X-Header	**DB0**	**DB1**	**DB2**	**DB3**	**XOR-Byte**
0x0A	0x00	**0x40**	0x00	**0x24**	**0xFF**	**0**	**RegAdr**	**Value**	XOR-Byte

Note: **RegAdr**: 0=Register1, 1=Register2, ..., 78=Register79.
Note: 0 ≤ **Value** ≤ 255, but some decoders only accept values from 0 to 80.

Reply from Z21:
2.9 LAN_X_BC_PROGRAMMING_MODE to subscribed clients, as well as the result
6.3 LAN_X_CV_NACK_SC or *6.5* LAN_X_CV_RESULT.

Note: Programming a Motorola decoder was not possible in the original Motorola format.
Therefore there exists no standardized and binding programming procedure for programming Motorola decoders. However, for the programming of Motorola decoders, the so-called "6021 programming mode" was implemented in the Z21. This allows writing values, but there is no way to read them. Hence the success of the write operation cannot be checked (except for short-circuit detection). This programming procedure works for many ESU, Zimo and Märklin decoders, but not necessarily for all MM decoders. For example, Motorola decoders with DIP switches cannot be programmed. Some decoders only accept values from 0 to 80, others accept values from 0 to 255 (see decoder description).

Since no feedback about the success of the write operation comes from the decoder during Motorola programming, the message *LAN_X_CV_RESULT* is only to be understood as "*MM programming process finished*" and **not** as "*MM programming process successful*".

Example:
`0x0A 0x00 0x40 0x00 0x24 0xFF 0x00 0x00 0x05 0xDE`
meaning: "Change the locomotive decoder address (register 1) to 5"

6.13 LAN_X_DCC_READ_REGISTER

From Z21 FW Version 1.25.

The following command can be used to read a register of a DCC decoder in register mode (S-9.2.3 Service Mode Instruction Packets for Physical Register Addressing) on the programming track.

Request to Z21:

DataLen		**Header**		**Data**			
				X-Header	**DB0**	**DB1**	**XOR-Byte**
0x08	0x00	**0x40**	0x00	**0x22**	**0x11**	**REG**	XOR-Byte

Note: **REG**: 0x01=Register1, 0x02=Register2, …, 0x08=Register8.
Note: 0 ≤ **Value** ≤ 255

Reply from Z21:
2.9 LAN_X_BC_PROGRAMMING_MODE to subscribed clients, as well as the result
6.3 LAN_X_CV_NACK_SC or *6.5* LAN_X_CV_RESULT.

Note: Programming in register mode is only required for very old DCC decoders. Direct CV is preferred.

6.14 LAN_X_DCC_WRITE_REGISTER

From Z21 FW Version 1.25.

With the following command a register of a DCC decoder in register mode (S-9.2.3 Service Mode Instruction Packets for Physical Register Addressing) can be written on the programming track.

Request to Z21:

DataLen		**Header**		**Data**				
				X-Header	**DB0**	**DB2**	**DB3**	**XOR-Byte**
0x09	0x00	**0x40**	0x00	**0x23**	**0x12**	**REG**	**Value**	XOR-Byte

Note: **REG**: 0x01=Register1, 0x02=Register2, …, 0x08=Register8.
Note: 0 ≤ **Value** ≤ 255

Reply from Z21:
2.9 LAN_X_BC_PROGRAMMING_MODE to subscribed clients, as well as the result
6.3 LAN_X_CV_NACK_SC or *6.5* LAN_X_CV_RESULT.

Note: Programming in register mode is only required for very old DCC decoders. Direct CV is preferred.

7 Feedback – R-BUS

The feedback modules (Roco 10787, 10808, 10819) on the R-BUS can be read out and configured with the following commands.

7.1 LAN_RMBUS_DATACHANGED

Report the changes on the feedback bus from the Z21 to the client.

This message is asynchronously reported to the client by the Z21 when the client

- activated the corresponding broadcast, see ***2.16*** LAN_SET_BROADCASTFLAGS, Flag 0x00000002
- or explicitly requested the feedback status, see below *7.2* LAN_RMBUS_GETDATA.

Z21 to Client:

DataLen		**Header**		**Data**	
0x0F	0x00	**0x80**	0x00	**Group index** (1 Byte)	**Feedback status** (10 Byte)

Group index: 0 ... feedback module with address from 1 to 10
1 ... feedback module with address from 11 to 20

Feedback status: 1 byte per feedback, 1 bit per input.
The order of feedback address and byte position is ascending.

Example:
Group Index = 1 and Feedback Status = 0x01 0x00 0xC5 0x00 0x00 0x00 0x00 0x00 0x00 0x00
means "feedback module #11, contact on input 1; feedback module #13, contact on input 8,7,3 and 1"

7.2 LAN_RMBUS_GETDATA

Poll the current status of the feedback modules.

Request to Z21:

DataLen		**Header**		**Data**
0x05	0x00	**0x81**	0x00	**Group index** (1 Byte)

Group index: see above

Reply from Z21:
See above *7.1* LAN_RMBUS_DATACHANGED

7.3 LAN_RMBUS_PROGRAMMODULE

Change the address of one feedback module.

Request to Z21:

DataLen		Header		Data
0x05	0x00	**0x82**	0x00	**Address** (1 Byte)

Address: New address for the feedback module to be programmed.
Supported value range: 0 and 1 ... 20.

Reply from Z21:
none

The programming-sequence is transmitted on the R-BUS until this command is sent again to the Z21 with address=0.

Only one single feedback module should be connected to the R-BUS during the programming process.

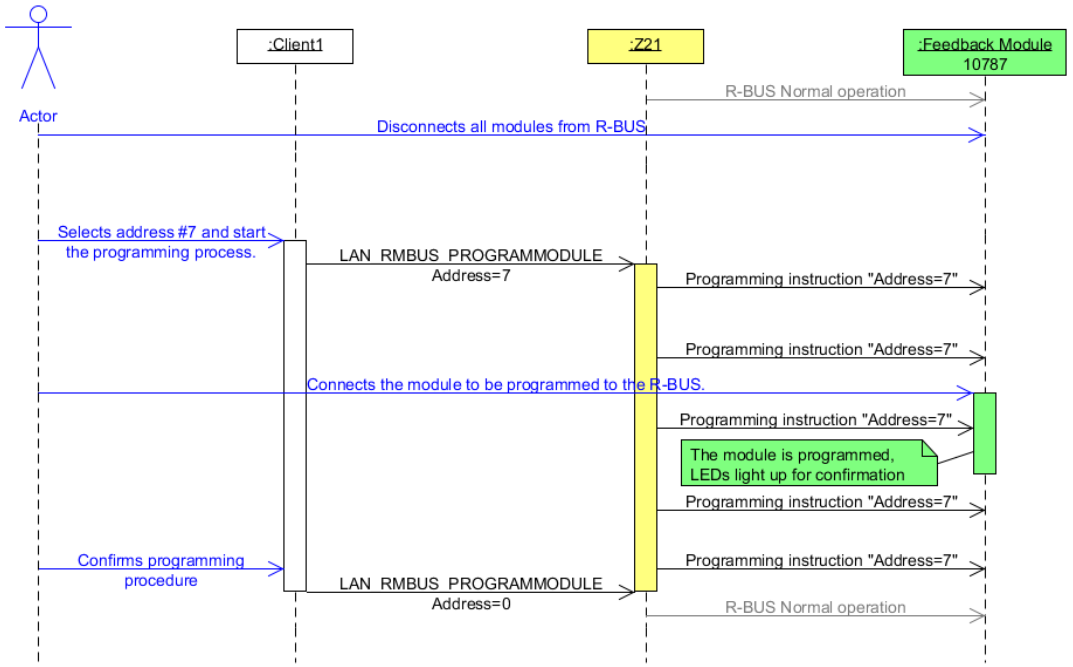

Figure 7 Example Sequence: Programming the feedback module

8 RailCom

The Z21 supports RailCom with:

- Generating the RailCom cutout in the track signal.
- Global RailCom receiver in the Z21.
- Local RailCom receivers, e.g. in the occupancy detectors 10808 or boosters 10806 and 10807. The data from RailCom channel 2 of the 10806, 10807 and 10808 can be forwarded via CAN to the Z21 and evaluated there from FW V1.29 onwards.
- Reading POM results.
  See also 6.8 LAN_X_CV_POM_READ_BYTE as of FW V1.22.
- Locomotive address recognition for occupancy detectors (CAN, LocoNet, X-BUS).
  See 9.5 *LAN_LOCONET_DETECTOR* from V1.22 and 10.1 *LAN_CAN_DETECTOR* from V1.30.
- Decoder speed (see below) from FW V1.29.
- Decoder QoS (see below) from FW V1.29.

In order to use these features, the decoder must be capable of RailCom, CV28 and CV29 must be correctly configured, and the option "RailCom" must activated in the Z21 settings.

Note: It heavily depends on the decoder firmware whether and in which form a decoder supports speed, QoS and POM!

8.1 LAN_RAILCOM_DATACHANGED

This message is sent to the clients by the Z21 from FW version 1.29 on as a response to the command 8.2 LAN_RAILCOM_GETDATA.

However, it is also sent to clients unsolicitedly, if

- the corresponding RailCom data have actually changed
- and the associated client has activated the corresponding broadcast.
  (see **2.16** LAN_SET_BROADCASTFLAGS, Flag 0x00000004) and the associated client has subscribed to the locomotive address with 4.1 LAN_X_GET_LOCO_INFO.
- or the associated client has subscribed to broadcast 0x00040000 (i.e. RailCom data of all locomotives, for PC control SW only).

Z21 to Client:

DataLen		**Header**		**Data**
0x11	0x00	**0x88**	0x00	**RailComData**

The structure **RailComData** is structured as follows (the 16-bit and 32-bit values are little endian)

Byte Offset	**Type**	**Name**	
0	UINT16	LocoAddress	Address of the detected decoder
2	UINT32	ReceiveCounter	Receive counter in Z21
6	UINT16	ErrorCounter	Receive error counter in Z21
8	UINT8	reserved	

9	UINT8	Options	Flags bitmask:
			`#define rcoSpeed1 0x01 // CH7 subindex 0`
			`#define rcoSpeed2 0x02 // CH7 subindex 1`
			`#define rcoQoS 0x04 // CH7 subindex 7`

10	UINT8	Speed	Speed 1 or 2 (if supported by decoder)
11	UINT8	QoS	Quality of Service (if supported by decoder)
12	UINT8	reserved	

The structure can be increased in the future versions; therefore it is absolutely necessary to consider DataLen in the evaluation.

8.2 LAN_RAILCOM_GETDATA

Poll RailCom data from Z21, available from FW V1.29 and higher:

Request to Z21:

DataLen		**Header**		**Data**	
0x07	0x00	**0x89**	0x00	**Type** 8 bit	**LocoAddress** 16 bit (little endian)

Type 0x01 = poll RailCom data for given locomotive address

LocoAddress **Loco address**
0= poll RailCom data of next loco (circular buffer)

Reply from Z21:
See above *8.2* LAN_RAILCOM_DATACHANGED

9 LocoNet

From Z21 FW Version 1.20.

As mentioned in the introduction, the Z21 can be used as **an Ethernet/LocoNet gateway**, where the Z21 is also the LocoNet master refreshing the slots and generating the DCC packets.

The LAN client can subscribe to the corresponding LocoNet messages using ***2.16*** LAN_SET_BROADCASTFLAGS in order to receive also the messages from LocoNet.

Messages received by the Z21 from the LocoNet bus are forwarded to the LAN client with the LAN header ***LAN_LOCONET_Z21_RX***.

Messages sent by the Z21 onto the LocoNet bus are also forwarded to the LAN client using the LAN header ***LAN_LOCONET_Z21_TX***.

With the Z21-LAN command ***LAN_LOCONET_FROM_LAN*** the LAN client itself can write messages onto the LocoNet bus. If there are other LAN clients with LocoNet subscriptions at the same time, they will also be notified with a message ***LAN_LOCONET_FROM_LAN***. Only the actual sender will not be notified.

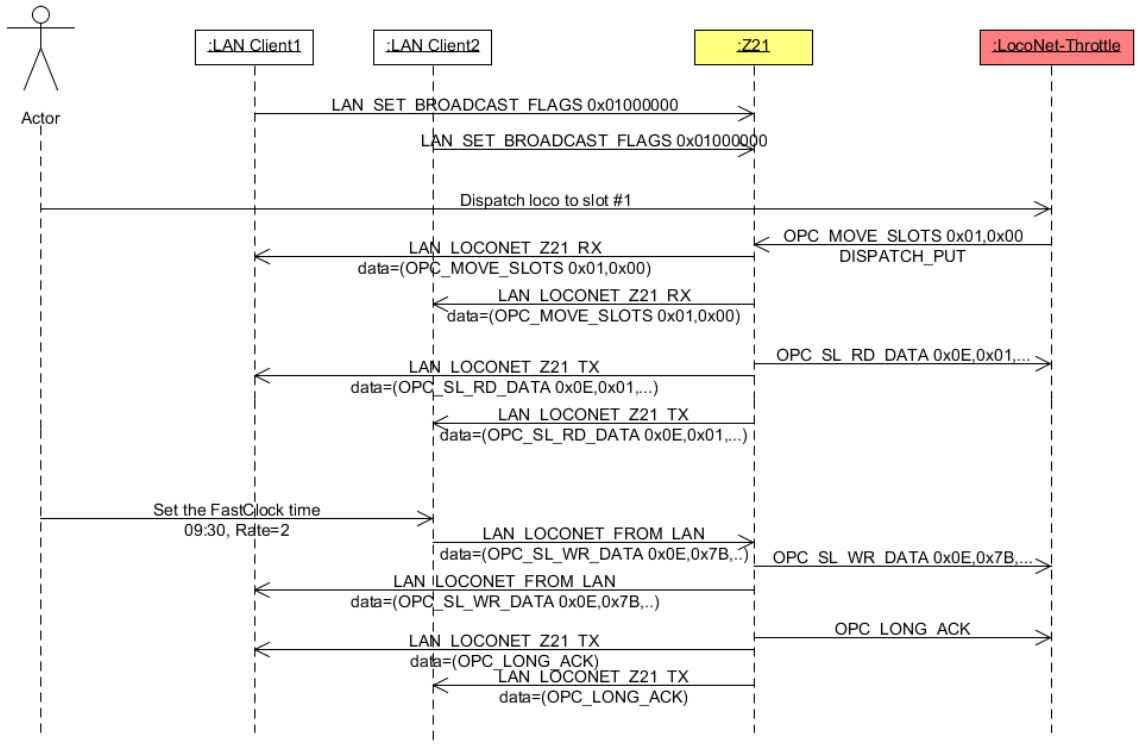

Figure 8 Example Sequence: Ethernet/LocoNet gateway

This example shows that even with trivial processes on the LocoNet bus, considerable network traffic can simultaneously occur on the Ethernet or Wi-Fi.

Please note that this Ethernet/LocoNet Gateway functionality has primarily been created for PC control SW as an additional tool for communicating with e.g. LocoNet feedback devices, etc.

When subscribing to the LocoNet messages, you should carefully consider whether the broadcast flags 0x02000000 (LocoNet locomotives) and 0x04000000 (LocoNet switches) are really necessary for your application. For conventional driving and switching, in particular, you should better use the LAN commands already described in chapters ***4*** Driving, ***5*** Switching and ***6*** Reading and writing Decoder CV. The actual LocoNet protocol is not described in more details in this specification. Please directly contact Digitrax or the manufacturer of the respective LocoNet hardware, especially if that manufacturer has extended the LocoNet protocol for e.g. configuration purposes etc.

9.1 LAN_LOCONET_Z21_RX

From Z21 FW Version 1.20.

This message is asynchronously reported to the client by the Z21 when the client

- activated the corresponding broadcast, see ***2.16*** LAN_SET_BROADCASTFLAGS, Flags 0x01000000, 0x02000000 or 0x04000000.
- and a message has been received by the Z21 from the LocoNet bus.

Z21 to Client:

DataLen		**Header**		**Data**
				LocoNet message incl. CKSUM
0x04+n	0x00	**0xA0**	0x00	n Bytes

9.2 LAN_LOCONET_Z21_TX

From Z21 FW Version 1.20.

This message is asynchronously reported to the client by the Z21 when the client

- activated the corresponding broadcast, see ***2.16*** LAN_SET_BROADCASTFLAGS, Flags 0x01000000, 0x02000000 or 0x04000000.
- and a message has been written to the LocoNet bus by the Z21.

Z21 to Client:

DataLen		**Header**		**Data**
				LocoNet message incl. CKSUM
0x04+n	0x00	**0xA1**	0x00	n Bytes

9.3 LAN_LOCONET_FROM_LAN

From Z21 FW Version 1.20.

This message allows a LAN client to write a message to the LocoNet bus.

This message is also asynchronously reported by the Z21 to a client when the client

- activated the corresponding broadcast, see ***2.16*** LAN_SET_BROADCASTFLAGS, Flags 0x01000000, 0x02000000 or 0x04000000.
- and **another** LAN client has written a message to the LocoNet bus via the Z21.

LAN client to Z21, or Z21 to LAN client:

DataLen		**Header**		**Data**
				LocoNet message incl. CKSUM
0x04+n	0x00	**0xA2**	0x00	n Bytes

9.3.1 DCC Binary State Control Instruction via LocoNet OPC_IMM_PACKET

From Z21 FW Version 1.42 on, the new command *4.3.3 LAN_X_SET_LOCO_BINARY_STATE* is recommended for switching binary states instead of using the method described below.
However, the following paragraph text, which is now somewhat outdated, remains for the sake of completeness:

From FW Version V1.25 any DCC packets can be generated at the track output using LAN_LOCONET_FROM_LAN and the LocoNet command OPC_IMM_PACKET, among them the Binary State Control Instruction (also called "F29...F32767"). This also applies to the white z21, which has no physical LocoNet interface, but however has a virtual LocoNet stack inside.

For the structure of the OPC_IMM_PACKET see LocoNet Spec (also in “personal edition” for learning purposes). For the structure of the Binary State Control Instruction see NMRA S-9.2.1 Section “Feature Expansion Instruction”.

9.4 LAN_LOCONET_DISPATCH_ADDR

From Z21 FW Version 1.20.

Prepare a loco address for the LocoNet dispatch.

This message allows a LAN client to prepare a specific locomotive address for the LocoNet dispatch. This corresponds to a "DISPATCH_PUT" and means that at the next "DISPATCH_GET" (triggered by handset controller) the slot belonging to this loco address is reported back by Z21. If necessary, the Z21 automatically occupies a free slot for this purpose.

Request to Z21:

DataLen		**Header**		**Data**
0x06	0x00	**0xA3**	0x00	16 bits Loco address (**little endian**)

Reply from Z21:
Z21 FW Version < 1.22: none
Z21 FW Version ≥ 1.22:

Z21 to Client:

DataLen		**Header**		**Data**	
0x07	0x00	**0xA3**	0x00	16 bits Loco address (**little endian**)	**Result** 8 bit

Result 0 The "DISPATCH_PUT" for the given address failed.
This can happen, for example, if the Z21 is operated as a LocoNet slave and the LocoNet master has rejected the dispatch request because this locomotive address is already assigned to another handset.

>0 The "DISPATCH_PUT" was executed successfully. The loco address can now be transferred to a handset controller (e.g. FRED). The value of Result corresponds to the current LocoNet slot number for the given loco address.

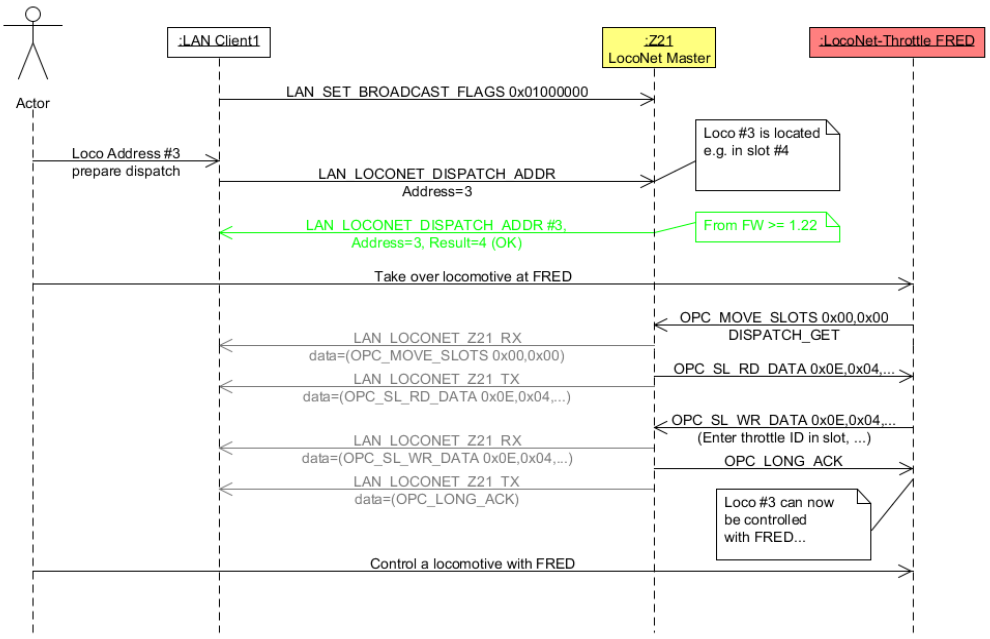

Figure 9 Example Sequence: LocoNet Dispatch per LAN-Client

9.5 LAN_LOCONET_DETECTOR

From Z21 FW Version 1.22.

If LAN client application wants to support a LocoNet track occupancy detector, there are two ways. The first would be to receive the LocoNet packets via ***9.1*** *LAN_LOCONET_Z21_RX* and process the corresponding LocoNet messages directly. However, this requires an exact knowledge of the LocoNet protocol and it would produce a lot of network traffic.

Therefore the following alternative was created, with which you can **poll** the occupied status **as well as** be **asynchronously informed** about a change of the occupied status, without having to go into the depths of the LocoNet protocol.

Information: please note the following essential difference between the Roco Feedback Module 10787 on the R-BUS (see ***7*** Feedback – R-BUS) and LocoNet Track Occupancy Detectors:

- 10787 is normally connected with mechanically operated switching contacts, which can be closed and reopened per axis of the train running over it.

- LocoNet track occupancy detectors are usually based on exact current measurement at the monitored track section or on advanced technologies (transponder, infrared, RailCom, ..) in order to reliably determine the occupancy state of the track. During normal operation, ideally only one message is generated when the occupied state changes.

The following command can be used to poll the status of one or more track occupancy detectors.

Request to Z21:

DataLen		**Header**		**Data**	
0x07	0x00	**0xA4**	0x00	**Type** 8 bit	16 bits report address (**little endian**)

Type **0x80** Request via "Stationary Interrogate Request" (SIC) according to Digitrax procedure. This procedure also can be used for the occupancy detectors from Blücher-Elektronik. The parameter “report address” is 0 (not relevant).

0x81 Request via so-called **report address** for Uhlenbrock occupancy detector.
This report address can be configured by the user e.g. UB63320 via LNCV 17 in the occupancy detector. The default value there is 1017.
With type 0x81, this report address is only used for polling and should **not be confused** with the **feedback address**.
Note: At the LocoNet bus this query is implemented via turnout switching commands, therefore the value according to LocoNet has to be decremented by 1. Example:
`0x07 0x00 0xA4 0x00 0x81 0xF8 0x03`
means: "request status of all occupancy detectors with report address 1017 (Report address = 1017 = **0x03F8** +1 = 1016 + 1)"

0x82 **Status request for LISSY, from Z21 FW Version 1.23**
On the other hand, for Uhlenbrock LISSY the report address corresponds to the feedback address however. The type of the subsequent feedback message(s) strongly depends on the configured operating mode of the LISSY receiver. In the LISSY manual you can find out more about the extensive setting options of the LISSY receiver.

Please note that in the case of a single request, several occupancy detectors may be addressed at the same time, and therefore multiple responses are to be expected. Depending on the manufacturer of the occupancy detector, the status of the same input can be also reported several times after one request.

Reply from Z21:

Z21 to Client:

DataLen		Header		Data		
0x07 + ***n***	0x00	**0xA4**	0x00	**Type** 8 bit	**Feedback address** 16 bits (**little endian**)	**Info**[***n***]

This message is asynchronously reported to the client by the Z21 when the client

- has activated the corresponding broadcast, see ***2.16*** LAN_SET_BROADCASTFLAGS, Flag 0x08000000
- and the Z21 has received a corresponding message from a track occupancy detector **due to a status change on its input**, or **due to an explicit status-request (polling) by a LAN client** using the commands described above.

Feedback address Each input of an occupancy detector has its own feedback address, which can be configured by the user (e.g. for Uhlenbrock and Blücher via LNCV), and which describes the monitored block with an unique address.

Info[*n*] Byte-Array; content and length ***n*** depending on **Type**, see below

Type **0x01** For occupancy detector types like Uhlenbrock 63320 or Blücher GBM16XL reporting only the status "occupied" and "free" (by using LocoNet OPC_INPUT_REP, X=1).

n=1

Status of the input belonging to the feedback address can be found in **Info[0]:**
Info[0]=0 ... sensor is **LOW** (“free”)
Info[0]=1 ... sensor is **HIGH** (“occupied”)

0x02 **Transponder Enters Block**
0x03 **Transponder Exits Block**
For occupancy detectors such as Blücher GBM16**XN** etc. reporting also the information about the vehicle (e.g. locomotive address) inside the block to the command station (by using LocoNet OPC_MULTI_SENSE transponding encoding from Digitrax).
In addition to the feedback address, also a so-called transponder address is transmitted. The **transponder address** identifies the vehicle in the block. In the case of the GBM16**XN**, this is the **locomotive address** which was determined by the occupancy detector by using RailCom.

n=2

The transponder address is located in **Info[0]** und **Info[1]**, 16 Bit little endian:
Info[0] ... transponder address low byte
Info[1] ... transponder address high byte

Remark: due to lack of specification inside the LocoNet spec, the value ranges in OPC_MULTI_SENSE is not quite clear, which leaves the manufacturers of the occupancy detectors sometimes in the dark. Therefore in the case of GBM16XN the following must be observed according to our experience:

- You have to add +1 to the feedback address to get the feedback address configured in GBM16**XN**.
- Depending on the configuration of the GBM16**XN**, the direction of the vehicle on the track can also be coded in the bit under the mask 0x1000. Such a configuration is not recommended because this bit collides with the address space of long locomotive addresses!

0x10 LISSY Loco address from Z21 FW 1.23.

This message is sent to the Z21 LAN Client when an Uhlenbrock LISSY receiver reports a vehicle equipped with a LISSY transmitter and this LISSY receiver is configured to the "Transfer format (ÜF) Uhlenbrock" (LNCV 15=1). Furthermore, this message strongly depends on the configured operating mode (LNCV2, ...) of the LISSY receiver.
See also LISSY manual.

n=3

The Loco address is located in **Info[0]** und **Info[1]**, 16 Bit little endian:
Info[0] ... loco address low byte
Info[1] ... loco address high byte
Locomotives have a value range from 1..9999
Wagons have a value range from 10000 to 16382

Info[2] ... Additional info according to following bits: 0 **DIR1 DIR0** 0 **K3 K2 K1 K0**
DIR1=0: **DIR0** is to be ignored
DIR1=1: **DIR0**=0 is forwards, **DIR0**=1 is backwards
K3..K0: 4 bit “class information” stored in the LISSY sender.

Example Configuration for Lissy Receiver 68610:

LNCV	Value	Comment
2	98	optional module reset: sets all LNCV to 0, except LNCV 0 and 1 (address)
2	0	Basic function: readout locomotive data and direction information by using double sensor
15	1	Send using “Transfer format (ÜF) Uhlenbrock” onto LocoNet

0x11 LISSY block status from Z21 FW 1.23.

This message is sent to the Z21 LAN Client when an Uhlenbrock LISSY receiver sends a block occupancy status message using the "Transfer format (ÜF) Uhlenbrock". See also LISSY manual.

n=1

Status of the block belonging to the feedback address is in **Info[0]**:
Info[0]=**0** ... block is free
Info[0]=**1** ... block is occupied

Example Configuration for Lissy Receiver 68610:

LNCV	Value	Comment
2	98	optional module reset: sets all LNCV to 0, except LNCV 0 and 1 (address)
2	22	Automation function with block status message: Time-controlled waiting station
3	2	Automation active in both driving directions
4	3	Wait time 3 seconds
10	2	Block option: Block status change to "free" after 2 seconds
15	1	Send using “transfer format (ÜF) Uhlenbrock” onto LocoNet

0x12 LISSY Speed from Z21 FW 1.23.
This message is sent to the Z21 LAN Client when a Uhlenbrock LISSY receiver is configured for speed measurement.
See also LISSY manual.

n=2

The speed is located in **Info[0]** and **Info[1]**, 16 bit little endian:
Info[0] ... speed low byte
Info[1] ... speed high byte

Example Configuration for Lissy Receiver 68610:

LNCV	Value	Comment
2	98	optional module reset: sets all LNCV to 0, except LNCV 0 and 1 (address)
2	0	Basic function: readout locomotive data and direction information by using double sensor
14	**15660**	**Velocity Scaling factor = 1566 (H0-scale) * 10mm (sensor distance)**
15	1	Send transfer format Uhlenbrock to LocoNet

Note: **Type** will be extended by further IDs in the future as required.

10 CAN

10.1 LAN_CAN_DETECTOR

From Z21 FW Version 1.30.

The Roco CAN occupancy detector 10808 is supported from FW version 1.30 on. The occupancy detector can be used by the LAN client in four different ways:

1. **R-BUS emulation:** the CAN detector is forwarded to the LAN client as R-BUS detector in the Z21 firmware. So the LAN client can use the CAN detector as described in Chapter 7 Feedback - R-BUS.

2. **LocoNet emulation**: the CAN detector is forwarded in the Z21 firmware as LocoNet detector to the LAN client. Thus the LAN client can use the CAN detector as described in chapter 9.5 *LAN_LOCONET_DETECTOR* (type 0x01 "occupied/free" and the locomotive address via type 0x02 and 0x03 "Transponder Enters Block, Transponder Exits Block").

3. **LISSY emulation**: The CAN detector is emulated in the Z21 firmware by LISSY/Marco messages. The LAN client can use the CAN detector as described in chapter 9.5 *LAN_LOCONET_DETECTOR* (type 0x10 "Locomotive address" and type 0x11 "block status").

4. Direct access by the command LAN_CAN_DETECTOR (see below).

The type of emulation can be configured via the Z21 Maintenance Tool. The factory setting is: **R-BUS emulation=on**, **LocoNet emulation=on**, *LISSY emulation=off*.

However, the fastest method, however, and the one that is most economic concerning memory and bandwidth, is direct access using the command **LAN_CAN_DETECTOR 0xC4**. This is particularly recommended when a large number of CAN occupancy detectors are used at the same time. With the following command, the status of the CAN occupancy detectors can be polled:

Request to Z21:

DataLen		**Header**		**Data**	
0x07	0x00	**0xC4**	0x00	**Type** 8 bit	**CAN-NetworkID** 16 bit (little endian)

Type **0x00** Request the CAN occupancy detector with the given CAN network ID.
The CAN NetworkID **0xD000** means "**all CAN detectors**".
Example:
`0x07 0x00 0xC4 0x00 0x00 0x00 0xD0`
means: "poll status of all CAN occupancy detectors"

Please note that with a single request several CAN occupancy detectors are addressed at the same time, and therefore multiple responses are to be expected. Depending on the configuration of the emulation, the status of one and the same input can also be reported several times.

Reply from Z21:

Z21 to Client:

DataLen		Header		Data					
0x0E	0x00	**0xC4**	0x00	**NId** 16 bit	**Addr** 16 bit	**Port** 8 bit	**Type** 8 bit	**Value1** 16 bit	**Value2** 16 bit

This message is also reported to the client by the Z21 asynchronously when the client

- Has activated the corresponding broadcast, see ***2.16*** LAN_SET_BROADCASTFLAGS, Flag 0x00080000
- and the Z21 has received a corresponding message from the CAN detector, due to a **status change at its input**, **OR** due to an **explicit polling by a LAN client** using commands described above.

All 16 bit values are little endian coded.

NId Unchangeable CAN network ID of the occupancy detector.

Addr Configurable module address of the occupancy detector. Each CAN occupancy detector has a module address which can be set by the user.

Port Input pin number of the CAN occupancy detector (0 to 7)

Type **0x01** **Occupancy status** of input (free, busy, overload)

0x11 1st and 2nd recognized **locomotive address** at the entrance
0x12 3rd and 4th recognized **locomotive address** at the entrance
…
0x1F 29th and 30th. recognized **locomotive address** at the entrance

The value of Value1 and Value2 depends on the type.

If Type = 0x01 (occupied status):

Value1	**0x0000**	**Free, without voltage**
	0x0100	**Free, with voltage**
	0x1000	**Occupied, without voltage**
	0x1100	**Occupied, with voltage**
	0x1201	**Occupied, Overload 1**
	0x1202	**Occupied, Overload 2**
	0x1203	**Occupied, Overload 3**

If Type = 0x11 to 0x1F (RailCom Loco address):
Type 0x11 to 0x1F form a list of locomotive addresses.
This vehicle list ends with the locomotive address=0.

Value1 First detected locomotive address in section incl. direction information.
0 = no locomotive address detected (e.g. with decoder not capable of RailCom, or no locomotive), resp. end of locomotive address list

Value2 Second detected locomotive address in section incl. direction information.
0 = no locomotive address detected, resp. end of locomotive address list

The direction information is coded in the most significant two bits of Value1 resp. Value2:
0 x No direction detected
1 0 Vehicle has been placed forward on the track
1 1 Vehicle has been placed backwards on the track
The lower 14 bits contain the locomotive address.

10.2 CAN Booster

From Z21 FW Version 1.41.

CAN Booster Management with Roco CAN booster 10806, 10807 and 10869 are supported from Z21 FW version 1.41 on. Of course the following LAN command work only, if these boosters are connected to the Z21 with the CAN-Bus (and not with B-BUS).

10.2.1 LAN_CAN_DEVICE_GET_DESCRIPTION

Read the name from the booster.

The user can store a name (free text) in the booster so that he can later identify the device more easily.

Request to Z21:

DataLen		**Header**		**Data**
0x06	0x00	**0xC8**	0x00	**NId** 16 bit

NId is the CAN-NetworkID of the addressed booster (16 bit little endian, 0xC101 to 0xC1FF).
See also below ***10.2.3*** LAN_CAN_BOOSTER_SYSTEMSTATE_CHGD.

Z21 to Client:

DataLen		**Header**		**Data**	
0x16	0x00	**0xC8**	0x00	**NId** 16 bit	UINT8 **Name**[16]

Name corresponds to the stored designation as a null-terminated string. The string should be encoded according to **ISO 8859-1** (*Latin-1*).

Hint: do not request two LAN_CAN_BOOSTER_GET_DESCRIPTION directly one after another. Instead wait for the reply to the first request, and then send the second request.

Hint: With ***10.2.3*** LAN_CAN_BOOSTER_SYSTEMSTATE_CHGD NetworkIDs of all CAN boosters connected in the system.

10.2.2 LAN_CAN_DEVICE_SET_DESCRIPTION

Overwrite the name in the booster.

Request to Z21:

DataLen		**Header**		**Data**	
0x16	0x00	**0xC9**	0x00	**NId** 16 bit	UINT8 **Name**[16]

NId is the CAN-NetworkID of the addressed booster (16 bit little endian, 0xC101 to 0xC1FF).
See also below ***10.2.3*** LAN_CAN_BOOSTER_SYSTEMSTATE_CHGD.

Name corresponds to the stored designation as a null-terminated string. The string should be encoded according to **ISO 8859-1** (*Latin-1*). Fill the rest of data with 0x00.
Not allowed characters are the quotation mark " (0x22) and the backslash \ (0x5C).

Reply from Z21:
None

10.2.3 LAN_CAN_BOOSTER_SYSTEMSTATE_CHGD

Report the system status of the CAN booster to the client. This message comes about once per second per CAN booster and booster output.

This message is reported asynchronously from the control panel to the client if it

- has activated the corresponding broadcast,
  see ***2.16*** LAN_SET_BROADCASTFLAGS, Flag 0x00020000
- and at least one booster is connected to the control panel via CAN.

Z21 to Client:

DataLen		**Header**		**Data**
0x0E	0x00	**0xCA**	0x00	**CANBoosterSystemState** (10 Bytes)

CANBoosterSystemState is structured as follows (the 16-bit values are little endian):

Byte Offset	**Typ**	**Name**	**Wert**	
0	UINT16	**NId**	0xC101 … 0xC1FF	**CAN-NetworkID** of the booster
2	UINT16	Booster_OutputPort	1 2	1st track output 2nd track output (10807 only)
4	UINT16	Booster_State	bitmask	see below
6	UINT16	Booster_VCCVoltage	mV	voltage track output
8	UINT16	Booster_Current	mA	current track output

Bitmasks for **Booster_State**:

```
#define bsBgActive                  0x0001 // brake generator active (ZCAN SSP)
#define bsShortCircuit              0x0020 // short circuit on track (ZCAN UES)
#define bsTrackVoltageOff           0x0080 // track is switched off (OFF)
#define bsRailComActive             0x0800 // RailCom-Cutout active
```

From Booster FW Version V1.11 (Booster Manangement)**:**

```
#define bsOutputDisabled            0x0100 // track is deactivated (by user)
```

10.2.4 LAN_CAN_BOOSTER_SET_TRACKPOWER

Booster Management by user: deactivate and reactivate CAN Booster track outputs.

Request to Z21:

DataLen		**Header**		**Data**	
0x07	0x00	**0xCB**	0x00	**NId** 16 bit	**Power** 8 bit

NId is the CAN-NetworkID of the addressed booster (16 bit little endian, 0xC101 to 0xC1FF).
See also below ***10.2.3*** LAN_CAN_BOOSTER_SYSTEMSTATE_CHGD.

Power 0x00 … deactivate all booster track outputs
0xFF … reactivate all booster track outputs
Additionally, **from Z21 FW Version V1.42** and **Booster FW Version V1.11**:
0x10 … deactivate 1st booster track output
0x11 … reactivate 1st booster track output
0x20 … deactivate 2nd booster track output (Z21 dual BOOSTER)
0x22 … reactivate 2nd booster track output (Z21 dual BOOSTER)

Hint: The booster track outputs can only be switched on again if the Z21 is also switched on and is sending a valid track signal to the CAN boosters.
The settings of the booster management are not saved persistently.

Reply from Z21:
On change of the CANBoosterSystemState ***10.2.3*** LAN_CAN_BOOSTER_SYSTEMSTATE_CHGD to subscribed clients.

11 zLink

The **zLink interface,** introduced for the first time with Z21 single BOOSTER, also allows devices with smaller microcontrollers (without own LAN or Wi-Fi hardware interface) to be integrated into the network of the user.

Terminal devices with zLink interface are (01/2021):

- 10806 Z21 single BOOSTER
- 10807 Z21 dual BOOSTER
- 10869 Z21 XL BOOSTER
- 10836 Z21 switch DECODER
- 10837 Z21 signal DECODER

11.1 Adapter

An adapter can be connected to the zLink interface of the terminal devices mentioned above, via which the terminal device can communicate with the outside world.
One such adapter is the **10838 Z21 pro LINK**.

11.1.1 10838 Z21 pro LINK

The **10838 Z21 pro LINK** connects the **zLink interface to the Wi-Fi** as a **gateway** and can thus be used for the following tasks:

1. **Configuring** the terminal device (buttons & display, Z21 App, Z21 Maintenance Tool on PC)
2. **Firmware update** of the terminal device (Z21 Updater App, Z21 Maintenance Tool on PC)
3. Controlling the terminal device by Wi-Fi clients using the **Z21 LAN protocol**

The latter point 3 was initially intended as a test interface, but it soon became clear that this would open up promising possibilities for decentralized layouts connected via Wi-Fi. From a technical point of view, it means that a small Z21 protocol stack is implemented inside the terminal device. However, that reduced Z21 protocol stack had to be tailored to the purpose of the terminal device due to limited memory. See also *Table 1: Messages from Client to Z21* and *Table 2: Messages from Z21 to Clients*. As usual, commands can now be sent to the terminal device using UDP via the Wi-Fi / zLink gateway - just like to a Z21. For example, the track outputs of a booster can be switched on and off via the WLAN / zLink gateway, or the booster system status can be queried. Turnouts can also be switched on the Z21 switch DECODER directly, and signals can be controlled on the Z21 signal DECODER, even without any connection to the main track of the DCC control center. The decoders can even be configured via the zLink interface by using LAN commands for writing CVs.

An attempt was made to make it as transparent as possible for the Wi-Fi client, whether it is communicating with a control center or with a terminal device via the Wi-Fi / zLink gateway. However, regarding the very small CPU in the end device, following points should be considered:

- Restricted bandwidth: the effective transfer rate should remain well below 1024 bytes/s per terminal device. With the devices currently available, more would neither be necessary nor reasonable.
- Give the end device enough time to process the commands and data. Therefore, keep a pause of at least 50 ms between two requests.
- Use Z21 pro LINK preferably in client mode.
- If possible, connect only one Wi-Fi client to a Z21 pro LINK, max. 4 clients allowed.

Operation via UPD broadcasts is possible, but it is recommended to only do so for searching the devices in the network (see below). The terminal devices can then be clearly identified by their hardware type (LAN_GET_HWINFO) and serial number (LAN_GET_SERIAL_NUMBER), as well as by the IP address of the respective Z21 pro LINK. In addition, the user can store a name (free text) in each terminal device, which can also be displayed in the user interface.

LAN_ZLINK_GET_HW_INFO is an example for a command that the Z21 pro LINK does not forward to its terminal device, but rather processes and answers the request itself.

11.1.1.1 LAN_ZLINK_GET_HWINFO

This command can be used to query the properties of the Z21 pro LINK.

If this command is sent as a UDP broadcast, it is possible to use the responses to discover the Z21 pro LINKs registered in the Wi-Fi network and then manage a list with their respective MAC address and assigned IP address.

Request to Z21 pro LINK :

DataLen		**Header**		**Data**
0x05	0x00	**0xE8**	0x00	**0x06**

Data[0] = **0x06** = ZLINK_MSG_TYPE_HW_INFO

Reply from Z21 pro LINK:

DataLen		**Header**		**Data**	
0x3F	0x00	**0xE8**	0x00	**0x06**	**Z_Hw_Info** (58 Bytes)

Data[0] = **0x06** = ZLINK_MSG_TYPE_HW_INFO

Z_Hw_Info is structured as follows (the 16-bit values are little endian):

Byte Offset	**Type**	**Name**		**Example**
0	UINT16	**HwID**		401 (0x191)
2	UINT8	FW_Version_Major		1
3	UINT8	FW_Version_Minor		1
4	UINT16	FW_Version_Build		3217 (0xC91)
6	UINT8[18]	**MAC_Address**	string	„EC FA BC 4F 04 C6“
24	UINT8[33]	**Name**	string	„this_is_a_quite_long_device_name“
57	UINT8	Reserved	0x00	0

HwID
401 = 0x191 … Adapter **10838 Z21 pro LINK**

MAC_Address
MAC address of the adapter as a null-terminated string, 8-bit ASCII.

Name
User-configurable name of the adapter as a null-terminated string. Maximum 32 characters plus 0x00 zero terminator, encoding: 8-bit ISO 8859-1 (*Latin-1*).
Ignore all characters after the first 0x00.

Example:

```
                                  3f 00 e8 00 06 91        ?.....
01 01 01 91 0c 45 43 20 46 41 20 42 43 20 34 46  .....EC FA BC 4F
20 30 34 20 43 36 00 74 68 69 73 5f 69 73 5f 61   04 C6.this_is_a
5f 71 75 69 74 65 5f 6c 6f 6e 67 5f 64 65 76 69  _quite_long_devi
63 65 5f 6e 61 6d 65 00 00                       ce_name..
```

HwID: **0x191** = 401 = 10838 Z21 pro LINK
FW Version: V1.1.3217
MAC Adresse: „EC FA BC 4F 04 C6“
Name: „this_is_a_quite_long_device_name”

11.2 Booster 10806, 10807 und 10869

Available commands for boosters see *Table 1: Messages from Client to Z21* and *Table 2: Messages from Z21 to Clients*. In addition, some new commands have been introduced that are only valid for the boosters.

11.2.1 LAN_BOOSTER_GET_DESCRIPTION

Read the name from the booster.

The user can store a name (free text) in the booster so that he can later identify the device more easily.

Request to BOOSTER:

DataLen		Header		Data
0x04	0x00	**0xB8**	0x00	-

Reply from BOOSTER:

DataLen		Header		Data
0x24	0x00	**0xB8**	0x00	UINT8 **Name**[32]

Name corresponds to the stored designation as a null-terminated string. The string should be encoded according to **ISO 8859-1** (*Latin-1*) and should not be longer than 16 characters for compatibility with the CAN bus.

Special case: Name [0] can be 0xFF if a name has never been stored in the device. This case must be interpreted as an empty string.

Example: "Test"

```
                                  24 00 b8 00 54 65              $...Te
73 74 00 00 00 00 00 00 00 00 00 00 00 00 00 00   st..............
00 00 00 00 00 00 00 00 00 00 00 00 00 00         ..............
```

11.2.2 LAN_BOOSTER_SET_DESCRIPTION

Overwrite the name in the booster.

Request to BOOSTER:

DataLen		Header		Data
0x24	0x00	**0xB9**	0x00	UINT8 **Name**[32]

Name corresponds to the stored designation as a null-terminated string. The string should be encoded according to **ISO 8859-1** (*Latin-1*) and should not be longer than 16 characters for compatibility with the CAN bus. Fill the rest of data with 0x00.
Not allowed characters are the quotation mark " (0x22) and the backslash \ (0x5C).

Reply from BOOSTER:
None

11.2.3 LAN_BOOSTER_SYSTEMSTATE_GETDATA

Request the current system state.

Request to BOOSTER:

DataLen		Header		Data
0x04	0x00	**0xBB**	0x00	-

Reply from BOOSTER:
See below **11.2.4** LAN_BOOSTER_SYSTEMSTATE_DATACHANGED

11.2.4 LAN_BOOSTER_SYSTEMSTATE_DATACHANGED

Report a change in system state of the booster to the client.

This message is reported asynchronously from the booster to the client when the latter

- has activated the corresponding broadcast,
  see ***2.16*** LAN_SET_BROADCASTFLAGS, Flag 0x00000100
- has explicitly requested the system state,
  see above ***11.2.3*** LAN_BOOSTER_SYSTEMSTATE_GETDATA.

BOOSTER to Client:

DataLen		Header		Data
0x1C	0x00	**0xBA**	0x00	**BoosterSystemState** (24 Bytes)

BoosterSystemState is structured as follows (the 16-bit values are little endian):

Byte Offset	Type	Name		
0	INT16	Booster_1_MainCurrent	mA	current 1st track output
2	INT16	Booster_2_MainCurrent	mA	current 2nd track output
4	INT16	Booster_1_FilteredMainCurrent	mA	smoothed current 1st track output
6	INT16	Booster_2_FilteredMainCurrent	mA	smoothed current 2nd track output
8	INT16	Booster_1_Temperature	°C	temperature 1st amplifier
10	INT16	Booster_2_Temperature	°C	temperature 2nd amplifier
12	UINT16	SupplyVoltage	mV	supply voltage
14	UINT16	Booster_1_VCCVoltage	mV	voltage 1st track output
16	UINT16	Booster_2_VCCVoltage	mV	voltage 2nd track output
18	UINT8	CentralState	bitmask	see below
19	UINT8	CentralStateEx	bitmask	see below
20	UINT8	CentralStateEx2	bitmask	see below
21	UINT8	Reserved1		
22	UINT8	CentralStateEx3	bitmask	see below
23	UINT8	Reserved2		

Bitmasken für **CentralState**:

```
#define csTrackVoltageOff            0x02 // track is switched off
#define csConfigMode                 0x10 // configuration mode is active
#define csCanConnected               0x20 // CAN connection with Z21 is Ok
```

Bitmasken für **CentralStateEx**:

```
#define cseHighTemperature              0x01 // over temperature
#define csePowerLost                    0x02 // supply voltage too low
#define cseBooster_1_ShortCircuit       0x04 // short circuit on 1st track output
#define cseBooster_2_ShortCircuit       0x08 // short circuit on 2nd output
#define cseRevPol                       0x10 // supply voltage error
#define cseNoDCCInput                   0x80 // no DCC input signal
```

Bitmasken für **CentralStateEx2**:

```
#define cse2Booster_1_RailComActive     0x01 // RailCom active 1st track output
#define cse2Booster_2_RailComActive     0x02 // RailCom active 2nd track output
#define cse2Booster_1_MasterSettings    0x04 // CAN autosettings Ok 1st track output
#define cse2Booster_2_MasterSettings    0x08 // CAN autosettings Ok 2nd track output
#define cse2Booster_1_BgActive          0x10 // brake generator active 1st track output
#define cse2Booster_2_BgActive          0x20 // brake generator active 2nd track output
#define cse2Booster_1_RailComFwd        0x40 // RailCom forwarding active 1st track o.
#define cse2Booster_2_RailComFwd        0x80 // RailCom forwarding active 2nd track o.
```

Bitmasken **fürCentralStateEx3**:

```
#define cse3Booster_1_OutputInverted    0x01 // 1st track output inverted (autoinvert)
#define cse3Booster_2_OutputInverted    0x02 // 2nd track output inverted (autoinvert)
From Booster FW Version 1.11:
#define cse3Booster_1_OutputDisabled    0x10 // Track output 1 deactivated (by user)
#define cse3Booster_2_OutputDisabled    0x20 // Track output 2 deactivated (by user)
```

11.2.5 LAN_BOOSTER_SET_POWER

From Booster FW Version 1.11: Booster management by user.

If all track outputs are deactivated or reactivated here on the booster, then this command de facto corresponds to a LAN_X_SET_TRACK_POWER_OFF or LAN_X_SET_TRACK_POWER_ON to the booster. With LAN_BOOSTER_SET_POWER, on the other hand, it is possible to switch one dedicated track output off and on in the 10807 Z21 dual BOOSTER.

Request to BOOSTER:

DataLen		**Header**		**Data**	
0x06	0x00	**0xB2**	0x00	**BoosterPort 8 bit**	**BoosterPortState 8 bit**

BoosterPort
0x01 … select 1st booster track output
0x02 … select 2nd booster track output (Z21 dual BOOSTER only)
0x03 … select all booster track outputs

BoosterPortState
0x00 … deactivate selected booster track outputs
0x01 … reactivate selected booster track outputs

Hint: The booster track outputs can only be switched on again if the Z21 is also switched on and is sending a valid track signal to the CAN boosters.
The settings of the booster management are not saved persistently.

Reply from BOOSTER:
On change of the BoosterSystemState **11.2.4** LAN_BOOSTER_SYSTEMSTATE_DATACHANGED to subscribed clients.

11.3 Decoder 10836 und 10837

Available commands for decoders see *Table 1: Messages from Client to Z21* and *Table 2: Messages from Z21 to Clients*. In addition, some new commands have been introduced that are only valid for the decoders.

11.3.1 LAN_DECODER_GET_DESCRIPTION

Read the name from the decoder.

The user can store a name (free text) in the decoder so that he can later identify the device more easily.

Request to DECODER:

DataLen		Header		Data
0x04	0x00	**0xD8**	0x00	-

Reply from DECODER:

DataLen		Header		Data
0x24	0x00	**0xD8**	0x00	UINT8 **Name**[32]

Name encoding see 11.2.1 LAN_BOOSTER_GET_DESCRIPTION

11.3.2 LAN_DECODER_SET_DESCRIPTION

Overwrite the name in the decoder.

Request to DECODER:

DataLen		Header		Data
0x24	0x00	**0xD9**	0x00	UINT8 **Name**[32]

Name encoding see 11.2.2 LAN_BOOSTER_SET_DESCRIPTION.

Reply from DECODER:
None

11.3.3 LAN_DECODER_SYSTEMSTATE_GETDATA

Request the current system state.

Request to DECODER:

DataLen		Header		Data
0x04	0x00	**0xDB**	0x00	-

Reply from DECODER:

See below ***11.3.4*** LAN_DECODER_SYSTEMSTATE_DATACHANGED

11.3.4 LAN_DECODER_SYSTEMSTATE_DATACHANGED

Report a change in system state of the decoder to the client.

This message is reported asynchronously from the decoder to the client when the latter

- has activated the corresponding broadcast,
  see ***2.16*** LAN_SET_BROADCASTFLAGS, Flag 0x00000100
- has explicitly requested the system state,
  see above ***11.3.3*** LAN_DECODER_SYSTEMSTATE_GETDATA.

If the signal decoder does not report after 4 seconds despite the subscription via broadcastflags, because e.g., no signal aspect has changed, polling can also be carried out if necessary.

The responses of 10836 Z21 switch DECODER and 10837 Z21 signal DECODER **differ in structure and content** and can be recognized by **DataLen.**

11.3.4.1 SwitchDecoderSystemState

10836 Z21 switch DECODER to client:

DataLen		**Header**		**Data**
0x30	0x00	**0xDA**	0x00	**SwitchDecoderSystemState** (44 Bytes)

SwitchdecoderSystemState is structured as follows (the 16-bit values are little endian):

Byte Offset	**Type**	**Name**		
0	INT16	Current	mA	current
2	INT16	**FilteredCurrent**	mA	smoothed current
4	UINT16	Voltage	mV	internal voltage (3.3V)
6	UINT8	**CentralState**	bitmask	see below
7	UINT8	**CentralStateEx**	bitmask	see below
8	UINT8[8]	**OutputStates**[0..7]		Status per output
16	UINT8[8]	**OutputConfig**[0..7]		Operating mode per output
24	UINT8[4]	**OutputDimm**[0..7]		Dimming value per output
32	UINT16	**Address**		First decoder address
34	UINT16	**Address2**		Second decoder address
36	UINT8[6]	Reserved1		
42	UINT8	**Dimmed**		1 bit per output
43	UINT8	Reserved2		

FilteredCurrent
Sum of internal current + current at the terminal clamps.

Bitmasks for **CentralState**:

```
#define csEmergencyStop          0x01   // The emergency stop for decoder
#define csTrackVoltageOff        0x02   // The track voltage is switched off
#define csShortCircuit           0x04   // Short-circuit
#define csConfigMode             0x10   // configuration mode is active
```

Bitmasks for **CentralStateEx**:

```
#define csePowerLost             0x02   // Input voltage too low
#define cseRCN213                0x20   // addressing conform with RCN213
#define cseNoDCCInput            0x80   // no DCC input signal
```

OutputState
State of the output (enumeration)

```
#define oUnknown                0x00
#define oRedActive              0x11
#define oRedInactive            0x01
#define oGreenActive            0x12
#define oGreenInactive          0x02
```

OutputConfig
Operating mode of the output (enumeration)

```
#define ocfgNormal              0      // Impulsbetrieb (default)
#define ocfgBlinker             1      // Wechselblinker
#define ocfgBlinkSm             2      // Wechselblinker mit Ein- und Ausblenden
#define ocfg10775               3      // Momentbetrieb wie 10775
#define ocfgK84                 4      // Dauerbetrieb (zB für Beleuchtung)
#define ocfgK84Sm               5      // Dauerbetrieb mit Ein- und Ausblenden
```

OutputDimm
Dimming value
0 ... dimming deactivated, therefore corresponds to full output power.
1 to 100 ... minimum to maximum possible output power.

Address
A decoder address corresponds to 4 turnout numbers. That is:
First decoder address = 1 ... Turnout number 1 to 4
First decoder address = 2 ... Turnout number 5 to 8
First decoder address = 3 ... Turnout number 9 to 12
and so on…

Address2
Second decoder address **= 0**: 2nd decoder address **automatically** is "1st decoder address + **1**"
otherwise:
Second decoder address = 1 ... Turnout number 1 to 4
Second decoder address = 2 ... Turnout number 5 to 8
Second decoder address = 3 ... Turnout number 9 to 12
and so on...

Dimmed
1 bit per output pair:
0 ... Output pair is not dimmed.
1 ... Output pair is dimmed, or smooth dimming is configured by operation mode (light bulb simulation)
LSB = output pair 1; MSB = Output pair 8

11.3.4.2 SignalDecoderSystemState

10837 Z21 signal DECODER to client:

DataLen		Header		Data
0x2E	0x00	**0xDA**	0x00	**SignalDecoderSystemState** (42 Bytes)

SignalDecoderSystemState is structured as follows (the 16-bit values are little endian):

Byte Offset	Type	Name		
0	INT16	Current	mA	0 / Reserved
2	INT16	FilteredCurrent	mA	0 / Reserved
4	UINT16	**Voltage**	mV	voltage at the clamps
6	UINT8	**CentralState**	bitmask	see below
7	UINT8	**CentralStateEx**	bitmask	see below
8	UINT8[2]	**OutputStates**[0..1]		on/off status for outputs A1... B8
10	UINT8[2]	**BlinkStates**[0..1]		blinking status for outputs A1... B8
12	UINT8[4]	**SignalDccExt**[0..3]	DCCext	current signal aspect 1st to 4th signal
16	UINT8[4]	SignalCurrAsp[0..3]	index	current signal aspect 1st to 4th signal
20	UINT8[3]	Reserved1		
23	UINT8	**SignalCount**	2, 3, 4	number of signals used
24	UINT8[4]	**SignalConfig**[0..3]	Signal-ID	signal configuration 1st to 4th signal
28	UINT8[4]	SignalInitAsp[0..3]	index	Initialization 1st to 4th signal
32	UINT16	**Address**		First decoder address
34	UINT16[4]	Reserved2		

Bitmasks for **CentralState**:

```
#define csEmergencyStop          0x01   // The emergency stop for decoder
#define csTrackVoltageOff        0x02   // The track voltage is switched off
#define csShortCircuit           0x04   // Short-circuit
#define csConfigMode             0x10   // configuration mode is active
```

Bitmasks for **CentralStateEx**:

```
#define csePowerLost             0x02   // Input voltage too low
#define cseEEPromError           0x10   // EEPROM write / read error
#define cseRCN213                0x20   // addressing conform with RCN213
#define cseNoDCCInput            0x80   // no DCC input signal
```

OutputStates
OutputStates[0]: LSB = Output A1; MSB = Output A8
OutputStates[1]: LSB = Output B1; MSB = Output B8

BlinkStates
BlinkStates[0]: LSB = Output A1; MSB = Output A8
BlinkStates[1]: LSB = Output B1; MSB = Output B8

SignalDccExt and **SignalConfig**
SignalConfig exactly defines the configured signal type (**Signal-ID** value).
SignalDccExt defines the currently visible signal aspect (**DCCext** value) for the given Signal-ID.
For Signal-ID and DCCext values, see https://www.z21.eu/en/products/z21-signal-decoder/signaltypen.

Address
A decoder address corresponds to 4 signal addresses.
The signal decoder occupies 4 decoder addresses in a row and thus 4x4=16 signal addresses.
First decoder address = 1 ... Signal decoder uses signal addresses 1 to 16
First decoder address = 2 ... Signal decoder uses signal addresses 5 to 20
First decoder address = 3 ... Signal decoder uses signal addresses 9 to 24
and so on...

12 Fast Clock

With **firmware version 1.43**, the possibilities of the already existing LocoNet fast clock have been expanded considerably. Now the accelerated fast clock time of the Z21 is also available to the devices on the main track, X-BUS, and LAN. The fast clock can be accelerated by the user up to a rate ≤ 63.

However, the Z21 does not have a real-time clock that would continue to run after switching off the command station. Therefore, the fast clock always begins with the same default start time, which can be set by the user. The user can also configure the behavior in the event of an emergency stop and short-circuit, as well as the output on the track, LocoNet, X-BUS and IP Multicast.

- For DCC model time messages on the main track see RCN-211.
- On the LocoNet, the terminal equipment can poll the so-called clock slot (0x7B) every 70 to 100 seconds. See also LocoNet Spec, e.g., personal edition for learning purposes.
- On the X-BUS, the fast clock time is reported according to XpressNet™ V4.0 once per model minute.
- On the LAN interface, the fast clock time can optionally also be sent by using "MRclock" multicast messages. This allows the use of MRclock clients such as the Android MRclock app to display the fast clock time. If enabled, the MRclock multicast is then sent once per model minute (but at least three times per real minute) to the multicast address 239.50.50.20, port 2000.

Finally, there are also Z21 LAN commands for the fast clock, which are now described as follows.

12.1 LAN_FAST_CLOCK_CONTROL

12.1.1 Get Fast Clock Time

The following command can be used to read out the current fast clock time.

Request to Z21:

DataLen		**Header**		**Data**		
0x07	0x00	**0xCC**	0x00	0x21	0x2A	0x0B

Reply from Z21:
See below ***12.2*** LAN_FAST_CLOCK_DATA.

12.1.2 Set Fast Clock Time

The following command can be used to set the current fast clock rate and time to a desired point in time.

Request to Z21:

DataLen		**Header**		**Data**					
0x0A	0x00	**0xCC**	0x00	0x24	0x2B	DDDhhhhh	00mmmmmm	00rrrrrr	XOR-Byte

DDD The desired day of the week in 3 bits.
Value ranges from 0 = Monday to 6 = Sunday.

hhhhh The hour in 5 bits, value range 0 to 23.

mmmmmm The minute in 6 bits, value range 0 to 59.

rrrrrr The desired fast clock rate (acceleration factor) in 6 bits.
Value ranges from 0 to 63:
(0 ... fast clock stands still. Not recommended, better use ***12.1.4*** Stop Fast Clock Time)
1 ... real time
2 ... model time runs twice as fast
3 ... model time runs three times as fast
and so on…
Note: The fast clock rate is stored persistently in the Z21.

XOR-Byte XOR checksum across Data

Reply from Z21:
12.2 LAN_FAST_CLOCK_DATA to subscribed clients.

12.1.3 Start Fast Clock Time

The following command can be used to resume the fast clock operation.

Request to Z21:

DataLen		**Header**		**Data**		
0x07	0x00	**0xCC**	0x00	0x21	0x2C	0x0D

Reply from Z21:
12.2 LAN_FAST_CLOCK_DATA to subscribed clients.

Note: The changed status "fcFastClockEnabled" is stored persistently in the Z21.

12.1.4 Stop Fast Clock Time

The following command can be used to pause the fast clock operation.

Request to Z21:

DataLen		**Header**		**Data**		
0x07	0x00	**0xCC**	0x00	0x21	0x2D	0x0C

Reply from Z21:
12.2 LAN_FAST_CLOCK_DATA to subscribed clients.

Note: The changed status "fcFastClockEnabled" is stored persistently in the Z21.

12.2 LAN_FAST_CLOCK_DATA

Reports current fast clock time to clients. This message is reported asynchronously from the Z21 to the client, when the client

- has activated the corresponding broadcast, see ***2.16*** LAN_SET_BROADCASTFLAGS, Flag 0x00000010, or
- has explicitly requested the fast clock time, see above ***12.1.1*** Get Fast Clock Time.

When the fast clock is running, the Z21 reports this message asynchronously to the subscribed clients about once per model minute, but also when the fast clock is started, paused, or changed by the user.

The Z21 may also omit fast clock time messages, e.g., if it is not running in normal operation. Skipped time messages must be tolerated by the clients and, if necessary, can be further calculated by the clients themselves based on the acceleration factor known from the previous time message.

Z21 to Client:

DataLen		**Header**		**Data**
0x0C	0x00	**0xCD**	0x00	**FastClockTime** (8 Bytes)

FastClockTime is structured as follows:

Byte Offset	**Typ**	**Name**	**Wert**	
0	UINT8		0x66	
1	UINT8		0x25	
2	UINT8	**DDDh hhhh**		fast clock day of the week and hour
3	UINT8	00**mm mmmm**		fast clock minute
4	UINT8	**SH**ss ssss		fast clock second, including STOP and HALT flag
5	UINT8	00**rr rrrr**		fast clock rate
6	UINT8	**FcSettings**		fast clock settings flags
7	UINT8	XOR-Byte		XOR checksum across Data

DDD The current fast clock day of the week in 3 bits. Value range 0 = Monday to 6 = Sunday.

hhhhh The current fast clock hour in 5 bits, value range 0 to 23.

mmmmmm The current fast clock minute in 6 bits, value range 0 to 59.

S **STOP flag:** the fast clock is not running.
The reason may be that the fast clock is not enabled, or the rate = 0, etc.

H **HALT flag:** the fast clock has been temporarily paused.
The reason can be an emergency stop or a short circuit on the main track.

ssssss The current fast clock second in 6 bits, value range 0 to 59.

rrrrrr The current fast clock rate (acceleration factor) in 6 bits, value range 0 to 63:
(0 ... fast clock cannot run)
1 ... real time
2 ... model time runs twice as fast
3 ... model time runs three times as fast
and so on.

FcSettings The persistent fast clock settings flags, bit-coded.
Meaning of the bits see ***12.3*** LAN_FAST_CLOCK_SETTINGS_GET.

12.3 LAN_FAST_CLOCK_SETTINGS_GET

The following command can be used to read the persistent fast clock settings.

Request to Z21:

DataLen		Header		Data
0x05	0x00	**0xCE**	0x00	0x04

Reply from Z21:

DataLen		Header		Data			
0x08	0x00	**0xCE**	0x00	**FcSettings**	**Rate**	**StartDDDhhhhh**	**StartMMMMMM**

The individual parameters in Data are each 8 bits wide.

FcSettings The fast clock settings flags, bit-coded, see below.

Rate The desired fast clock rate (acceleration factor).
Value range 0 to 63:
(0 ... fast clock cannot run, not recommended)
1 ... real time
2 ... model time runs twice as fast
3 ... model time runs three times as fast
and so on.

StartDDDhhhhh Default start time: weekday and hour when switching on the command station.
DDD is the day of the week in 3 bits. Value range 0 = Monday to 6 = Sunday.
hhhhh is the hour in 5 bits, value range 0 to 23.

StartMMMMMM Default start time: minute when switching on the command station.
MMMMMM is the minute in 6 bits, value range 0 to 59

Bitmasks for **Fcsettings**:

```
#define fcFastClockLocoNetEn           0x01 // activate output on LocoNet (polled)
#define fcFastClockXBUSEn              0x02 // activate broadcast on XBUS
//                                     0x04 // reserved
#define fcFastClockDCCEn               0x08 // activate DCC broadcast on main track
#define fcFastClockMRclockEn           0x10 // enable sending MRclock multicasts
//                                     0x20 // reserved
#define fcFastClockEmergenyHaltEn      0x40 // halt fast clock on emergency stop aut.
#define fcFastClockEnabled             0x80 // activate fast clock
```

All bits declared here as "reserved" are reserved for future extensions and should neither be evaluated nor changed.

The flag **fcFastClockEmergenyHaltEn** causes the fast clock to be automatically paused during an emergency stop or short circuit.

The bit **fcFastClockEnabled** is *the* enable flag for the fast clock. Similar to **Rate,** it is not only changed via the LAN_FAST_CLOCK_SETTINGS_SET command described below, but also indirectly via LAN_FAST_CLOCK_CONTROL, i.e., by starting or stopping the fast clock.

The **factory defaults** are FcSettings=0x4F, Rate=1, StartDDDhhhhh=0, and StartMMMMMM=0.

12.4 LAN_FAST_CLOCK_SETTINGS_SET

The persistent fast clock settings can be specifically overwritten with the following commands.
The individual parameters in Data are each 8 bits wide.

Request to Z21:

DataLen		**Header**		**Data**
0x05	0x00	**0xCF**	0x00	**FcSettings**

This will only overwrite the fast clock settings flags **FcSettings**.

Request to Z21:

DataLen		**Header**		**Data**	
0x06	0x00	**0xCF**	0x00	**FcSettings**	**Rate**

This will only overwrite the fast clock settings flags **FcSettings** and the fast clock **Rate**.

Request to Z21:

DataLen		**Header**		**Data**			
0x08	0x00	**0xCF**	0x00	**FcSettings**	**Rate**	**StartDDDhhhhh**	**StartMMMMMM**

This overrides the fast clock settings flags **FcSettings,** the fast clock **Rate** and the **default start time**.
The default start time is the time that is adopted when the Z21 is powered on.

Description of the individual fields see above ***12.3*** LAN_FAST_CLOCK_SETTINGS_GET.

Reply from Z21:
None.

Appendix A – Command overview

Client to Z21

These messages can be sent from a client to a Z21 or zLink device.

Header	Data			Name	LAN		zLink	
	X-Header	DB0	Parameter		Z21 Z21 XL	z21 z21start	Booster 10806 10807 10869	Decoder 10836 10837
0x10	-			LAN_GET_SERIAL_NUMBER	✓	✓	✓	✓
0x18	-			LAN_GET_CODE	✓	✓	×	×
0x1A	-			LAN_GET_HWINFO	✓	✓	✓	✓
0x30	-			LAN_LOGOFF	✓	✓	✓	✓
0x40	0x21	0x21	-	LAN_X_GET_VERSION	✓	✓	✓	✓
0x40	0x21	0x24	-	LAN_X_GET_STATUS	✓	✓	✓	✓
0x40	0x21	0x80	-	LAN_X_SET_TRACK_POWER_OFF	✓	✓	✓	✓
0x40	0x21	0x81	-	LAN_X_SET_TRACK_POWER_ON	✓	✓	✓	✓ (4)
0x40	0x22	0x11	Register	LAN_X_DCC_READ_REGISTER	✓	✓	×	×
0x40	0x23	0x11	CV-Address	LAN_X_CV_READ	✓	✓	×	✓
0x40	0x23	0x12	Register, Value	LAN_X_DCC_WRITE_REGISTER	✓	✓	×	×
0x40	0x24	0x12	CV-Address, Value	LAN_X_CV_WRITE	✓	✓	×	✓
0x40	0x24	0xFF	Register, Value	LAN_X_MM_WRITE_BYTE	✓	✓	×	×
0x40	0x43	Turnout address		LAN_X_GET_TURNOUT_INFO	✓	✓	×	✓
0x40	0x44	Accessory decoder address		LAN_X_GET_EXT_ACCESSORY_INFO	✓	✓	×	✓ (3)
0x40	0x53	Turnout address, command		LAN_X_SET_TURNOUT	✓	✓ (1)	×	✓
0x40	0x54	Accessory decoder address, State		LAN_X_SET_EXT_ACCESSORY	✓	✓ (1)	×	✓
0x40	0x80	-		LAN_X_SET_STOP	✓	✓	×	✓ (5)
0x40	0x92	Loco address		LAN_X_SET_LOCO_E_STOP	✓	✓	×	×
0x40	0xE3	0x44	Loco address	LAN_X_PURGE_LOCO	✓	✓	×	×
0x40	0xE3	0xF0	Loco address	LAN_X_GET_LOCO_INFO	✓	✓	×	×
0x40	0xE4	0x1s	Loco address, Speed	LAN_X_SET_LOCO_DRIVE	✓	✓ (1)	×	×
0x40	0xE4	0xF8	Loco address, Function	LAN_X_SET_LOCO_FUNCTION	✓	✓ (1)	×	×
0x40	0xE4	Group	Loco address, Function group	LAN_X_SET_LOCO_FUNCTION_GROUP	✓	✓ (1)	×	×
0x40	0xE4	0x5F	Loco address, Binary state	LAN_X_SET_LOCO_BINARY_STATE	✓	✓	×	×
0x40	0xE6	0x30	POM-Param, Option 0xEC	LAN_X_CV_POM_WRITE_BYTE	✓	✓	×	✓
0x40	0xE6	0x30	POM-Param, Option 0xE8	LAN_X_CV_POM_WRITE_BIT	✓	✓	×	×
0x40	0xE6	0x30	POM-Param, Option 0xE4	LAN_X_CV_POM_READ_BYTE	✓	✓	×	✓
0x40	0xE6	0x31	POM-Param, Option 0xEC	LAN_X_CV_POM_ACCESSORY_WRITE_BYTE	✓	✓	×	✓
0x40	0xE6	0x31	POM-Param, Option 0xE8	LAN_X_CV_POM_ACCESSORY_WRITE_BIT	✓	✓	×	×
0x40	0xE6	0x31	POM-Param, Option 0xE4	LAN_X_CV_POM_ACCESSORY_READ_BYTE	✓	✓	×	✓
0x40	0xF1	0x0A	-	LAN_X_GET_FIRMWARE_VERSION	✓	✓	✓	✓
0x50	Broadcast-Flags			LAN_SET_BROADCASTFLAGS	✓	✓	✓	✓
0x51	-			LAN_GET_BROADCASTFLAGS	✓	✓	✓	✓
0x60	Loco address			LAN_GET_LOCOMODE	✓	✓	×	×
0x61	Loco address, Mode			LAN_SET_LOCOMODE	✓	✓	×	×
0x70	Accessory decoder address			LAN_GET_TURNOUTMODE	✓	✓	×	×
0x71	Accessory decoder address, Mode			LAN_SET_TURNOUTMODE	✓	✓	×	×
0x81	Group index			LAN_RMBUS_GETDATA	✓	✓	×	×
0x82	Address			LAN_RMBUS_PROGRAMMODULE	✓	✓	×	×
0x85	-			LAN_SYSTEMSTATE_GETDATA	✓	✓	×	×
0x89	Address			LAN_RAILCOM_GETDATA	✓	✓	✓	×
0xA2	LocoNet message			LAN_LOCONET_FROM_LAN	✓	✓ (1)(2)	×	×
0xA3	Loco address			LAN_LOCONET_DISPATCH_ADDR	✓	×	×	×
0xA4	Type, Report address			LAN_LOCONET_DETECTOR	✓	✓ (2)	×	×
0xC4	Type, NId			LAN_CAN_DETECTOR	✓	×	×	×
0xC8	NetID			LAN_CAN_DEVICE_GET_DESCRIPTION	✓	×	×	×
0xC9	NetID, Name			LAN_CAN_DEVICE_SET_DESCRIPTION	✓	×	×	×
0xCB	NetID, PowerState			LAN_CAN_BOOSTER_SET_TRACKPOWER	✓	×	×	×
0xCC	Fastclock Start/Stop/Get/Set Command			LAN_FAST_CLOCK_CONTROL	✓	✓	×	×
0xCE	Len			LAN_FAST_CLOCK_SETTINGS_GET	✓	✓	×	×
0xCF	Fastclock Settings			LAN_FAST_CLOCK_SETTINGS_SET	✓	✓	×	×
0xB2	BoosterPort, BoosterPowerState			LAN_BOOSTER_SET_POWER	×	×	✓	×
0xB8	-			LAN_BOOSTER_GET_DESCRIPTION	×	×	✓	×
0xB9	String			LAN_BOOSTER_SET_DESCRIPTION	×	×	✓	×
0xBB	-			LAN_BOOSTER_SYSTEMSTATE_GETDATA	×	×	✓	×
0xD8	-			LAN_DECODER_GET_DESCRIPTION	×	×	×	✓
0xD9	String			LAN_DECODER_SET_DESCRIPTION	×	×	×	✓
0xDB	-			LAN_DECODER_SYSTEMSTATE_GETDATA	×	×	×	✓
0xE8	0x06	-		LAN_ZLINK_GET_HWINFO	×	×	✓ (6)	✓ (6)

Table 1: Messages from Client to Z21

(1) z21start: fully functional only with activation code (order number 10814 or 10818)
(2) z21, z21start: virtual LocoNet stack (for example GBM16XN with XPN interface)
(3) from decoder FW V1.11
(4) Decoder: Turn on signal lamps again (10837 only)
(5) Decoder: shows stop aspect if the second bit (0x02) is set in CV38 (10837 only)
(6) Answered by the 10838 Z21 pro LINK, not by its terminal device (booster or decoder)

Z21 to Client

These messages can be sent to a client from a Z21 or zLink device.

Header	Data			Name	LAN		zLink	
	X-Header	DB0	Daten		Z21 Z21 XL	z21 z21start	Booster 10806 10807 10869	Decoder 10836 10837
0x10	Serialnumber			Reply to LAN_GET_SERIAL_NUMBER	✓	✓	✓	✓
0x18	Code			Reply to LAN_GET_CODE	✓	✓	×	×
0x1A	HWType, FW Version (BCD)			Reply to LAN_GET_HWINFO	✓	✓	✓	✓
0x40	0x43	Turnout information		LAN_X_TURNOUT_INFO	✓	✓ (1)	×	✓
0x40	0x44	Accessory state information		LAN_X_EXT_ACCESSORY_INFO	✓	✓ (1)	×	✓ (3)
0x40	0x61	0x00	-	LAN_X_BC_TRACK_POWER_OFF	✓	✓	✓	×
0x40	0x61	0x01	-	LAN_X_BC_TRACK_POWER_ON	✓	✓	✓	×
0x40	0x61	0x02	-	LAN_X_BC_PROGRAMMING_MODE	✓	✓	×	×
0x40	0x61	0x08	-	LAN_X_BC_TRACK_SHORT_CIRCUIT	✓	✓	× (4)	× (4)
0x40	0x61	0x12	-	LAN_X_CV_NACK_SC	✓	✓	×	×
0x40	0x61	0x13	-	LAN_X_CV_NACK	✓	✓	×	✓
0x40	0x61	0x82	-	LAN_X_UNKNOWN_COMMAND	✓	✓	✓	✓
0x40	0x62	0x22	State	LAN_X_STATUS_CHANGED	✓	✓	✓	✓
0x40	0x63	0x21	XBus Version, ID	Reply to LAN_X_GET_VERSION	✓	✓	✓	✓
0x40	0x64	0x14	CV-Result	LAN_X_CV_RESULT	✓	✓	×	✓
0x40	0x81	-		LAN_X_BC_STOPPED	✓	✓	×	×
0x40	0xEF	Loco information		LAN_X_LOCO_INFO	✓	✓ (1)	×	×
0x40	0xF3	0x0A	Version (BCD)	Reply to LAN_X_GET_FIRMWARE_VERSION	✓	✓	✓	✓
0x51	Broadcast-Flags			Reply to LAN_GET_BROADCASTFLAGS	✓	✓	✓	✓
0x60	Loco address, Mode			Reply to LAN_GET_LOCOMODE	✓	✓	×	×
0x70	Accessory decoder address, Mode			Reply to LAN_GET_TURNOUTMODE	✓	✓	×	×
0x80	Group index, Feedback status			LAN_RMBUS_DATACHANGED	✓	✓	×	×
0x84	SystemState			LAN_SYSTEMSTATE_DATACHANGED	✓	✓	×	×
0x88	RailCom data			LAN_RAILCOM_DATACHANGED	✓	✓	✓	×
0xA0	LocoNet-Meldung			LAN_LOCONET_Z21_RX	✓	×	×	×
0xA1	LocoNet-Meldung			LAN_LOCONET_Z21_TX	✓	✓ (2)	×	×
0xA2	LocoNet-Meldung			LAN_LOCONET_FROM_LAN	✓	✓ (2)	×	×
0xA3	Loco address, Ergebnis			LAN_LOCONET_DISPATCH_ADDR	✓	×	×	×
0xA4	Type, Feedback address,Info			LAN_LOCONET_DETECTOR	✓	✓ (2)	×	×
0xC4	Occupancy message			LAN_CAN_DETECTOR	✓	×	×	×
0xC8	NetID, Name			Reply to LAN_CAN_DEVICE_GET_DESCRIPTION	✓	×	×	×
0xCA	CANBoosterSystemState			LAN_CAN_BOOSTER_SYSTEMSTATE_CHGD	✓	×	×	×
0xCD	Fastclock Time			LAN_FAST_CLOCK_DATA	✓	✓	×	×
0xCE	Fastclock Settings			LAN_FAST_CLOCK_SETTINGS_GET	✓	✓	×	×
0xB8	String			Reply to LAN_BOOSTER_GET_DESCRIPTION	×	×	✓	×
0xBA	BoosterSystemState			LAN_BOOSTER_SYSTEMSTATE_DATACHANGED	×	×	✓	×
0xD8	String			Reply to LAN_DECODER_GET_DESCRIPTION	×	×	×	✓
0xDA	DecoderSystemState			LAN_DECODER_SYSTEMSTATE_DATACHANGED	×	×	×	✓
0xE8	0x06	Z_Hw_Info		Reply to LAN_ZLINK_GET_HWINFO	×	×	✓ (5)	✓ (5)

Table 2: Messages from Z21 to Clients

(1) z21start: fully functional only with activation code (order number 10814 or 10818)
(2) z21, z21start: virtual LocoNet stack (for example GBM16XN with XPN interface)
(3) from decoder FW V1.11
(4) Short-circuit is reported in the corresponding booster/decoder system state.
(5) Answered by the 10838 Z21 pro LINK, not by its terminal device (booster or decoder)

List of figures

List of tables